Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Entwicklung einer Mobilen Anwendung mit Flutter SDK zur Planung der Studienvorbereitung der indonesischen Studierenden in Deutschland

Bachelorarbeit

Fachbereich 2

Ingenieurwissenschaften – Technik und Leben

Im Studiengang Ingenieurinformatik

Hochschule für Technik und Wirtschaft Berlin

Autor:	Adib Ghassani Waluya
Version vom:	04.01.2022
Erstgutacher:	Prof. Dr.-Ing. Frank Neumann
Zweitgutacher:	Prof. Dr. Frank Burghardt

Vorwort

Diese Arbeit wurde an der Hochschule für Technik und Wirtschaft Berlin geschrieben. Die Idee und das Konzept, diese mobile Anwendung zu entwickeln, wurden durch die Erfahrungen vieler indonesischer Studierenden inspiriert, die bei der Vorbereitung ihres Studiums in Deutschland vor zahlreichen Herausforderungen standen.

Zuallererst möchte ich mich bei Herrn Prof. Dr.-Ing. Frank Neumann für seine Geduld und die Zeit bedanken, die er für hilfreiches und kritisches Feedback während der gesamten Betreuungszeit aufgebracht hat. Zweitens möchte ich auch bei Herrn Prof. Dr. Frank Burghardt für seine Bereitschaft danken, diese Arbeit zu betreuen.

Ich möchte auch meinen besten Freunden, Citra Fauzan Alqodri und Hakeem Nanda Saputra, für ihre unterschiedlichen Perspektiven bei der Lösung einiger der Herausforderungen danken, auf die ich bei der Entwicklung des Systems gestoßen bin.

Mein besonderer Dank gilt schließlich meiner Familie für ihre unerschütterliche moralische Unterstützung während des Schreibens dieser Arbeit. Ohne ihre Unterstützung und ihren Beistand wäre alles sicherlich viel komplizierter gewesen.

Adib Ghassani Waluya

Berlin, 31.12.2021

Kurzzusammenfassung

Aufgrund der Globalisierung ist ein Studium im Ausland eine beliebte Option für indonesische Studienbewerber geworden, um globale Fähigkeiten und eine bessere Ausbildung zu erwerben. Deutschland ist eines der beliebtesten Ziele, da es für seine Spitzenuniversitäten bekannt ist und der Bildung einen hohen Stellenwert einräumt, wie das hochwertige Bildungssystem des Landes zeigt. Das ist der Hauptgrund, warum die Zahl der indonesischen Studienbewerber in den letzten fünf Jahren gestiegen ist. Daher ist es für die Bewerber wichtig, den gesamten Prozess der Studienvorbereitung zu verstehen.

Ziel dieser Arbeit ist es, eine mobile App mit Flutter SDK zu entwickeln. Die App soll als eine Plattform dienen, um vereinfachte, aber gründliche Informationen über den Studienvorbereitungsprozess bereitzustellen. Darüber hinaus kann die App auch als Planer benutzt werden, mit dem der Vorbereitungsprozess des Bewerbers verfolgt werden kann. Um den Bewerber bei der Überwindung der Vorbereitungshindernisse zu unterstützen und ihm zu helfen, sollte ein Beratungsservice in Form eines Live-Chats in die App integriert werden.

Um den Anforderungen der oben genannten App gerecht zu werden, wird die Implementierung von Flutter SDK für die UI-Entwicklung der App und Laravel Framework für das Backend der App eingehend besprochen. Außerdem werden die notwendigen Plugins zur Erfüllung der Anforderungen der App analysiert. Anschließend kann die App getestet und bewertet werden. Zusammenfassend lässt sich sagen, dass Flutter SDK eine praktikable Option für die effiziente Entwicklung mobiler Apps ist.

Abstract

Due to the globalization, studying abroad has become a popular option for the Indonesian study applicants to acquire global skills and better education. Germany is one of the favorite destinations as it is well known for its top-ranked universities and it places a high emphasis on education, as seen by the country's high-quality education system. It is the primary reason why the number of Indonesian study applicants has been increasing over the last five years. Therefore, it is necessary for the applicants to understand the whole study preparation process.

The aim of this thesis is to develop a mobile app, more specifically an Android app developed using Flutter SDK, that serves as a platform to provide simplified yet thorough information of the study preparation process. Furthermore, the app can also be used as a planer which can be utilized to track the applicant's preparation process. To assist and help the applicant on overcoming the preparation obstacles, consultation service in form of live chat should be implemented within the app.

To answer the above-mentioned app's requirements, the implementation of Flutter SDK for the app's UI development and Laravel Framework for the app's backend will be discussed intensively. Furthermore, the necessary plugins will be analyzed to fulfill the app's requirements. Subsequently, the app can be tested and evaluated. In summary, Flutter SDK proves to be a viable option for the efficient mobile app development.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis

Abkürzungen

Abkürzung	Bedeutung
AOT	Ahead-Of-Time
API	Programmierschnittstelle (englisch *Application Programming Interface*)
APK	Android Package
BPMN	Geschäftsprozessmodell und -notation (englisch *Business Process Model and Notation*)
CRUD	Create, Read or Retrieve, Update, Delete or Destroy
CSS	gestufte Gestaltungsbögen (englisch *Cascading Style Sheets*)
DAAD	Deutscher Akademischer Austauschdienst (englisch *German Academic Exchange Service*)
DOM	Document Object Model
FPS	frames per second
GPS	Globales Positionsbestimmungssystem (englisch *Global Positioning System*)
HMAC	Hash-basierter Nachrichtenauthentifizierungscode (englisch *Hash-based Message Authentication Code*)
HTML	Hypertext-Auszeichnungssprache (englisch *HyperText Markup Language*)
HTTPS	sicheres Hypertext-Übertragungsprotokoll (englisch *Hypertext Transfer Protocol Secure*)
JIT	Just-In-Time
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
MVC	Model-View-Controller
ORM	Objektrelationale Abbildung (englisch *Object-Relational Mapping*)
PDF	Portable Document Format
PHP	Hypertext Preprocessor
RSA	Rivest-Shamir-Adleman
SDK	Software Development Kit
SHA	sicherer Hash-Algorithmus (englisch *Secure Hash Algorithm*)
SQL	Strukturierte Abfrage-Sprache (englisch *Structured Query Language*)
UI	Benutzerschnittstelle/Benutzeroberfläche (englisch *User Interface*)
UNESCO	Organisation der Vereinten Nationen für Bildung, Wissenschaft und Kultur (englisch *United Nations Educational, Scientific and Cultural Organization*)
URL	einheitlicher Ressourcenzeiger (englisch *Uniform Resource Locator*)
WAMP	Windows, Apache, MySQL, and PHP
XAMPP	die Anfangsbuchstaben von X für alle beliebigen Betriebssysteme, Apache, MySQL, PHP und Perl.

1. Einleitung

1.1. Problemstellung

In den letzten 6 Jahren ist die Zahl der indonesischen Studierenden, die im Ausland studieren, um 21% gestiegen, basierend auf eine Schätzung der Organisation der Vereinten Nationen für Bildung, Wissenschaft und Kultur (UNESCO) [1], während andere Daten darauf hindeuten, dass diese Zahl erheblich höher war. Indonesier ziehen nun ein breiteres Spektrum an Auslandsstudienzielen in Betracht, darunter auch Deutschland. Nach Angaben des Deutschen Akademischen Austausch Dienstes (DAAD) waren 5.133 indonesische Studierende im Sommersemester 2018 an deutschen Hochschulen und Universitäten eingeschrieben [2]. Diese Zahl entspricht etwa 3,8% aller asiatischen Studierenden in Deutschland und nimmt Indonesien den Platz 6 nach China, Indien, Iran, Vietnam und Südkorea ein [2].

Trotz dieser steigenden Zahl gibt es jedes Jahr ähnliche Herausforderungen, denen indonesische Studierende sich stellen müssen, bevor sie an deutschen Universitäten eingeschrieben werden. Diese Schwierigkeiten sind die Beschaffung aktueller und vollständiger Informationen über die Studienvorbereitung (eine gründliche Schritt-für-Schritt-Vorbereitung), das Budget und die Suche nach einer Unterkunft. Die Regelungen zur Studienvorbereitung selbst ändern sich, was es noch schwieriger macht, die aktuellen Informationen zu bekommen. Bisher ist die effektivste Lösung zur Überwindung dieser Hindernisse die Konsultation mit einem Bildungsberater, was ca. Rp. 50.000.000 - Rp. 100.000.000 (3.087 Euro – 6.175 Euro, Rp. 16.193,65 = 1 Euro[1]) kosten kann.

Obwohl die Regeln ständig ändern, bleiben die Schritte der Studienvorbereitung bisher unverändert. Das heißt, die Reihenfolge der Vorbereitungsschritte ist jedes Jahr immer das Gleiche. Aus diesem Grund gibt es viele Studierende, sie die Ausreise eigenständig vorbereiten, um den enorm hohen Kostenaufwand zu vermeiden. Dennoch berichteten viele von ihnen, dass sie aufgrund der Informationsüberflutung zur Studienvorbereitung und der Fehlkalkulationen im Zeitmanagement ein Trial-and-Error-Szenario erleben müssen, das nicht nur Geld, sondern auch Zeit kosten könnte.

[1] 3. Januar 2021, 00:49 UTC

1.2. Ziel der Arbeit

Trotz des beträchtlichen Anstiegs der Zahl der indonesischen Studierenden, die in Deutschland studieren möchten, gibt es bis heute keine mobile Anwendung, die eine Lösung zur Vereinfachung der Studienvorbereitung bietet. Deshalb wird im Rahmen dieser Arbeit eine einfache, kostengünstige und praktische Mobilanwendung entwickelt und getestet, die eine zentrale, unkomplizierte, aktuelle und gründliche Anleitung zur schrittweisen Studienvorbereitung enthält. Dieser Leitfaden kann in Form von Notizen sein. Diese mobile Anwendung sollte auch als Zeitplaner fungieren, um den Studierenden zu helfen, die wichtigen Termine oder Verabredungen zu organisieren, z.B. den Termin für die Beantragung des Visums, die notarielle Beglaubigung der erforderlichen Dokumente, usw. Der Planer gibt auch Auskunft über die übliche Zeiteinschätzung der Studienvorbereitung (z.B. die Anzeige des Enddatums der Vorbereitung). Die On-Demand-Beratung in Form eines Live-Chats wird entwickelt, um den Studierenden zu helfen, einige der Vorbereitungshindernisse zu überwinden.

In Zukunft sollte die App für ein breiteres Publikum zugänglich sein. Daher sollte diese mobile App in ihrer weiteren Entwicklung sowohl für verschiedene mobile Plattformen (Android und iOS) als auch für das Web verfügbar sein. Für die Zwecke dieser Arbeit wird jedoch nur die Android-Version entwickelt und getestet. Dies ist der Hauptgrund für die Implementierung des Flutter SDK, da es für die effiziente plattformübergreifende Entwicklung mobiler Apps mit nur einer einzigen Codebasis vorzuziehen ist. Des Weiteren sollte eine relationale Datenbank für den Backend-Dienst der App implementiert werden, weil die App in der zukünftigen Entwicklung eine komplexe Datenbankstruktur beinhaltet. Um die Anforderungen zu klären, werden erstens die theoretische Grundlage erörtert, die den nötigen Tech Stack enthält und später bei der Entwicklung der Applikation notwendig ist. Anschließend wird der Umfang des Entwicklungsprozesses anhand der Liste der Softwareanforderungen erläutert. Abschließend wird der Prozess der Anwendungsentwicklung analysiert und dokumentiert und eine Evaluation zur Darstellung der Stärken und Schwachstellen dieser App durchgeführt.

2. Grundlagen

In diesem Kapitel wird zunächst der Stand der Technik der nötigen Lösungsansätzen erklärt. Danach werden die theoretische Grundlage zur Entwicklung einer Android-Mobileapplikation sowie Flutter-SDK als die durchführbare Lösung für die plattformübergreifende App-Entwicklung beschrieben. Die Verwendung von Laravel als serverseitiges PHP-Framework, werden nachher erarbeitet.

2.1. Stand der Technik

Im letzten Jahrzehnt ist die rasche Verbreitung mobile Geräte beispiellos. Innovation und Digitalisierung sind mehrere der treibenden Faktoren, die zu dieser weiten Verbreitung beigetragen haben. Die Digitalisierung prägt bereits die Art und Weise, wie wir leben und kommunizieren. Daher nimmt die Entwicklung mobiler Apps täglich zu und die Wichtigkeit der Apps wird im Laufe der Zeit immer größer. Der berühmte Slogan von Apple im Jahr 2009, “Es gibt für fast alles eine App”, hat auch heute noch seine Gültigkeit. Das beweist auch die Menge der heute zur Verfügung stehenden kostenlosen mobilen Entwicklungstools. Beispielsweise werden Android- und iOS-SDK für Native App-Entwicklung verwendet. Darüber hinaus gibt es auch viele Open-Source-SDK, die für die Entwicklung plattformübergreifender Anwendungen geeignet sind, z.B. Flutter-SDK und React Native, die bei der App-Entwicklung ihre eigenen Vorteile haben.

Aus diesem Grund stellt sich vor der Entwicklung einer App normalerweise die Frage, mit welcher Technologie die App umgesetzt werden soll. Dieser Abschnitt beschreibt den aktuellen Stand der Technik der grundlegenden Vorgehensweisen bei der mobilen Entwicklung, wie zum Beispiel Native Entwicklung, Hybride- oder Cross-Plattform Apps und Web-Apps.

In einem bestimmten Stadium der Entwicklung einer mobilen App sollte eine Entscheidung darüber getroffen werden, wie eine App entwickelt werden soll. Heutzutage gibt es drei grundlegende Ansätze, um eine mobile App zu entwickeln: Native Entwicklung, Hybride- und Cross-Plattform Apps sowie Web-Apps [3] [4]. Diese Optionen können ähnliche Funktionalitäten für eine App bieten und jeder Ansatz hat seine eigenen Vor- und Nachteile. Es

ist wichtig, die Unterschiede dieser Vorgehensweisen zu verstehen, bevor man sich für eine bestimmte App-Entwicklung entscheidet, da jeder Ansatz das Endprodukt beeinflussen wird.

Die Apps, die in einer plattformspezifischen Sprache geschrieben wurden und für die Ausführung auf einem bestimmten Betriebssystem konzipiert sind, werden als native Apps bezeichnet [5] [6]. Die spezifischen Sprachen dieser Apps sind vor allem Swift und Objective C bei iOS und Java, Kotlin, sowie C++ bei Android. Native Apps ermöglichen die höchstmögliche Geschwindigkeit und Leistung auf verbundenen Geräten. Neben der Geschwindigkeit und dem verbesserten Benutzererlebnis können viele native Apps auch ohne Internet laufen und Daten lokal auf dem Gerät speichern[2], bis sie mit der Cloud synchronisiert oder wieder mit dem Internet verbunden werden. Ironischerweise ist der größte Vorteil der nativen Entwicklung auch ihr größter Nachteil. Eine native App kann nicht auf einem Gerät ausgeführt werden, das nicht das gleiche Betriebssystem unterstützt. Aufgrund der unterschiedlichen Plattform-Programmiersprache wird den Entwicklern keine Flexibilität geboten, was bedeutet, dass sie jeweils nur für eine Plattform programmieren müssen. Es müssen zwei einzigartige Designs und Codebasen geschrieben werden (getrennte Programmierung für Android und iOS), was die Entwicklung einer nativen App im Allgemeinen teurer macht.

Die Web-App stellt einen anderen Ansatz dar. Es geht um eine auf einem Webserver laufende Anwendung und nur über den Webbrowser aufgerufen werden kann. Sie wird nicht direkt für mobile Geräte programmiert, sondern ist so konzipiert, dass sie responsiv ist. Responsiv bedeutet, dass die App ihre Ansicht anpassen kann, um auf verschiedenen Auflösungen und Ausrichtungen von Mobil- und Tablet-Geräten richtig angezeigt zu werden (vgl. Saleh 2014, S. 4). Web-Apps sind auf HTML5 und CSS angewiesen, um Informationen anzuzeigen und zu formatieren. JavaScript wird implementiert, um die Leistung der App zu verwalten. Zur Entwicklung noch komplexerer Web-Apps werden häufig Frameworks wie Angular und VueJs verwendet. Außerdem müssen Web-Apps nicht die Tests des App Stores bestehen, da sie einfach über den Webbrowser aufgerufen werden können [3]. Für manche mag das ein Vorteil sein, aber für manche kann es auch der größte Nachteil sein, wenn sie die Downloads ihrer App monetarisieren möchten. Auch die Konnektivität oder Internetabhängigkeit ist für Web-Apps von entscheidender Bedeutung, d.h. die Benutzer können nicht auf die App mit allen Funktionen zugreifen, wenn sie offline ist. Zu guter Letzt sind Smartphone-Funktionen wie

[2] Vgl. Trogrlic, Ivan: Benefits of developing native apps, in: DECODE, 2021, https://decode.agency/article/native-app-development-benefits/ (abgerufen am 10.11.2021)

GPS, Kamera und andere Zusatzfunktionen nicht immer gut für responsive/mobile Websites entwickelt [7].

Hybride- und Cross-Plattform-Apps werden als Kompromiss zwischen den beiden oben genannten Ansätzen betrachtet. Viele Menschen nehmen immer noch an, dass plattformübergreifende App- und hybride App-Entwicklung identisch sind [8]. Im Gegenteil, beide spielen eine andere Rolle und haben völlig unterschiedliche Bedeutungen. Eine hybride App ist historisch gesehen ähnlich wie eine Web-App, kombiniert aber die nativen und die Web-Lösungen, indem der in HTML, CSS und JavaScript geschriebene Code mit Hilfe von Plugins wie Apache Cordova, Capacitor von Ionic als native Laufzeitumgebung und dergleichen in eine native App eingebettet wird, um den Zugriff auf die nativen Funktionen zu erhalten. Bei der Entwicklung einer hybriden App wird mit Hilfe eines Frameworks wie Ionic der Code nur einmal geschrieben und derselbe Code kann auf mehreren Plattformen verwendet werden. Die Fähigkeiten des Geräts haben einen erheblichen Einfluss auf das Benutzererlebnis und die Leistung einer Hybrid-App [9], das zur Nutzung der App verwendet wird. Das bedeutet, dass hybride Apps mit immer schnelleren Geräten ein besseres und mit nativen Apps vergleichbares Benutzererfahrung bieten.

Da es sich bei der hybriden App jedoch im Kern um eine Webseite handelt, die in einen nativen Wrapper verpackt ist, kann selbst die makelloseste hybride Anwendung oft kein ausgewogenes und optimales Benutzererfahrung garantieren. Es fehlt die Navigationsmuster, weil nicht alle Geräte der Nutzer alle Navigationselemente enthalten und die meisten Daten direkt vom Hauptserver geladen werden [6]. In diesem Fall befinden sich zwei häufige Probleme, die die Gesamtperformance der App beeinträchtigen können. Erstens die Anzahl der Serveranfragen (wie viele Benutzer gleichzeitig denselben Server anrufen) und zweitens die Lastverteilung (die Anfragen von mobilen Geräten, die den gleichen Server anpingen). Nach Auffassung der Experten ist das DOM (englisch *Document Object Model*), die zur Übermittlung von Informationen zwischen der mobilen Benutzeroberfläche und dem Server verwendet wird, trotz aller Bemühungen möglicherweise nicht schnell und zuverlässig genug für mobile Anwendungen [6].

Die einzige Gemeinsamkeit zwischen hybriden und plattformübergreifenden Anwendungen ist die gemeinsame Nutzung von Code. Während die Hybride-Apps aufgrund ihrer Abhängigkeit vom Gerät, der Softwareversion und der Geschwindigkeit des genutzten Webbrowsers oft träge und nicht besonders reaktionsschnell sind, kennen die Cross-Plattform-Apps solche Probleme

nicht. Cross-Plattform-Frameworks wie zum Beispiel Xamarin, React Native und Flutter SDK werden verwendet, um gemeinsam nutzbaren und wiederverwendbaren Code für die Entwicklung einer App für verschiedene Betriebssysteme zu erstellen. Auf diese Weise können die Entwicklungskosten und der Aufwand minimiert werden. Die Wartung und Bereitstellung des Codes ist ebenfalls einfacher, weil es nur eine einzige Codebasis gibt. Außerdem können Aktualisierungen einfach und schnell über alle Plattformen hinweg synchronisiert werden. Einige Studien deuten jedoch auf einen inhärenten Leistungsverlust bei solchen Anwendungen hin, auch wenn die Endbenutzer dies bei der alltäglichen Nutzung möglicherweise nicht als negativ empfinden [10].

2.2. Flutter SDK

Im Jahr 2015 wurde Flutter SDK, ein neues kostenloses Open-Source UI Software Development Kit, von Google auf dem Dart-Entwickler-Gipfel 2015 (*Dart Developer Summit 2015*) vorgestellt [11]. Früher unter dem Namen Sky bekannt, basiert Flutter auf der objektorientierten Programmiersprache Dart, die von Google mit dem Ziel entwickelt wurde, eine schnelle und reaktionsschnelle App zu erstellen. Dieses Ziel wurde verwirklicht, nachdem es dem Dart-Team mit seiner Demo-App gelungen war, auf einem leistungsfähigen Gerät konstant mit 60 FPS zu rendern. Wenn das Gerät die 120 Hz-Updates unterstützt, kann es sogar 120 FPS ausführen. Zu diesem Punkt lag der von den meisten Entwicklern angestrebte Glättungsstandard bei 60 FPS[3]. Der Fokus des Flutter-Teams hat sich daraufhin weiterentwickelt. Das nächste Ziel von Flutter ist es, ein Framework für die Erstellung von Anwendungen und die Gestaltung des Benutzererlebnisses bzw. *User Experience* auf jedem Gerät oder Formfaktor zu haben, ohne dabei Abstriche bei Qualität und Leistung zu machen. Seit der Veröffentlichung der ersten stabilen Version von Flutter (Flutter 1.0) am 4. Dezember 2018 werden die Multiplattform-Fähigkeiten von Flutter weiter vorangetrieben und die Benutzer sind nun in der Lage, die Apps sowohl für Android als auch für iOS produktiv zu gestalten, ohne dass das zugrunde liegende Framework Einschränkungen mit sich bringt [12]. Die Produktivität der Entwickler wurde durch die Einführung der *Hot-Reload*-Funktion verbessert, weil Dart die JIT-Kompilierung (*Just in Time*) verwendet und auch AOT zu nativem Code kompiliert wird. AOT spielt eine Rolle bei der Sicherstellung der schnellen Startzeiten und der stabilen Leistung für den Release-Modus [13]. Mit dem zustandsabhängigen *Hot-Reload* kann das Ergebnis der Benutzeroberfläche der App nach der Änderung des Codes durch den Benutzer sofort angezeigt werden, ohne dass die App neu gestartet werden muss oder ihren Zustand verliert. Dadurch kann die Entwicklung erheblich beschleunigt werden und es bleibt mehr Raum zum Experimentieren.

Flutter implementiert das deklarative Programmierparadigma. Deklaratives Paradigma bedeutet, dass die Beschreibung des Ablaufs einer Softwarelogik keine Rolle spielt, da sie nur darin besteht, einem Programm mitzuteilen, was erreicht werden muss [14] [15]. Mit anderen Worten: Anstatt die Arbeitsschritte zu definieren, um das Ergebnis zu erreichen, kommt es bei der Verwendung deklarativer Programmiersprachen darauf an, wie das gewünschte

[3] Google Developers: Sky: An Experiment Writing Dart for Mobile (Dart Developer Summit 2015), 2015, [YouTube] https://www.youtube.com/watch?v=PnIWl33YMwA&t=2s, 1:14–1:33.

Endergebnis beschrieben werden soll. Dieses Paradigma prägt den ersten Ansatz der Entwicklung in Flutter, der darin besteht, ein vollständiges Bild oder eine Beschreibung der zu gestaltenden Benutzeroberfläche zu haben. Durch die deklarative Programmierung kann die Komplexität des Codes signifikant reduziert und die Codequalität verbessert werden. Das folgende vereinfachte Beispiel zeigt, wie die Benutzeroberfläche mit imperativem (**Listing 2.1**) und deklarativem Programmierungsstil (**Listing 2.2**) geändert werden kann.

```
1 // Imperativer Stil
2 b.setColor(red)
3 b.clearChildren()
4 ViewC c3 = new ViewC(...)
5 b.add(c3)
```

Listing 2.1: Imperativer Stil [16]

```
1 // Deklarativer Stil
2 return ViewB(
3   color: red,
4   child: ViewC(...),
5 )
```

Listing 2.2: Deklarativer Stil [16]

Anstelle des Abrufs der Instanz b von ViewB mit einer Methode wie z.B. findViewById in Kotlin und des Aufrufs von Mutationen darauf im imperativen Ansatz (**Listing 2.1**), werden in Flutter Widget-Instanzen erstellt. ViewC ist jetzt das Kind des Widgets ViewB und die Farbe Rot ist jetzt deklarativ als Property von ViewB definiert, ohne dass eine andere Methode implementiert werden muss (**Listing 2.2**).

Der deklarative Lösungsansatz bietet den Vorteil, dass es für jeden beliebigen Zustand der Benutzeroberfläche nur einen einzigen Codepfad gibt. Die Benutzeroberfläche von Flutter-Apps wird auf der Basis des aktuellen Status der App erstellt. Das bedeutet, dass die Benutzeroberfläche von Grund auf neu gezeichnet wird, wenn sich der Status (die Daten) ändert. Die Ansichtskonfigurationen in Flutter werden durch die Widgets von Flutter dargestellt. Ein Widget ist ein Element mit einer bestimmten Funktionalität, das zur Beschreibung eines Teils der deklarativen Benutzeroberfläche verwendet wird und keinen veränderbaren Zustand hat (*immutable*) [11]. Um die deklarative Benutzeroberfläche zu ändern, initiieren die entsprechenden Widgets einen Selbstaufbau. Das Widget ist nicht nur auf sichtbare Strukturelemente wie Schaltfläche, Bild oder Text beschränkt, sondern besteht auch

aus unsichtbaren Elementen, die bei der Gestaltung des Layouts helfen, z.B. Padding und Margin. Die Reihenfolge, in der die Widgets für die Gestaltung der App verwendet werden, bestimmt die Ansicht der Benutzeroberfläche. Die Struktur der organisierten Widgets wird als Widget-Baum bzw. *Widget Tree* bezeichnet. Um die Konfiguration der Benutzeroberfläche von Flutter zu verstehen, werden die Systemarchitektur von Flutter in Abschnitt 2.2.1 gründlich beschrieben. Flutter Widgets, Element und RenderObject werden in Abschnitt 2.2.2 ebenfalls separat erklärt.

2.2.1. Flutter Systemarchitektur

Die grundlegende Architektur von Flutter besteht aus zwei wichtigen Teilen:

- Ein Open-Source-Framework mit einer Widget-basierten UI-Bibliothek. Es enthält eine Vielzahl von wiederverwendbaren Elementen der Benutzeroberfläche wie Texteingaben, Schaltflächen usw.,
- *Software Development Kit* (SDK). Eine Anwendung wird durch ein Bündel von Entwicklungswerkzeugen oder SDK erstellt und in nativen Maschinencode kompiliert.

Was die architektonischen Schichten betrifft, so ist Flutter ein mehrschichtiges, erweiterbares System. Die Systeme bestehen aus plattformspezifischen Einbettungen bzw. *embedders*, der Flutter-Engine als Kern der Flutter-Anwendung und dem reaktiven Flutter-Framework, das aus grundlegenden Bibliotheken und einer Reihe von Schichten besteht [17] [18]. Die Komponenten innerhalb des Frameworks können verändert werden (austauschbar) und sind optional. Zum besseren Verständnis werden in **Abb. 2.1** die Architekturschichten von Flutter.

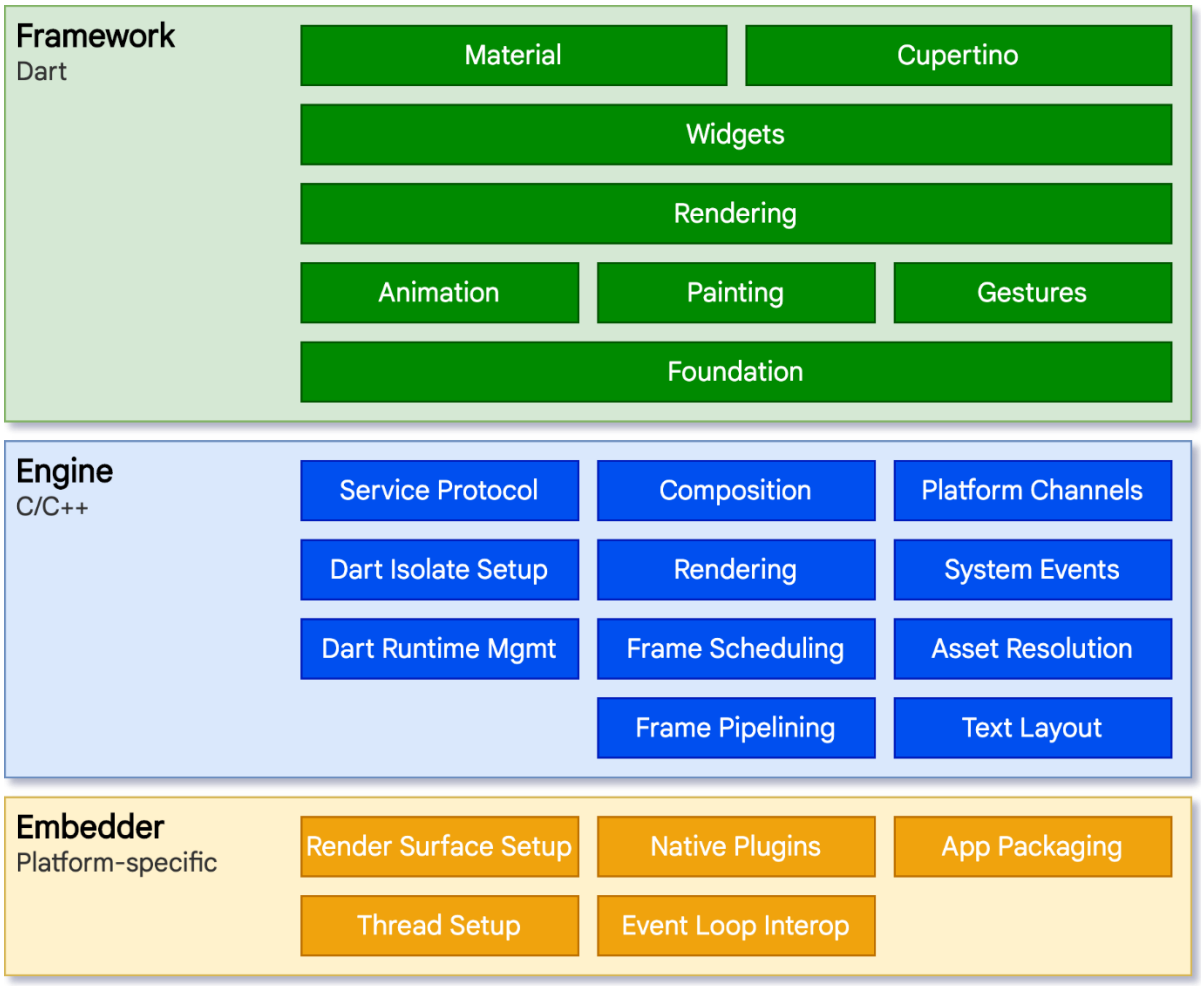

Abb. 2.1: Flutter Architekturebenen [17]

Wie bereits erwähnt, befinden sich die grundlegenden Klassen von Flutter, auch bekannt als dart:ui, und die grundlegenden Bausteindienste, die Animation, *Painting* und Gesten umfassen, auf der ersten Schicht der Architektur. Eine API wird von der Dart UI-Bibliothek verwendet, um direkt mit der Rendering-Engine des Geräts zu kommunizieren. Die Layout-Objekte können in der Rendering-Schicht dynamisch verwendet werden, und anschließend kann ein Baum dieser renderbaren Layout-Objekte, der so genannte RenderObject-Baum, aufgebaut werden. Der RenderObject-Baum spielt eine wichtige Rolle bei der Optimierung der UI-Rendering-Performance, da er eine von drei Instanzen von Bäumen ist, die Flutter während der Build-Phase parallel verwaltet. Die anderen beiden Bäume sind Widget-Baum und Element-Baum. Die Beziehung zwischen diesen drei Bäumen wird in Abschnitt 2.2.2 näher erläutert. Schließlich befinden sich über der Widgets-Schicht Material- und Cupertino-UI-Steuerungsbibliotheken, auch bekannt als designspezifische Widgets, die für die Implementierung von Material- oder iOS-Designsprachen verantwortlich sind. Wenn die Entwickler High-Level-Funktionen implementieren möchten, wie z. B. HTTP-Anfragen oder

das Hinzufügen von Higher-Level-Futures wie der Kamera, werden Pakete bzw. *packages* implementiert.

Im Kern der Flutter-Anwendung ist die Flutter-Engine, die hauptsächlich auf C++ basiert, für die Netzwerkanfragen, die Dateieingabe und -ausgabe verantwortlich und sorgt für das Übersetzungsrendering mit der Skia-Grafikrendering-Bibliothek, indem sie die Pixel der erstellten Grafiken in neu gemalte Rahmen umwandelt und auf dem Bildschirm rendert [17] [19]. Der Prozess der Kompilierung von Dart-Codes mit Hilfe der Dart-Laufzeit- und Kompilierungs-Toolchain wird ebenfalls von der Kern-API von Flutter implementiert. Auf Dienste wie Ereignisschleifen bzw. *event loops*, Zugänglichkeit und Rendering-Oberflächen kann über die Embedders zugegriffen werden, die mit dem zugrunde liegenden Betriebssystem kommunizieren [17] [18]. Die Embedders, mit denen Flutter in verschiedene Plattformen eingebettet werden kann, werden in einer Programmiersprache geschrieben, die sich nach der jeweiligen Plattform richtet. C++ und Java werden für Android verwendet, Objective C/Objective C++ für iOS und macOS und schließlich C++ für Windows und Linux.

2.2.2. Flutter Widget, Element und RenderObject

Wie bereits kurz erläutert, ist ein Widget die Kernkomponente oder ein Grundbaustein der Flutter-Benutzeroberfläche. Es kann auch als die Ansichtsanweisungen oder Konfigurationen für verschiedene Komponenten der Benutzeroberfläche beschrieben werden [11]. Diese Ansichtskonfigurationen enthalten endgültige Eigenschaften, die nicht geändert werden können oder unveränderlich (*immutable*) sind. Ein Widget wird als leichtgewichtiger *Blueprint* bezeichnet, weil es nur Daten enthält, keine Referenzen hat und bei Änderungen der Benutzeroberfläche kostengünstig neu erzeugt werden kann. Trotz der Unveränderlichkeit kann der Benutzer die Widgets frei ersetzen, entfernen oder neu anordnen. Die häufige Verwendung der Konfigurationsterminologie für Widgets kann manchmal Verwirrung stiften, da sie vage klingen kann. Im Wesentlichen bedeutet sie, dass alle in dem Widget gekapselte Daten als Konfigurationen definiert sind [20]. Da der Benutzer in seiner App viele verschiedene Widgets zusammenstellt, kann ein Widget-Baum erstellt werden.

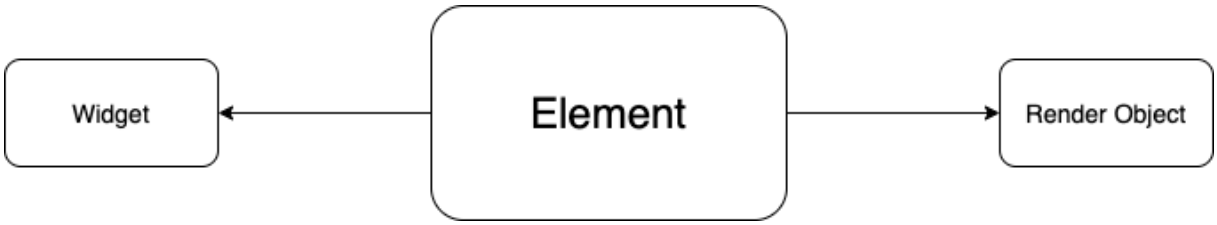

Abb. 2.2: Beziehung zwischen Widget, Element und RenderObject [21]

Der Unterschied zwischen einem Element und einem Widget besteht darin, dass das Widget als Konfiguration verwendet wird, um jedes Element zu erstellen. Das Element ist ein veränderbares Element, das für die Verwaltung des Lebenszyklus des Widgets verantwortlich ist. Mit veränderlich ist gemeint, dass die Eigenschaften des Elements geändert werden können. Elemente verwalten den Zustand und das Beziehungsmodell zwischen Widgets. Da ein Widgetbaum viele verschiedene Widgets enthält, kann auch ein visueller Baum von Elementen (Element-Baum) erstellt werden, der aus gerenderten Widgets besteht. Die Erstellung eines Elements und eines Widgets erfolgt zur gleichen Zeit. Während jedes Widget erstellt wird und der Widget-Baum vom Benutzer aufgebaut wird, baut das Flutter-Framework gleichzeitig den Element-Baum auf. Die Referenzen eines Widgets und eines RenderObjects werden in einem Element gespeichert, so dass verschiedene Widgets voneinander unterschieden werden können. Das Element ist auch für die Verknüpfung des Widgets mit dem entsprechenden Renderobjekt verantwortlich, da ein separater Render-Baum von einem Elementbaum verwaltet wird (siehe **Abb. 2.3**).

Die Definition des allgemeinen abstrakten Protokolls zur Erstellung der Widgets (*Painting*), das Compositing und das grundlegende Layout einer bestimmten Widget-Instanz werden mit RenderObject durchgeführt. Es spielt eine wichtige Rolle bei der Steuerung aller Logik, Größen, Layouts und anderer Konfigurationsparameter für das Malen des zugehörigen Widgets. Die Manipulation der verschiedenen Parameter und das eigentliche Rendering finden innerhalb des RenderObjects statt. Außerdem ist das RenderObject sehr schwer, da es Cache-Werte und Referenzen enthält, was seine Instanziierung sehr teuer macht. Die Widget-Schicht löst dieses Problem, indem sie eine reaktive API bereitstellt.

Der Prozess des Renderns von Widgets auf dem Bildschirm beginnt damit, wie die Widgets im Widget-Baum in Elemente umgewandelt werden und wie diese Elemente in RenderObjects umgewandelt werden. Jeder der drei verschiedenen Bäume (Widget-, Element- und RenderObject-Baum) steuert unterschiedliche Aspekte und Zustände des eigentlichen Renderings der Benutzeroberfläche:

- Die Konfiguration der Ansicht wird über die Widgetstruktur abgewickelt. Die Eigenschaften und Werte der einzelnen zugehörigen Widgets können über eine API zugewiesen werden.
- Der Lebenszyklus dessen, was auf dem Bildschirm zu sehen ist, und die Darstellung der App-Struktur werden vom Elementbaum verwaltet. Als veränderliches Element ist der Elementbaum für die Verwaltung der Aktualisierungen der Benutzeroberfläche verantwortlich.
- Das eigentliche Malen auf dem Bildschirm erfolgt mit dem RenderObjects-Baum, der auch als Schnittstelle zwischen High-Level-Code und der Low-Level-Bibliothek dart:ui dient [20].

Flutter Framework benutzt diese Bäume, um etwas auf dem Bildschirm zu zeichnen oder zu aktualisieren. Trotz der Unveränderlichkeit enthält das Widget den Zustand, der bestimmt, ob das Widget neu erstellt werden muss oder nicht. Bei jeder einzelnen Änderung muss der Widgetbaum aktualisiert werden. Der Rest des Prozesses, der in den beiden anderen Bäumen stattfindet, muss jedoch nicht aktualisiert werden. Als Beispiel präsentierte das Flutter-Team auf den *Google Developer Days 2019* in China eine einfache App, die einen Text enthält, um die einfache Vorgehensweise beim Rendern eines einfachen Widgets zu demonstrieren[4]. Das Beispiel ist unten zu sehen.

```
1 void main() {
2   runApp(RichText(
3       text: const TextSpan(text: 'Hello World!'),
4       textDirection: TextDirection.ltr));
5 }
```

Listing 2.3: Eine App, die ein einfaches Text-Widget enthält

Wenn die App vom Flutter-Framework gestartet wird, wird die Funktion runApp() implementiert. Als Eingabeparameter wird das Widget in einen Widget-Baum eingefügt (**Listing 2.3**). Anschließend wird ein Element, das als RenderObject-Element bezeichnet wird, durch das Widget erstellt (**Listing 2.5**). Anschließend wird das RenderObject durch das Element erstellt. Die erforderlichen Ansichtskonfigurationen werden dann vom Widget an RenderObject übergeben (**Listing 2.6**), in diesem Fall das RichText-Widget, das textbezogene Eigenschaften enthält. Die Übersicht über die drei Bäume wird in **Abb. 2.4** dargestellt.

[4] Flutter: How Flutter renders Widgets, 2019, [YouTube] https://www.youtube.com/watch?v=996ZgFRENMs&t=816s, 8:40–13:41.

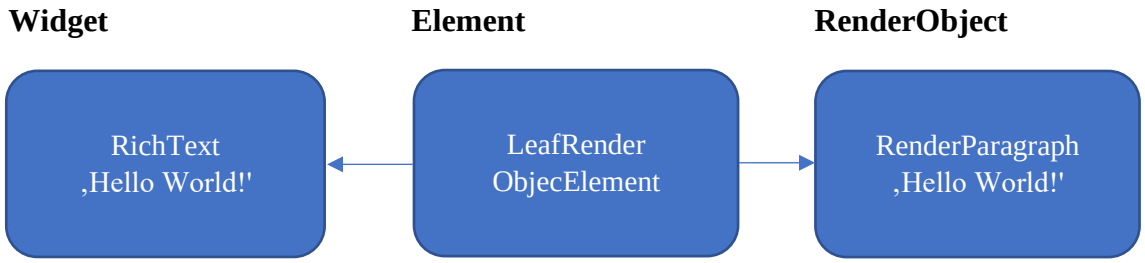

Abb. 2.3: Drei Bäume vor der Änderung

```
1 void runApp(Widget app) {
2   WidgetsFlutterBinding.ensureInitialized()
3     ..scheduleAttachRootWidget(app)
4     ..scheduleWarmUpFrame();
5 }
```

Listing 2.4: Funktion scheduleAttachRootWidget setzt das Widget auf die Wurzel des Baums

```
abstract class LeafRenderObjectWidget extends RenderObjectWidget {

  const LeafRenderObjectWidget({ Key? key }) : super(key: key);

  @override
  LeafRenderObjectElement createElement() => LeafRenderObjectElement(this);
}
```

Listing 2.5: RenderObject Element wird durch das Widget erstellt

```
  @override
  void mount(Element? parent, Object? newSlot) {
    super.mount(parent, newSlot);
    assert(() {
      _debugDoingBuild = true;
      return true;
    }());
    _renderObject = widget.createRenderObject(this);
  // ...
  }
```

Listing 2.6: Die Erstellung von RenderObject

Die Implementierung der drei Bäume ist sehr hilfreich, wenn einige der Aspekte des Widgets geändert werden müssen. Ein einfaches Beispiel: Der Wert des obigen Textwidgets soll in "Hallo Deutschland!" geändert werden. Wenn die Funktion runApp() erneut aufgerufen wird, wird das vorherige Textwidget zerstört und durch das neue Textwidget ersetzt. Die Erstellung eines neuen Elements und RenderObjects ist nicht mehr notwendig, da sie bereits vom

Framework erstellt wurden. Flutter ruft vom Textelement aus die Funktion canUpdate() auf (**Listing 2.7**), die den Typ des vorherigen Widgets mit dem neuen vergleicht. In diesem Fall haben beide Widgets denselben Typ, nämlich RichText.

```
static bool canUpdate(Widget oldWidget, Widget newWidget) {
    return oldWidget.runtimeType == newWidget.runtimeType
        && oldWidget.key == newWidget.key;
  }
```

Listing 2.7: canUpdate()-Funktion

Um die neue aktualisierte Ansichtskonfiguration zu rendern, wird die Funktion updateRenderObject() (**Listing 2.8**) von dem Element aufgerufen. Mit dieser Funktion wird das vorhandene RenderObject ausgewertet und sein alter Wert aktualisiert. Die Übersicht der drei Bäume nach der Aktualisierung wird in **Abb. 2.5** angezeigt.

```
@override
  void updateRenderObject(BuildContext context, RenderParagraph
renderObject) {
    assert(textDirection != null || debugCheckHasDirectionality(context));
    renderObject
      ..text = text
      ..textAlign = textAlign
      ..textDirection = textDirection ?? Directionality.of(context)
      ..softWrap = softWrap
      ..overflow = overflow
      ..textScaleFactor = textScaleFactor
      ..maxLines = maxLines
      ..strutStyle = strutStyle
      ..textWidthBasis = textWidthBasis
      ..textHeightBehavior = textHeightBehavior
      ..locale = locale ?? Localizations.maybeLocaleOf(context);
  }
```

Listing 2.8: updateRenderObject-Funktion

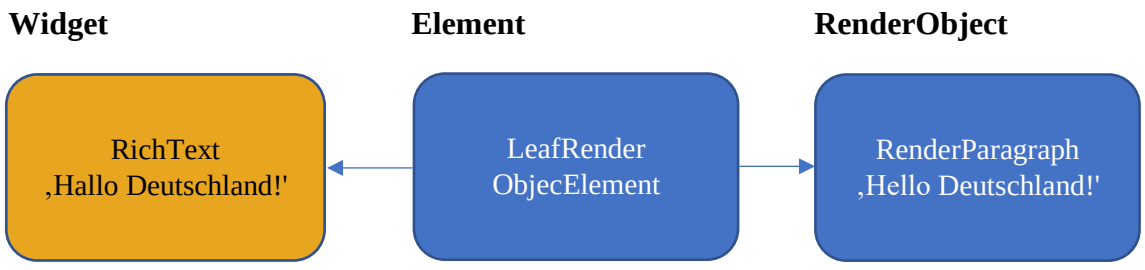

Abb. 2.4: Drei Bäume vor der Änderung

Die Verwendung dieser Bäume zeigt, dass das Framework intern intelligent sein kann, indem es teure RenderObjekte recycelt, während es mit der Zerstörung von preiswerten Widgets sorglos umgeht.

2.2.3. Stateless- und Stateful-Widget

Wenn Entwickler ihre Benutzeroberfläche entwerfen, gibt es zwei Haupttypen von Widgets, die meistens verwendet werden. Das statische Widget, das sich nur auf seine anfängliche Konfiguration stützt, wird als *Stateless*-Widget bezeichnet. Bei dieser Art von Widget muss kein veränderlicher Zustand, d. h. keine Eigenschaften oder Werte, die sich im Laufe der Zeit ändern, verfolgt werden. Die Daten des Widgets können nur geändert werden, wenn das Framework das Widget zerstört und neu erstellt [20], wie es in **Abb. 2.5** angezeigt wird.

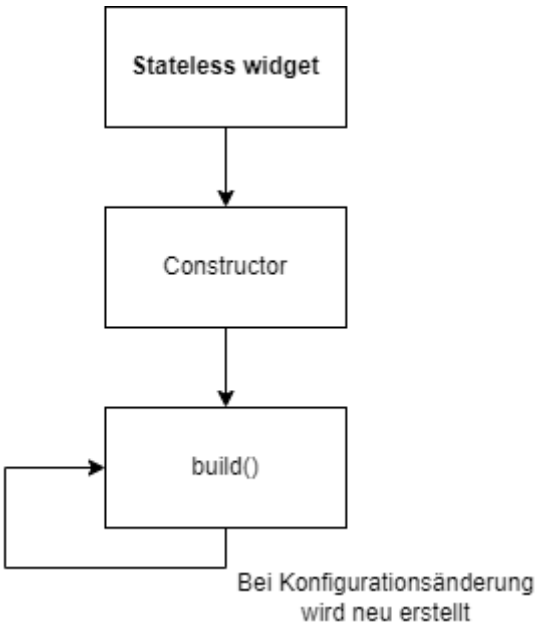

Abb. 2.5: Lebenszyklus eines Stateless-Widgets [20]

Wenn sich die Komponente der Benutzeroberfläche nur auf ihre eigene Ansichtskonfiguration verlässt, dann ist die Verwendung eines *Stateless*-Widgets sinnvoll. Ein Beispiel für ein *Stateless*-Widget ist ein einfaches Text-Widget und ein Icon-Widget. Der Text oder das Symbol enthält eine Beschreibung und ändert sich nicht.

```
class LoginBackground extends StatelessWidget {
  const LoginBackground({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Text('Wenn StatelessWidget verwendet wird...');
  }
}
```

Listing 2.9: Basisstruktur von *Stateless*-Widget

Die andere Art von Widget, auf die der Benutzer häufig trifft, ist das *Stateful*-Widget. Ein *Stateful*-Widget ist ein Widget, das seinen eigenen Zustand beibehalten kann. Das Widget selbst ist zwar unveränderlich bzw. *immutable*, kann sich aber dynamisch verändern. Um den veränderlichen Zustand zu erreichen, umfasst ein *Stateful*-Widget zwei Klassen: die StatefulWidget-Klasse und die State-Klasse. Das Flutter-Framework erzwingt die Trennung zwischen UI und *State Management* (**Abb. 2.7**). Mit dieser Art des Softwaredesigns kann die Performance verbessert werden, weil das Neuberechnen der Zustandsdaten und das Neuzeichnen der Konfiguration Zeit in Anspruch nehmen. Durch die Trennung ermöglicht das Framework dem Prozessor, jede dieser Aufgaben unabhängig und effizient zu verwalten [22].

Genau wie das *Stateless*-Widget kann der Benutzer das *Stateful*-Widget neu erstellen, nachdem seine Ansichtskonfigurationen geändert wurden. Die wartbaren und veränderbaren Daten werden jedoch innerhalb der State-Klasse gespeichert. Betrachtet man die Basisstruktur des *Stateful*-Widgets in **Listing 2.10**, implementiert jedes *Stateful*-Widget die Funktion createState(), die ein State-Objekt zurückgibt. Der Zustand kann innerhalb des zugehörigen State-Objekts persistiert werden, wenn das Framework das Widget neu aufbaut. Um mitzuteilen, ob das State-Objekt geändert oder erneuert wurde, wird die Funktion setState() implementiert. Anschließend wird das Widget von der State-Klasse mit der build()-Funktion neu aufgebaut oder neu gezeichnet und die aktualisierten Werte können angezeigt werden. Es ist erwähnenswert, dass beim Entfernen des Stateful-Widgets aus dem Widget-Baum und dem erneuten Einfügen in den Baum zu einem späteren Zeitpunkt ein neues State-Objekt erstellt wird. Die Änderung, Anpassung und Manipulation der Daten kann in verschiedenen Phasen des Lebenszyklus des Widgets erreicht werden, indem verschiedene Teile von StatefulWidget überschrieben werden. Einige der wichtigsten Überschreibungen von Stateful-Widget sind initState(), dispose(), didUpdateWidget() und setState().

```dart
class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key}) : super(key: key);

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {

  @override
  Widget build(BuildContext context) {
  // ...
  }
}
```

Listing 2.10: Basisstruktur von *Stateful*-Widget

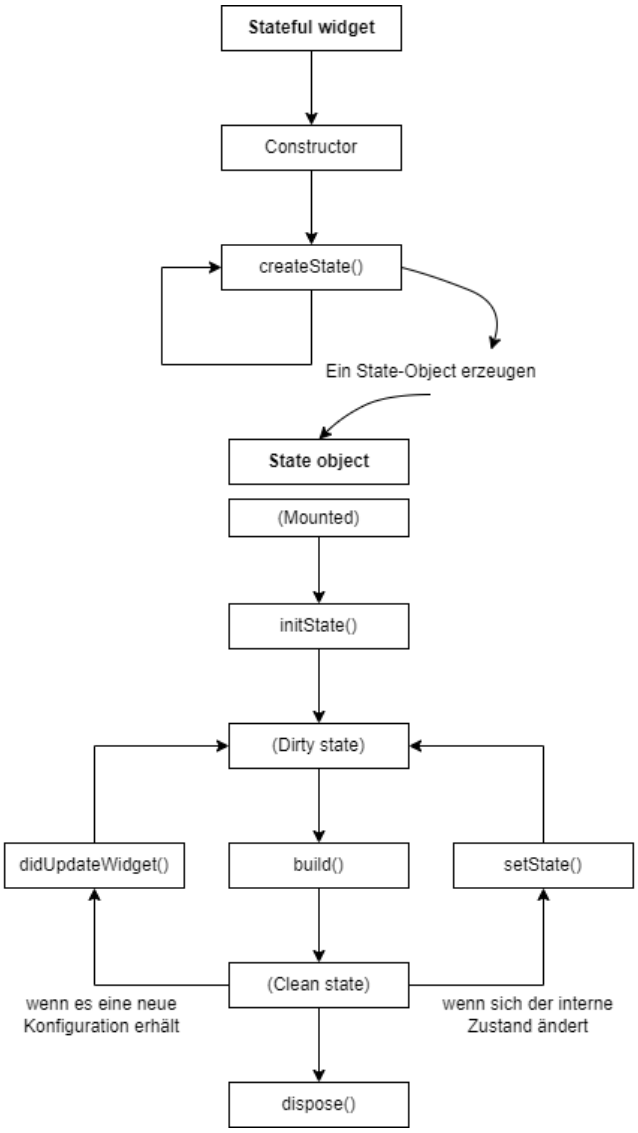

Abb. 2.6: Lebenszyklus eines Stateful-Widgets [20]

2.3. PHP-Framework Laravel

2.3.1. Überblick

Wie bereits in Kapitel eins erläutert, wird für den Backend-Service der App eine relationale Datenbank verwendet, um die Erstellung komplexer Datenbankstrukturen für die zukünftige Entwicklung der App zu erleichtern. Die Lösung für diese Anforderung ist die Implementierung von Laravel Framework.

Laravel ist ein kostenloses Open-Source-Webframework, das 2011 von Taylor Otwell entwickelt wurde, um eine Alternative zum CodeIgniter-Framework zu bieten. CodeIgniter ist ebenfalls eine Open-Source-Software für die Erstellung dynamischer Websites. Laut Otwell fehlten CodeIgniter aufgrund der langsamen Weiterentwicklung moderner Technologien und Patterns wesentliche Funktionen, die von den Nutzern zu dieser Zeit häufig verwendet werden, wie z. B. Benutzerauthentifizierung und -autorisierung [23]. Das inspirierte Otwell dazu, sein eigenes Framework mit vielen neuen eingebauten Funktionen zu entwickeln. Laravel basiert auf Symfony, einem anderen Open-Source-Framework für Webanwendungen. Die meisten Bibliotheken von Drittanbietern stammen aus Symfony-Komponenten, so dass 30 % des Laravel-Codes aus Symfony besteht [24]. Um die erforderlichen Abhängigkeiten oder Bibliotheken zu verwalten, verwendet Laravel den Composer.

Laravel ist in PHP entwickelt und ermöglicht es dem Entwickler, Anwendungen nach dem MVC-Muster zu erstellen. Otwell selbst hat jedoch erklärt, dass Laravel nicht unbedingt ein MVC-Framework ist, denn MVC selbst ist ein einschränkender Begriff in der Art und Weise, wie er auf die Webentwicklung angewandt wird[5]. Für die Zwecke dieser Arbeit wird die MVC-Architektur nicht besprochen, weil die App nur einen Webservice benötigt, der nicht dem MVC-Architekturdesign folgt.

Es ist allerdings wichtig, einige Begriffe zu verstehen, die bei der Einrichtung des Backend-Servers häufig verwendet werden, z. B. Modell und Controller. Das Modell ist die Darstellung der Ressourcen. Mit anderen Worten, es ist für die Verwaltung der dynamischen Daten und aller domänenlogischen Prozesse zuständig, z.B. für das Abrufen, Validieren, Speichern,

[5] https://mobile.twitter.com/taylorotwell/status/634394209479319552

Aktualisieren und Löschen der Daten [25]. Der Controller verarbeitet die Eingaben des Benutzers und die HTTP-Anfragen [23]. Der Prozess der Definition der Anfragelogik wird mit Hilfe des Controllers organisiert. Normalerweise wird eine einzige Klasse verwendet, um die relevanten Funktionen zur Bearbeitung von Anfragen zu kombinieren [26]. Als einfaches Beispiel enthält die integrierte Basisklasse UserController die Logik zur Steuerung aller eingehenden Anfragen, die mit einem Benutzer verbunden sind, wie beispielsweise das Erstellen, Aktualisieren, Anzeigen und Löschen von Benutzern.

2.3.2. Eloquent ORM

Um die miteinander verbundenen Modelle zu definieren und auf die Datenbank zuzugreifen, verwendet Laravel den Eloquent *Object Relational Mapper* (Eloquent ORM). Ein ORM ist eine Software, die als Abstraktionsschicht über den Datenbanktreiber fungiert. In dem Datenbanktreiber werden die Daten einer Anwendung gespeichert. Das ORM kann die Steuerung von Datenbankeinträgen erleichtern, indem es Daten als Objekte ausdrückt [27]. Als Modellschicht in der Anwendung ist Eloquent für die Kommunikation mit Datenbanktabellen verantwortlich, wobei eine objektorientierte Methode zum Einfügen, Löschen und Aktualisieren von Datenbankeinträgen sowie eine schlanke Schnittstelle zur Ausführung von SQL-Abfragen implementiert wird. Es handelt sich um ein ActiveRecord ORM. Es bedeutet, dass eine erstellte Eloquent-Modellklasse in Laravel nicht nur die Funktion unterstützt, bei der der Benutzer mit der Tabelle als Ganzes kommunizieren kann (z.B. User::all() holt alle Benutzer), sondern auch die Darstellung einer einzelnen Tabellenzeile ermöglicht (z.B. $john = new User) [23]. Eloquent ORM kann mit einer Vielzahl von Datenbanken verwendet werden, z.B. MySQL, SQLite, SQL Server und PostgreSQL.

Eloquent ORM verfügt über eine Funktionalität, mit der sich Beziehungen zwischen Modellinstanzen einfach herstellen lassen. Sie wird als *Model Instance Relations* bezeichnet. Es gibt viele verschiedene Arten von Beziehungen, die von Eloquent unterstützt werden: *one-to-one, one-to-many, many-to-many, has-many-through*, polymorphe Beziehungen und *many-to-many* polymorphe Beziehungen [28]. In dieser Arbeit werden jedoch nur drei wesentliche Arten von Beziehungen zwischen Modellen erörtert: *one-to-one, one-to-many und many-to-many* Beziehungen.

Ein praktisches Beispiel für eine einfache *one-to-one*-Beziehung ist, wenn ein Benutzer nur eine einzige Telefonnummer hat. Anders ausgedrückt: Ein Benutzermodell ist mit einem einzigen Telefonmodell verknüpft. Die Art und Weise, wie die Beziehung definiert ist, ist in **Listing 2.11** zu sehen. Der Name der verknüpften Modellklasse, in diesem Fall der Phone-Klasse, wird als Eingabeparameter der Funktion hasOne() angegeben. Die Phone-Klasse enthält nicht nur relevante Informationen, sondern auch einen Fremdschlüssel bzw. *foreign key* des Benutzers. Um anschließend vom Benutzermodell aus auf die Daten des Telefonmodells zuzugreifen, wird die Funktion belongsTo() implementiert (siehe **Listing 2.12**). Da die Funktion user() in der umgekehrten Funktion implementiert ist, versucht Eloquent, nach einem User-Modell mit einer ID zu suchen, die mit der user_id-Spalte des Phone-Modells übereinstimmt.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone associated with the user.
     */
    public function phone()
    {
        return $this->hasOne(Phone::class);
    }
}
```

Listing 2.11: eine Eins-zu-Eins-Beziehung für das Benutzermodell definieren [28]

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

Listing 2.12: die Umkehrung der Beziehung [28]

One-to-many-Beziehung wird implementiert, wenn eine einzelne Model-Klasse als übergeordnete Klasse fungiert und mehr als ein untergeordnetes Model hat. Das gängige Beispiel ist, dass in einem Nachrichtensystem eine unendliche Anzahl von Kategorien berichtet werden kann. Die Beziehung kann durch die Implementierung der Funktion hasMany() in der übergeordneten Klasse hergestellt werden, die in diesem Fall die Klasse News ist (siehe **Listing 2.13**). Die Sammlung der Kategorien kann dann abgerufen werden, sobald die Beziehungsmethode definiert wurde. Damit eine Kategorie auf ihre übergeordneten Nachrichten zugreifen kann, wird eine umgekehrte Methode von hasMany() namens belongsTo() in der untergeordneten Modellklasse implementiert, in diesem Fall die Category-Klasse (siehe **Listing 2.14**). Anschließend kann dann auf die übergeordneten Nachrichten der Kategorie zugegriffen werden.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class News extends Model
{
    /**
     * Get the categories for the news.
     */
    public function categories()
    {
        return $this->hasMany(Category::class);
    }
}
```

Listing 2.13: Festlegen einer *one-to-many*-Beziehung im Nachrichtenmodell

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    /**
     * Get the news that owns the category.
     */
    public function news()
    {
        return $this->belongsTo(News::class);
    }
}
```

Listing 2.14: belongsTo()-Funktion als umgekehrte Methode zur hasMany()-Funktion

Wie der Name schon sagt, ist eine *many-to-many*-Beziehung komplexer als die anderen oben genannten Beziehungen. Es bedeutet, dass viele Modelle mehrere Modelle haben können. Für diese Beziehung ist eine Zwischentabelle erforderlich, die als Verbindungstabelle oder in Laravel als Pivot-Tabelle bezeichnet wird [23]. Sie ist erforderlich, um Modelle zu unterstützen, die mehreren Modellen zugeordnet sind.

Ein einfaches Beispiel, das die Implementierung dieser Beziehung veranschaulicht, ist ein Berechtigungssystem, in dem einem Benutzer eine Teilmenge von Berechtigungen zugewiesen werden kann. Die verschiedenen Berechtigungen können auf der Grundlage der Rolle, die dem Benutzer zugewiesen ist, zugeordnet werden. Das bedeutet, dass ein Benutzer mehrere Rollen haben kann und dass die Rollen auch vielen Benutzern zugewiesen werden können. Wenn die Spalte user_id einfach in die roles-Tabelle eingefügt wird, ergibt sich ein weiteres Problem. Denn das bedeutet, dass eine Rolle nur einem einzigen Benutzer zugewiesen werden kann. Um die Zuweisung von Rollen an mehrere Benutzer zu erleichtern, wird daher die Pivot-Tabelle role_user erstellt. Da die Tabellen user und role miteinander verknüpft sind, benötigt die Pivot-Tabelle zwei Spalten: user_id und role_id.

Zur Anwendung dieser Beziehung wird die Funktion belongsToMany() im Benutzermodell definiert (siehe **Listing 2.16**). Anschließend kann der Benutzer auf seine Rollen zugreifen. Die *many-to-many*-Beziehung verwendet ebenfalls die Funktion belongsToMany() als inverse Methode (siehe **Listing 2.17**).

```
users
    id - integer
    name - string

roles
    id - integer
    name - string

role_user
    user_id - integer
    role_id - integer
```

Listing 2.15: Struktur der Beziehungstabelle [28]

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    public function roles()
    {
        return $this->belongsToMany(Role::class);
    }
}
```

Listing 2.16: Initialisierung einer *many-to-many*-Beziehung [28]

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    public function users()
    {
        return $this->belongsToMany(User::class);
    }
}
```

Listing 2.17: inverse Methode der *many-to-many*-Beziehung [28]

3. Entwicklung der Systeme

Vor der Erläuterung des Entwurfs der mobilen App werden der Kontext und der Umfang des Projekts festgelegt und ausgearbeitet. Danach werden die funktionalen und nicht-funktionalen Anforderungen erörtert. Anschließend wird der Konzeptionsabschnitt, der die Mockups der Anwendung enthält, besprochen. Abschließend kann der Entwicklungsprozess des Systems abgebildet werden.

3.1. Auslegung des Systems

3.1.1. Überblick

Das Ziel dieser Arbeit ist es, eine mobile Anwendung mit Flutter SDK zu entwickeln, die Studienbewerbern in Indonesien bei der Vorbereitung ihres Studiums in Deutschland hilft. Die App soll als Plattform dienen, auf der alle getrennten Informationen zur Studienvorbereitung in Deutschland aus dem Internet übersichtlich zusammengefasst werden können, um so die Verwirrung und den Zeitaufwand der Studienbewerber bei der Recherche zu verringern.

Die App soll auf Android und in Zukunft auch auf iOS verfügbar sein. Daher wird das Flutter SDK als Softwarelösung verwendet, das bereits in Kapitel 2.2 besprochen wurde, so dass nur eine einzige Codebasis geschrieben werden muss, um die Anwendung für beide Betriebssysteme effektiv zu entwickeln. Darüber hinaus können die Entwickler auch Zeit sparen, um die iOS-Version der App in Zukunft zu entwerfen. Für diese Arbeit wird derzeit nur die Android-Version der App entwickelt.

Um die grundlegenden Anforderungen und die wichtigen Funktionalitäten sowie deren Verbindung zueinander klarer zu definieren, wird ein Anwendungsfalldiagramm erstellt (siehe **Abb. 3.1**). Darin gibt es zwei Akteure, den Nutzer und den Kundenservice, die jeweils unterschiedliche Interaktionen mit dem System haben. Die detaillierten Prozess- und Informationsflüsse der App werden mit Hilfe eines BPMN-Diagramms genauer beschrieben, das in Unterkapitel 3.2 behandelt wird.

Außer der Möglichkeit, detaillierte Informationen zu den Studienvorbereitungen abzurufen, sollten die Studienbewerber auch die Möglichkeit haben, sich an den erfahrenen Kundendienst

zu wenden, wenn sie Hilfe benötigen. Daher wird der Live-Chat-Service mit dem Kommunicate Plugin angeboten, um diese Anforderung zu erfüllen. Außerdem sollten die Studienbewerber in der Lage sein, ihren Vorbereitungsfortschritt anhand einer Reihe von Checklisten zu verfolgen, die aus jedem Vorbereitungsschritt und dem dazugehörigen Kontrollkästchen bestehen. Um dieses System aufzubauen, wird ein Backend-Service umgesetzt. In dieser Arbeit wird das Web-Framework Laravel verwendet, das bereits in Unterkapitel 2.3 beschrieben wurde. Mit den in Laravel integrierten Authentifizierungs- und Autorisierungsfunktionen können die Studienbewerber ihren Vorbereitungsfortschritt speichern und verfolgen.

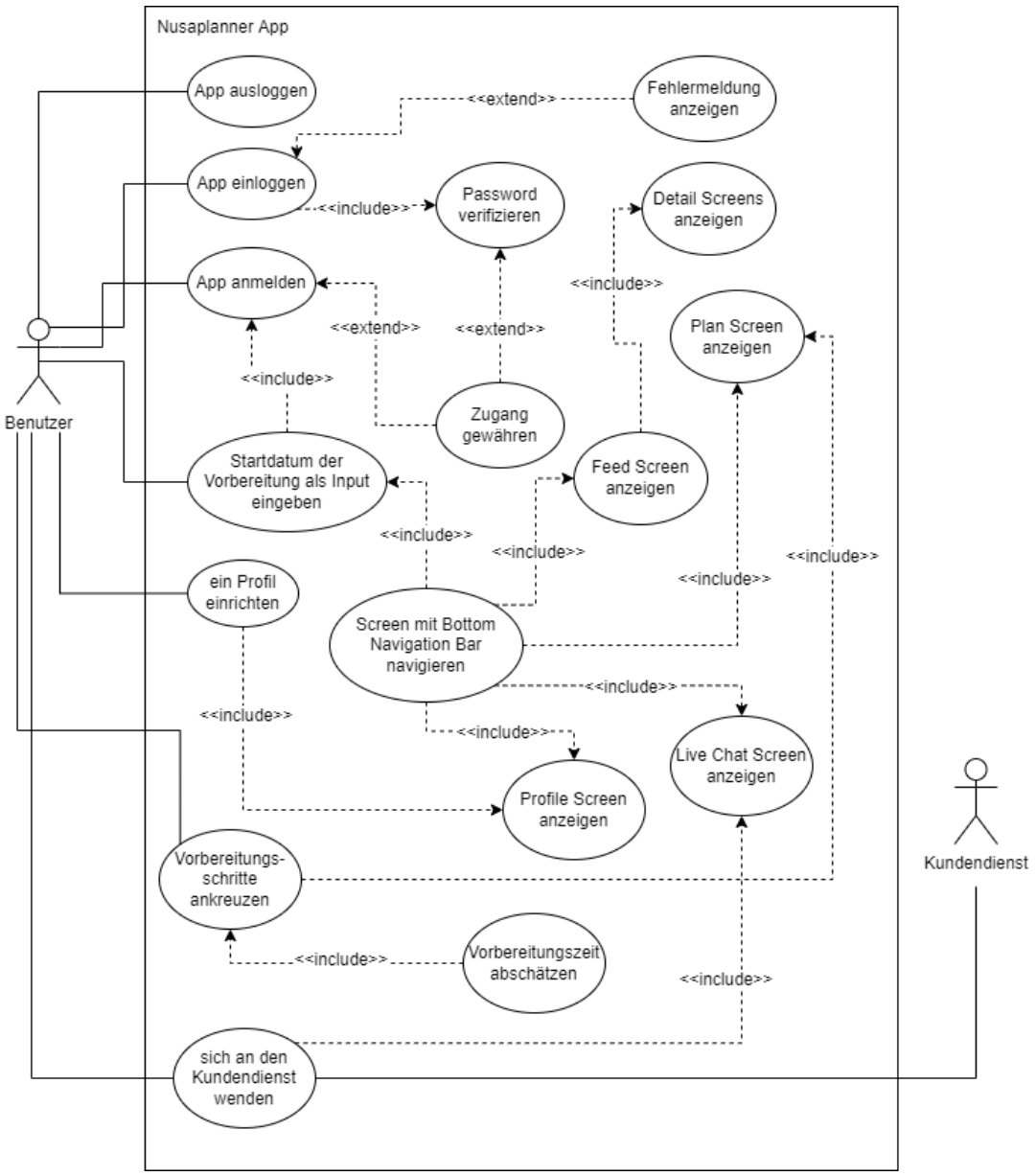

Abb. 3.1: Use-Case Diagramm von Nusaplanner App

Bei der Entwicklung dieser App muss sichergestellt werden, dass jedes Softwaresystem oder -element seine Funktion korrekt ausführt. Daher ist die Verwendung der Systemarchitektur, die aus verschiedenen Softwareelementen und deren Beziehung zueinander besteht, in einer sehr frühen Phase des Entwicklungsprozesses wichtig. Die allgemeine Architektur einer mobilen App besteht aus drei verschiedenen Schichten (siehe **Abb. 3.2**).

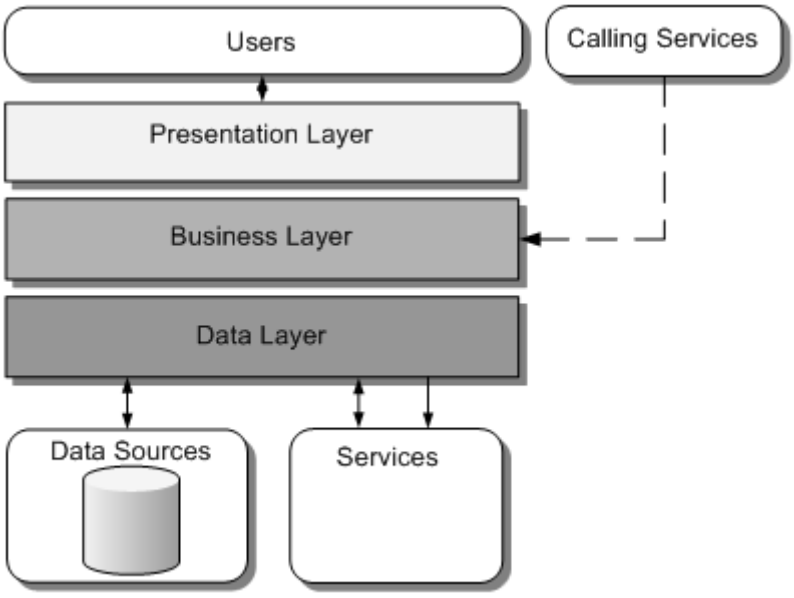

Abb. 3.2: Allgemeines Design der mobilen App-Architektur [29]

Die erste Schicht, die aus UI-Elementen besteht, heißt *Presentation Layer*. Die Business-Komponenten und die Logik der Anwendung werden in der Business-Schicht definiert. Die Logik oder der Workflow für den Zugriff auf den Datenspeicher (z.B. Lesen/Einfügen in die Datenbanktabelle), die durch die Logikkomponenten für den Datenzugriff dargestellt werden, und die Tools zum Ändern und Transformieren von Daten sowie zur Handhabung der Datenbankverbindung befinden sich in der Datenschicht.

Die Beziehung zwischen den Softwareelementen dieser App, wie z.B. der Benutzeroberfläche und dem Backend-Service, wird in einem Systemarchitekturdiagramm visualisiert (siehe **Abb. 3.3**).

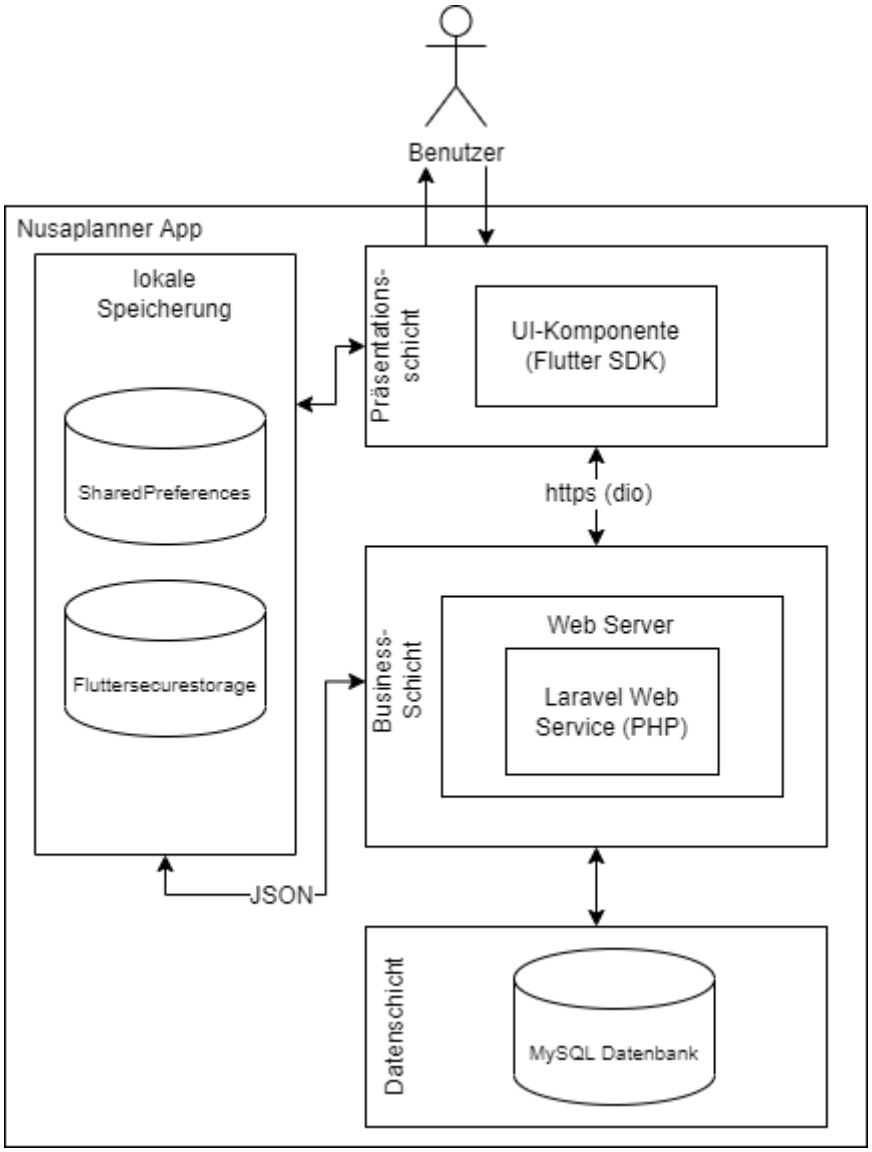

Abb. 3.3: Systemarchitektur Diagramm

Wie im obigen Diagramm dargestellt, besteht die App aus drei wichtigen Teilen. In der App findet die gesamte Interaktion des Benutzers mit der App, z.B. die Eingabe des Startdatums für die Studienvorbereitung und die Änderung der Checkliste der Vorbereitungsschritte durch Ankreuzen des Kästchens, in den UI-Komponenten der App statt. Die persönlichen Daten des Benutzers, das Startdatum und die Werte der Checkliste werden im Business-Schicht mit Laravel verarbeitet und schließlich werden alle Daten in der MySQL-Datenbank gespeichert. Es ist auch möglich, die Benutzerdaten wie beispielsweise Zugriffstoken lokal zu speichern, indem man das Fluttersecurestrorage-Plugin verwendet. Kleine Mengen von Daten wie z.B. das Start- und Enddatum der Studienvorbereitung kann man ebenfalls lokal mit dem SharedPreference-Plugin aufbewahren.

3.1.2. Anforderungsanalyse

In diesem Abschnitt werden die Anforderungen und ihre Priorität in 2 Teile unterteilt, nämlich in funktionale und nicht-funktionale Anforderungen. Zur besseren Übersichtlichkeit sind die Anforderungen unten in Form von Tabellen aufgeführt.

Funktionale Anforderung

Nr.	Anforderung	Priorität
F1.1	Der Benutzer kann ein Konto mit seiner eigenen E-Mail erstellen	Hoch
F1.2	Der Benutzer kann sich mit seinem eigenen Konto bei der App anmelden	Hoch
F1.3	Wenn der Benutzer sein Passwort vergessen hat, kann er ein neues Passwort erstellen.	Niedrig
F2.1	Die App sollte über Startbildschirme (*Splash Screens*) verfügen, die Informationen zur Verwendung der App enthalten	Hoch
F2.2	Einer der Startbildschirme sollte eine Funktion enthalten, bei der der Benutzer das Datum des Beginns seiner Studienvorbereitung eingeben kann, damit die Schätzung seiner Abreise gespeichert und angezeigt werden kann.	Hoch
F3	Die App besteht aus 4 Hauptseiten: *Feed*-Seite (Dashboard), *Plan*-Seite, *Inbox*-Seite und *Profile*-Seite	Mittel
F4.1	Die App verfügt über eine untere Navigationsleiste, mit der die Benutzer zwischen den Ansichten der obersten Ebene in der App navigieren können.	Hoch
F4.2	Oben auf der *Feed*-Seite wird ein Karussell angezeigt, das verschiedene Fotos oder Miniaturansichten von verschiedenen Nachrichten oder Artikeln zur Studienvorbereitung enthält.	Niedrig
F4.3	Auf der *Feed*-Seite sollte der Benutzer in der Lage sein, die Details der einzelnen Vorbereitungen in der allgemeinen Übersicht zu sehen.	Hoch
F4.4	Jedes Vorbereitungsdetail enthält mehrere Seiten und kann aus Dokumentdateien in Form von PDF und externen Links bestehen	Mittel
F4.5	Der Benutzer sollte in der Lage sein, eine bestimmte PDF-Datei mit dem PDF-Viewer in der App zu öffnen	Hoch
F4.6	Der Benutzer kann den externen Link in der App öffnen	Hoch

F4.7	Auf der *Feed*-Seite können eine Reihe von Tools verwendet werden, um die Studienvorbereitung zu erleichtern	Hoch
F4.8	Auf der *Feed*-Seite kann das Startdatum geändert werden, wodurch sich auch die Schätzung der Abreise ändert.	Hoch
F4.9	Auf der *Feed*-Seite kann der Live-Chat-Service aufgerufen werden	Mittel
F5.1	Auf der *Plan*-Seite kann der Benutzer die geschätzte Abfahrtszeit sehen	Hoch
F5.2	Auf der *Plan*-Seite kann der Benutzer die Checklisten auf der Liste der Vorbereitung aktualisieren	Hoch
F6	Der Live-Chat-Service kann über die *Inbox*-Seite aufgerufen werden	Hoch
F7	*Push-Notification* sollte implementiert werden	Niedrig
F8	Der Benutzer kann sich bei der App abmelden	Hoch
F9	Der Benutzer kann eingeloggt bleiben, wenn die App geschlossen wird	Hoch
F10	Die Werte der Checkliste (TRUE oder FALSE) können lokal gespeichert werden. Wenn die App geschlossen wird, bleiben die Daten erhalten	Hoch

Tabelle 3.1: Funktionale Anforderungen

Nicht-Funktionale Anforderung

Nr.	Anforderung	Priorität
NF 1	Die Informationen in jedem Detail der Vorbereitungsschritte sollten klar und lesbar sein	Hoch
NF2	Das App-Design sollte eine hohe Benutzerfreundlichkeit (*high usability*) aufweisen	Mittel
NF3	Die App sollte die aktuellen Informationen zur Studienvorbereitung enthalten	Hoch
NF4	Der Benutzer sollte den Fortschritt seiner Vorbereitung verfolgen können	Hoch
NF5	Die App soll eine Plattform sein, um die Verwirrung bezüglich der Studienvorbereitung in Deutschland zu lösen	Hoch
NF6	Die App soll auf gängigen Android-Geräten laufen.	Hoch

Tabelle 3.2: Nicht-Funktionale Anforderung

3.2. Entwurf

Das Design einer App ist einer der wichtigsten Schritte bei der Entwicklung einer Anwendung. Daher widmet sich dieses Kapitel der schrittweisen Umsetzung der Idee, angefangen von der Beschreibung des Projektablaufs, der in **Unterabschnitt 3.2.1** (Prozessüberblick) kurz erläutert wird, bis hin zur Implementierung der gesamten Projektidee.

3.2.1. Prozessüberblick

Vor der Realisierung dieser App müssen zunächst einige visuelle Diagramme erstellt werden, um eine detaillierte Abfolge der Informationsflüsse zu beschreiben, die innerhalb der App ablaufen, so dass der Prozess durch eine leicht verständliche visuelle Darstellung besser verstanden werden kann. BPMN ist zum Beispiel eine Flussdiagramm-Methode, die die Schritte eines geplanten Geschäftsprozesses von Anfang bis Ende modelliert [30]. Die BPMN-Diagramme helfen dabei, den komplizierten App-Prozess zu erklären, der zum Beispiel im **Unterabschnitt 3.2.2** (Konzeptionierung der App) erläutert wird. Danach können die Mockups und das Design der App konzeptioniert werden. Das Mockup der App wird mit Figma entworfen und ist in der **Abb. 3.4** zu sehen.

Die Einrichtung des Backend-Dienstes unter Verwendung des Laravel-Frameworks erfolgte gleichzeitig mit der Erstellung des Mockups. Der Einrichtungsprozess des Dienstes begann mit der Erstellung des neuen Laravel-Projekts, das bereits Codes für die Benutzerauthentifizierung und -autorisierung enthält. Die Codes müssen jedoch noch geändert werden, um die zuvor genannten Anforderungen zu erfüllen. Nach der Einrichtung des Backend-Dienstes wird die Benutzeroberfläche des Dienstes mit dem Flutter SDK entworfen. Neben der Gestaltung der Benutzeroberfläche selbst wird in diesem Prozess auch die Navigation der App definiert. Am Ende werden das Design der App und die Navigationsstruktur evaluiert. Die App ist jedoch noch nicht mit den externen Diensten, wie dem Backend-Dienst und dem Live-Chat-Dienst, verbunden. Der detaillierte Ablauf dieses Schrittes wird im **Unterabschnitt 3.2.2**. besprochen.

Nachdem die Benutzeroberfläche der App und ihr Navigationsfluss erfolgreich entworfen und konfiguriert wurden, besteht der nächste komplizierte Prozess darin, die App mit dem externen Dienst zu verbinden. Dieser Prozess wird im **Unterabschnitt 3.2.3** besprochen. Wenn die App

erfolgreich mit den externen Diensten verbunden ist und der Backend-Dienst während des Verbindungsaufbaus ordnungsgemäß läuft, kann die App anschließend getestet werden. Erst nach dem erfolgreichen Test der App kann der Backend-Dienst dann auf einem Live-Webhost bereitgestellt werden. Um sicherzustellen, dass die Verbindung zwischen der App und dem Backend-Dienst weiterhin problemlos funktioniert, muss die App auch nach der Bereitstellung des Backend-Dienstes noch getestet werden. Schließlich kann die App als .APK-Daten exportiert werden, womit der gesamte Entwicklungsprozess abgeschlossen ist.

Um die vereinfachte Visualisierung des Entwicklungsprozesses zu sehen, siehe **Abb. 3.4** unten.

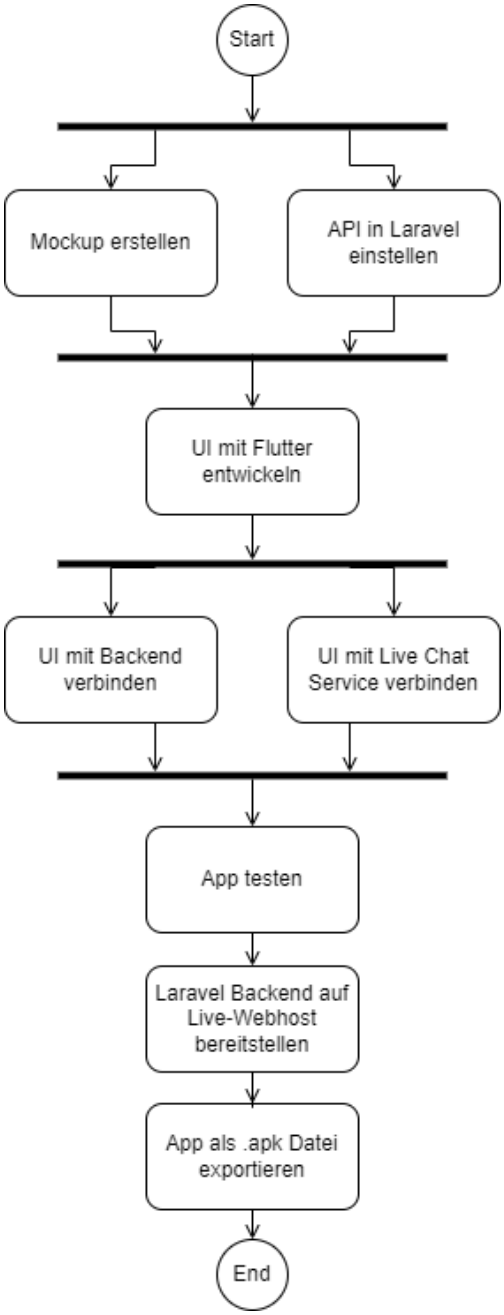

Abb. 3.4: Diagramm des Realisierungsprozesses

3.2.2. Konzeptionierung der App

Das Konzeptionsdesign ist ein wichtiger Schritt vor der Implementierung des gesamten Systems. Es ist ein sich wiederholender kreativer Prozess, um die Struktur der App zu gestalten, die die Designs und die Funktionalitäten der App umfasst. Das Ergebnis des Konzeptionsdesigns kann daher später dazu verwendet werden, den Entwickler bei der Ausführung der ersten Phase der App-Entwicklung zu unterstützen. Diese Phase umfasst die Erstellung verschiedener UI-Seiten und der grundlegenden Navigation. Während des Konzeptionsdesigns wird Figma als Werkzeug zur Visualisierung der Benutzeroberfläche und ihrer Navigation verwendet (siehe **Abb. 3.5**). Dank der leicht verständlichen Anwendung kann der Designprozess effizient erledigt werden.

Die wichtigsten Seiten der App sind unten aufgeführt:

- Seite für die Authentifizierung und Autorisierung sowie Registrierung von Benutzern: *SignIn*- und *SignUp*-Seite,
- Seite für die Informationen zur Verwendung der App: *Splash*-/*Onboarding*-Seite,
- Seite für die Eingabe des Startdatums der Studienvorbereitung: *AddDate*-Seite,
- Seite für die Anzeige des Enddatums der Studienvorbereitung und der Checklisten: *Plan*-Seite,
- Seite, auf der ausführliche Informationen zur Studienvorbereitung angezeigt werden: *Feed*-Seite,
- Seite für den Live-Chat-Service: *Inbox*-Seite,
- Seite, auf der sich der Benutzer von der App abmelden kann: *Profile*-Seite.

Die App sollte außerdem intuitiv sein. Das bedeutet, dass die App eine benutzerfreundliche Oberfläche haben sollte, so dass der Lernaufwand für die Nutzung der App gering ist. Ursprünglich wird der Inhalt der App auf Indonesisch verfasst sein. Für die Zwecke dieser Arbeit wird er jedoch auf Englisch verfasst. Die Übersicht über die Gestaltung der Benutzeroberfläche werden in **Abb. 3.7** bis **Abb. 3.10** dargestellt. Der kurze Überblick über die Funktionalitäten der App wird visuell in einem BPMN-Modell dargestellt (siehe **Abb. 3.6**).

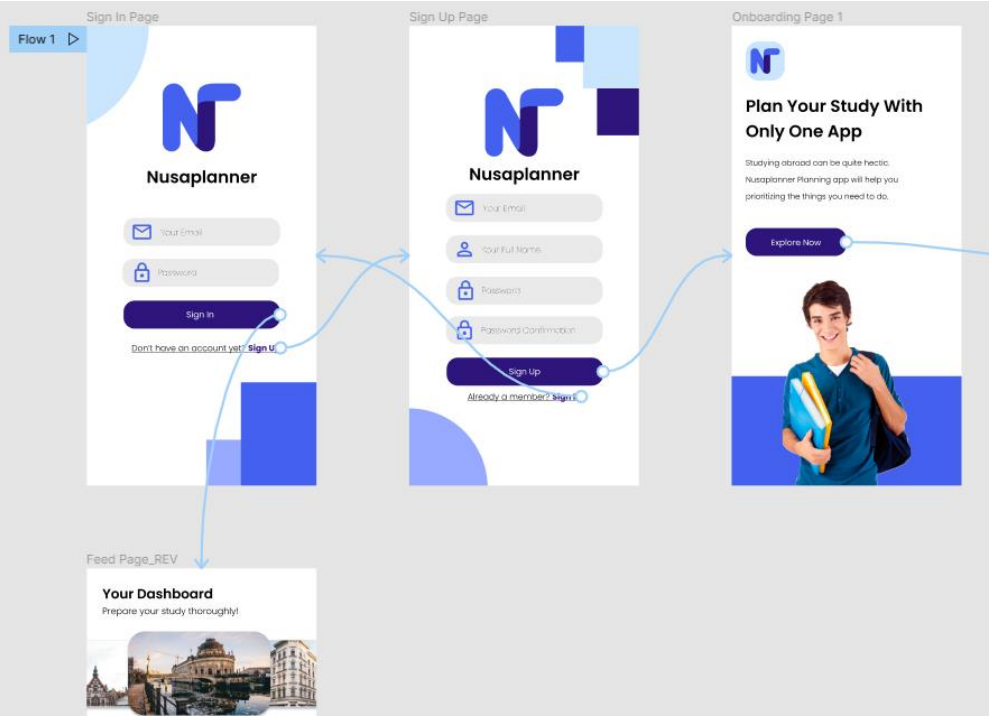

Abb. 3.5: Überblick über die in Figma implementierte grundlegende Navigation der App

Der Navigationsfluss der App beginnt, wenn der Benutzer die Anwendung öffnet. Als erstes wird die *SignIn*-Seite angezeigt. Auf dieser Seite hat der Benutzer die Möglichkeit, sich bei seinem Konto anzumelden, wenn er bereits ein Konto besitzt. Die erforderlichen Informationen sind die E-Mail-Adresse und das Passwort des Benutzers. Falls nicht, kann der Benutzer auf die *SignUp*-Seite gehen und ein neues Konto erstellen. Der Fehlerdialog kann angezeigt werden, wenn der Benutzer die erforderlichen Textfelder falsch ausfüllt. Dies gilt auch, wenn der Benutzer während des Anmeldevorgangs die falschen Anmeldedaten eingibt.

Nach der Anmeldung wird der Benutzer zur *Plan*-Seite weitergeleitet. Wenn der Benutzer das Konto jedoch gerade erst erstellt hat, muss er mehrere Einführungsbildschirme bzw. *Onboarding*-Seiten durchlaufen (siehe **Abb. 3.8**), die die Hinweise zur Verwendung der App enthalten. Diese Bildschirme werden nur einmal angezeigt (nach der Anmeldung des Benutzers). Anschließend wird der Benutzer aufgefordert, das Startdatum der Studienvorbereitung einzugeben. Ziel dieser Funktion ist es, dem Benutzer die voraussichtliche Abfahrtszeit (Enddatum der Studienvorbereitung) zu berechnen, die auf der *Plan*-Seite angezeigt wird.

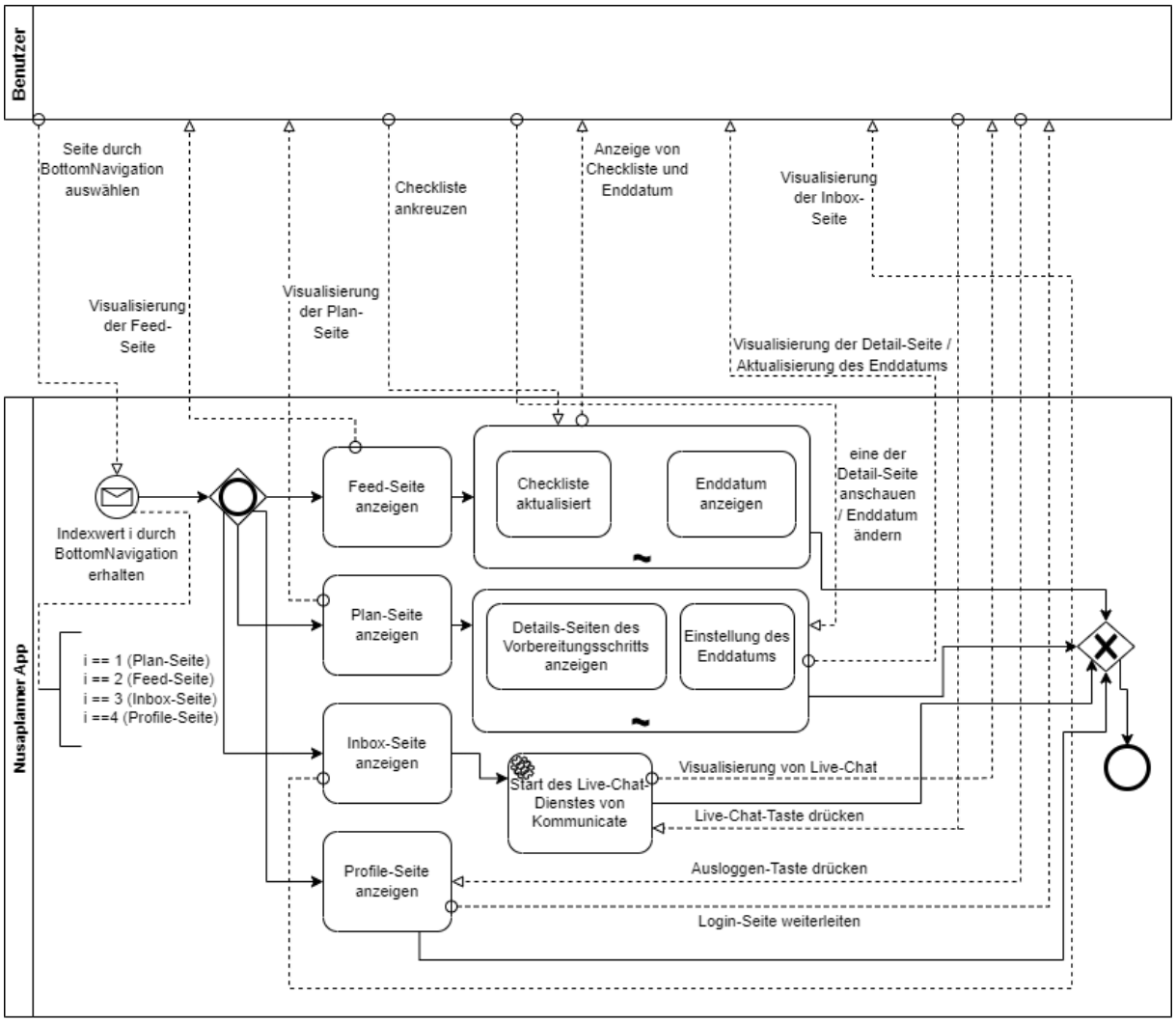

Abb. 3.6: ein BPMN-Diagramm, das einen kurzen Überblick über einige der Funktionen der App zeigt, nachdem sich der Benutzer bereits authentifiziert hat

Um die Navigation zwischen den Seiten der App zu erleichtern, wurde eine untere Navigationsleiste eingeführt. Auf der *Plan*-Seite kann der Benutzer die geschätzte Abfahrtszeit über der ToDo-Liste sehen, die die Liste der wichtigen Vorbereitungsschritte enthält. Diese ToDo-Liste ist die Standardliste der Vorbereitungsphasen und kann nicht geändert werden (Hinzufügen oder Löschen von Phasen). Dies wird angewandt, damit der Benutzer nicht versehentlich einen der wichtigen Schritte löschen kann. Außerdem wurden diese Schritte auf der Grundlage langer Recherchen des Entwicklers über die Studienvorbereitung von Indonesiern in Deutschland erstellt. Der Benutzer kann jedoch immer noch das zugehörige Kontrollkästchen ankreuzen, so dass er die Schritte nacheinander bis zum Ende der Vorbereitung verfolgen kann.

Auf der *Feed*-Seite kann der Benutzer die detaillierten Erklärungen zu jeder wichtigen Studienvorbereitung einsehen und lesen. Darüber hinaus kann der Nutzer auf dieser Seite auch sein Vorbereitungsdatum ändern, indem er eines der Tools namens Kalender verwendet. Diese Funktion wird angeboten, um das mögliche Problem zu lösen, dass der Benutzer nach der Anmeldung irrtümlich ein falsches Startdatum für die Studienvorbereitung eingibt. Das Enddatum kann dann auf der Grundlage des angegebenen Startdatums aktualisiert werden. Neben dem Kalender ist auch der Live-Chat-Service eines der Tools, die der Benutzer sowohl auf der *Feed*-Seite als auch auf der *Inbox*-Seite findet.

Der Zweck des Live-Chat-Dienstes besteht darin, Hilfe zu leisten, wenn der Benutzer Schwierigkeiten bei der Verwendung der App oder Verwirrung bei einem Schritt der Studienvorbereitung hat. Der Benutzer wird dann zu einem Chat-Fenster weitergeleitet und von einem Chat-Bot begrüßt, die allgemeinen Informationen zu den häufigsten Unklarheiten bei jeder Studienvorbereitung liefert. Der Nutzer kann auch individuelle oder persönliche Fragen zu den Schritten stellen und wird dann an den erfahrenen Kundenservice weitergeleitet.

Wenn sich der Benutzer abmelden möchte, wird auf der *Profile*-Seite die Abmelden-Taste angezeigt. Wenn der Benutzer die Taste klickt, wird er dann auf die *SignIn*-Seite weitergeleitet. In diesem Unterabschnitt werden nur die allgemeinen Funktionalitäten beschrieben. Wie der Informationsfluss zwischen der Benutzeroberfläche und dem externen Dienst, z.B. dem Backend-Dienst, funktioniert, wird im nächsten Unterabschnitt ausführlich erläutert.

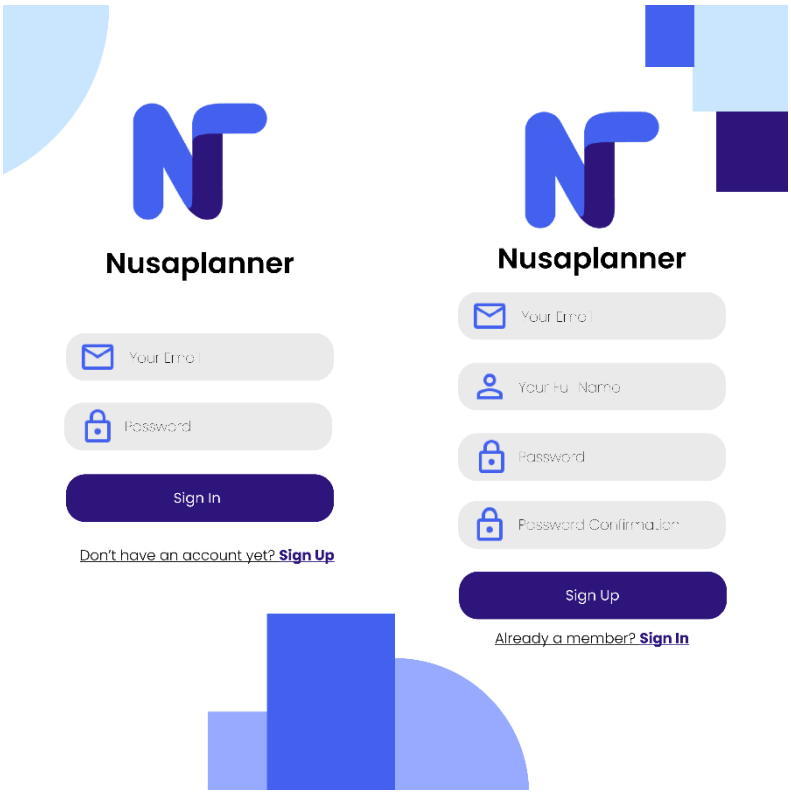

Abb. 3.7: UI-Design von *SignIn*- und *SignUp*-Seiten

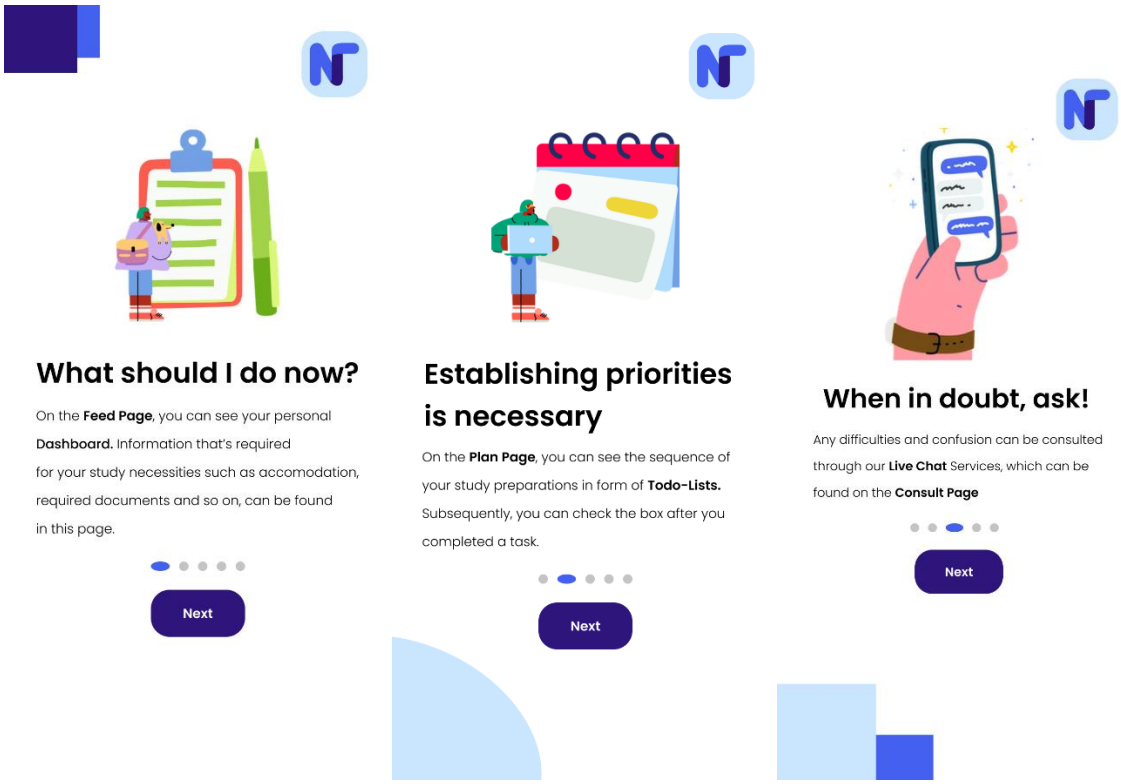

Abb. 3.8: UI-Design von *Onboarding*-Seite (*Onboarding Screens*)

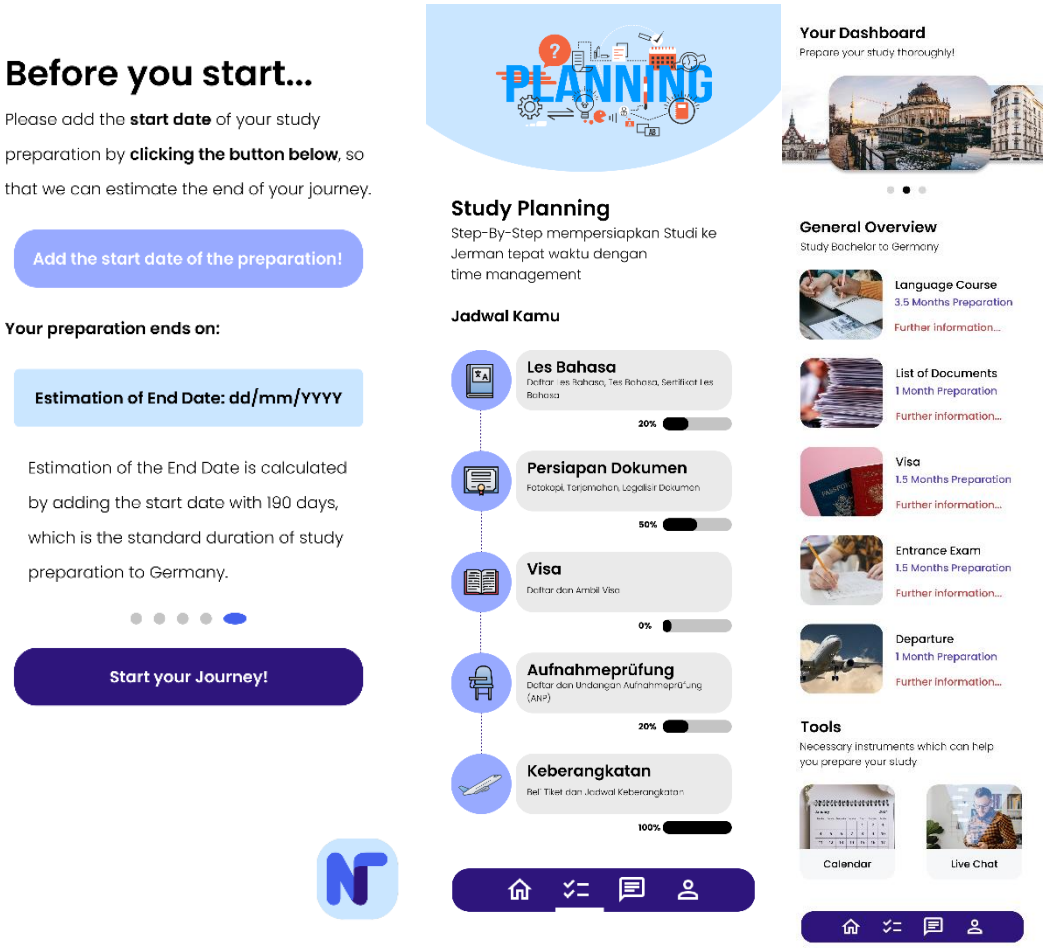

Abb. 3.9: UI-Design von *AddDate*-, *Plan*- und *Feed*-Seiten

3.3. Implementierung der App

Der Überblick über die Funktionsweise der App wurde bereits im letzten Unterabschnitt kurz beschrieben. In diesem Unterkapitel wird der Entwicklungsprozess der App näher erläutert. Die folgenden Prozesse werden ausführlich erörtert:

- Erstellung des Backend-Dienstes mit dem Laravel Framework,
- Herstellung der Verbindung zwischen der Benutzeroberfläche und dem Backend-Dienst mithilfe des Dio-Plugins,
- Realisierung der Benutzeroberfläche und der Einstellung des Live-Chat-Dienstes mit dem Kommunicate-Plugin,
- das Endprodukt der App (.APK-Daten).

3.3.1. Erstellung des Backend-Dienstes mit Laravel

Bevor ein neues Projekt in Laravel erstellt werden kann, müssen einige notwendige Tools installiert werden, wie z.B.:

- PHP >= 7.3.,
- Einen localhost Webserver (for example WAMP or XAMPP),
- Composer als Software zur Verwaltung von Abhängigkeiten bzw. *Dependencies* für PHP,
- Datenbank (MySQL), die bereits auf dem lokalen Webserver installiert ist.

Um das Framework zu installieren und ein Laravel-Projekt zu erstellen, wird der folgende Befehl in die Befehlszeile eingegeben:

```
composer create-project laravel/laravel fakeapione
```

Listing 3.1: Composer-Befehl zum Erstellen eines neuen Laravel-Projekts

Aus dem obigen Beispiel wird ein Verzeichnis namens fakeapione erstellt und alle primären Laravel-Dateien werden in dieses Verzeichnis geladen. Die wichtigen Ordner für die Entwicklung des Projekts sind die Controllers- und Models-Ordner. Der Controller ist für die Bearbeitung aller eingehenden HTTP-Anfragen zuständig. Mit anderen Worten, er vermittelt die Verbindung zwischen HTTP und der Anwendung. In diesem Fall wird also das Modell mit der in Flutter entwickelten Benutzeroberfläche verbunden. Die gesamte Domain-Logik bzw. Geschäftslogik der App wird im Ordner Models gespeichert. Standardmäßig wird das Modell User bereits mit Standardattributen wie Name, E-Mail, Passwort und Zeitstempel erstellt.

Um die Anforderungen der App zu erfüllen, muss noch ein Modell erstellt werden, das die Logik der Todoliste in unserer App enthält. Dieses Modell wird daher Todolist-Modell genannt. Das User-Modell steht in einer *one-to-one*-Beziehung zum Todolist-Modell. Da Laravel Eloquent als ORM enthält, ist es einfach, eine Beziehung zwischen diesen beiden Modellen herzustellen. Im User-Modell wird der Code in **Listing 3.2** geschrieben, um die Beziehung zwischen dem User und der Todolist zu definieren. Als umgekehrte Funktion wird der Code in **Listing 3.3** im Todolist-Modell geschrieben. Anschließend wird erfolgreich eine *one-to-one*-Beziehung erstellt.

```
public function todolist() {
        return $this->hasOne(Todolist::class);
    }
```

Listing 3.2: hasOne-Funktion in User-Modell

```
public function user() {
        return $this->belongsTo(User::class);
    }
```

Listing 3.3: belongsTo-Funktion in Todolist-Modell

Wie bereits in der kurzen Übersicht über die Funktionen der App erläutert, muss der Benutzer nur seine E-Mail-Adresse und sein Passwort eingeben, um sich anzumelden. Nachdem der Benutzer seine erforderlichen Informationen eingegeben hat, wird das Zugriffstoken generiert und der Benutzer wird authentifiziert. Wenn die Benutzervalidierung aufgrund ungültiger Anmeldedaten fehlschlägt, wird der Fehler 401 ausgegeben. Bei der Registrierung sind zusätzliche Informationen erforderlich, z. B. der Benutzername des Benutzers. Die Spalte Start- und Enddatum wird ebenfalls definiert, nachdem sich der Benutzer angemeldet hat. Diese Prozesse können dann implementiert werden, nachdem die folgenden Codes deklariert wurden:

```
public function login(Request $request){
        $validator = Validator::make($request->all(), [
            'email' => 'required|email',
            'password' => 'required|string|min:6',
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        if (! $token = auth()->attempt($validator->validated())) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }

        return $this->createNewToken($token);
}
```

Listing 3.4: login-Funktion

```
public function register(Request $request) {
        $validator = Validator::make($request->all(), [
            'name' => 'required|string|between:2,100',
            'email' => 'required|string|email|max:100|unique:users',
            'password' => 'required|string|confirmed|min:6',
            'is_splash_two' => 'required|integer|between:0,1',
            'is_splash_three' => 'required|integer|between:0,1',
            'is_splash_four' => 'required|integer|between:0,1',
            'is_splash_five' => 'required|integer|between:0,1',
            'is_splash_six' => 'required|integer|between:0,1',
            'date_start' => 'required|string',
            'date_end' => 'required|string',
        ]);

        if($validator->fails()){
            return response()->json($validator->errors()->toJson(), 400);
        }

        $user = User::create(array_merge(
                    $validator->validated(),
                    ['password' => bcrypt($request->password)]
                ));

        $token = JWTAuth::fromUser($user);
        return $this->createNewTokenRegis($token, $user);

}
```

Listing 3.5: register-Funktion

Irgendwann gibt der Benutzer das Startdatum der Studienvorbereitung ein oder möchte das Datum möglicherweise ändern. Um das Datum des zugehörigen Benutzers zu aktualisieren, wird die E-Mail des Benutzers als Eingabe verwendet, die später lokal auf dem Gerät gespeichert wird. Die Methode zur Definition dieses Prozesses wird unten erklärt:

```
$user = User::whereEmail($request->input('email'))->update([
                'date_start' => $request->input('date_start'),
                'date_end' => $request->input('date_end'),
]);
```

Listing 3.6: Funktion zur Aktualisierung von Start- und Enddatum der Studienvorbereitung

Nachdem alle erforderlichen Methoden zum Speichern und Abrufen der persönlichen Benutzerdaten und der Werte der Kontrollkästchen definiert wurden, kann der Pfad zu den Controllern oder API-Routen in einer separaten Datei namens api.php definiert werden. Alle CRUD-Operationen (Erstellen, Lesen, Aktualisieren, Löschen) werden in dieser Datei definiert.

3.3.2. Herstellung der Verbindung zwischen UI und dem Backend-Dienst mithilfe des Dio-Plugins

Vor der Entwicklung ist es wichtig, die notwendigen Tools in das Flutter-Projekt zu importieren. Die Plugins werden in der Datei pubspec.yaml deklariert, die bei der Erstellung eines Flutter-Projekts automatisch erstellt wird und die Aufgabe hat, die Pakete, Versionen und Ressourcen von Drittanbietern zu verwalten.

```
23  dependencies:
24    flutter:
25      sdk: flutter
26
27
28    # The following adds the Cupertino Icons font to your application.
29    # Use with the CupertinoIcons class for iOS style icons.
30    cupertino_icons: ^1.0.2
31    google_fonts: ^2.1.0
32    carousel_slider: ^4.0.0
33    intl: ^0.17.0
34
35    # bottom navigation bar
36    floating_bottom_navigation_bar: ^1.5.2
37    provider: ^6.0.1
38
39    # http
40    dio: ^4.0.0
41
42    # local storage
43    flutter_secure_storage: ^4.2.1
44    shared_preferences: ^2.0.9
45    kommunicate_flutter: ^1.3.1
46    flutter_svg: ^1.0.0
```

Abb. 3.10: Überblick über die Datei pubspec.yaml

Um API-Anfragen erfolgreich durchzuführen, wird das Dio-Plugin angewendet. Dio ist eine leistungsstarke Netzwerkbibliothek, die Interceptors, globale Konfiguration, FormData, Abbruch von Anfragen, Herunterladen von Dateien und Timeout [31] [32] unterstützt. Mit der Funktion Interceptors ist es möglich, jede Anfrage abzufangen, die gestellt wird. Sie ist nützlich für die Autorisierung mit JSON Web Token (JWT) und das Parsen von JSON. Die JWT-Autorisierungs- und Akzeptanz-Header können jedes Mal hinzugefügt werden, wenn die

Anfrage abgefangen wird. Ein Header ist eine der JWT-Strukturen, die aus zwei Teilen besteht: dem Typ des Tokens (JWT) und dem Signieralgorithmus (HMAC SHA256 oder RSA) [33].

Abb. 3.11: JWT Header [33]

Für die Einrichtung der Netzwerkoperationen wird eine eigene Dio-Klasse erstellt. Zunächst wird die Basis-URL des Backend-Dienstes oder die API-URL definiert. Anschließend können die spezifischen Header, d.h. der Accept- und der Authorization-Header, zur Liste der Interceptoren hinzugefügt werden. Für den Wert des Accept-Headers wird application/json geschrieben und für den Wert des Authorization-Headers wird *Bearer Token* als Eingabe eingegeben. Das *Bearer Token* ist eine kryptische Zeichenfolge, die vom Server automatisch generiert wird, nachdem sich der Benutzer angemeldet hat [34]. Mit diesen Anweisungen kann der Benutzer nun authentifiziert werden.

```
Dio dio() {
  Dio dio = new Dio();

  dio.options.baseUrl =
'https://nusaplannerbackend.000webhostapp.com/api/';

  dio.interceptors.add(InterceptorsWrapper(onRequest: (request, handler)
async {
    request.headers['Accept'] = 'application/json';
    var token = await storage.read(key: 'token');

    if (token.toString().isNotEmpty) {
      request.headers['Authorization'] = 'Bearer $token';
    }
    return handler.next(request);
  }));
  return dio;
}
```

Listing 3.7: Dio-Klasse

Um die benötigten Funktionen für die Ausführung der Netzwerkanfragen zu definieren, wird eine weitere separate Klasse namens Auth erstellt. Diese Klasse bearbeitet alle Arten von API-Anfragen, z.B. das Abrufen einzelner Benutzerdaten durch Definition einer GET-Anfrage oder das Senden einer Anfrage zur Erstellung eines neuen Benutzers an die API mit einer POST-Anfrage. Zur Verwaltung der Daten, die von einer REST-API-Anfrage zurückgegeben werden,

muss eine Modellklasse definiert werden. Um zum Beispiel die von der API empfangenen Benutzerdaten zu speichern, wird eine Modellklasse User erstellt (siehe **Abb. 3.9**). In dieser Anwendung werden nur zwei Modellklassen erstellt, nämlich die User- und Todolist-Klasse. In jeder Modellklasse ist ein Fabrikkonstruktor bzw. *factory constructor* implementiert, der nach dem Parsen der JSON-Response eine neue zugehörige Instanz (ein User- oder Todolist-Objekt) erzeugt. Mit anderen Worten, der Fabrikkonstruktor ist für die Umwandlung des geparsten JSON in ein neues Objekt oder eine Instanz verantwortlich. Ein Beispiel für eine API-Funktion in Flutter, in diesem Fall die Login-Funktion, ist in **Abb. 3.12** zu sehen.

```
Future signin(
      {Map? data, required Function success, required Function error})
async {
    try {
      Dio.Response response =
          await dio().post('auth/login', data: json.encode(data));

      var token = json.decode(response.toString())['access_token'];
      var id = json.decode(response.toString())['user']['id'];
      var email = json.decode(response.toString())['user']['email'];
      var dateEnd = json.decode(response.toString())['user']['date_end'];

      this._setStoredToken(token);

      await UserSimplePreferences.setDate(dateEnd);

      this.attempt(token: token);

      notifyListeners();

      success();
    } catch (e) {
      error();
    }
}
```

Listing 3.8: Sign-In-Methode

```
class User {
  int id;
  String name;
  String email;
  int isSplashScreenTwo;
  String dateStart;
  String dateEnd;

  User(
      {required this.id,
      required this.name,
      required this.email,
      required this.isSplashScreenTwo,
      required this.dateStart,
      required this.dateEnd});

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
        id: json['id'],
        name: json['name'],
        email: json['email'],
        isSplashScreenTwo: json['splashTwo'],
        dateStart: json['dateStart'],
        dateEnd: json['dateEnd']);
  }
}
```

Listing 3.9: User-Klasse

3.3.3. Realisierung der Benutzeroberfläche und der Einstellung des Live-Chat-Dienstes mit dem Kommunicate-Plugin

Jetzt kann die Entwicklung der Benutzeroberfläche begonnen werden. Die erste Seite, die dem Benutzer nach dem Öffnen der App angezeigt wird, ist die *SignIn*-Seite. Auf dieser Seite findet ein simpler Prozess statt. Wenn der Benutzer über ein Konto verfügt und die richtigen Anmeldedaten eingibt, wird er authentifiziert und kann sich anmelden. Wenn der Benutzer die falschen Anmeldedaten eingibt, wird ein Fehlerdialog angezeigt. Wenn der Benutzer noch kein Konto hat, kann er auf die *SignUp*-Seite umgeleitet werden. Nach Eingabe der richtigen Anmeldedaten kann ein neues Konto erstellt werden und der Benutzer wird authentifiziert.

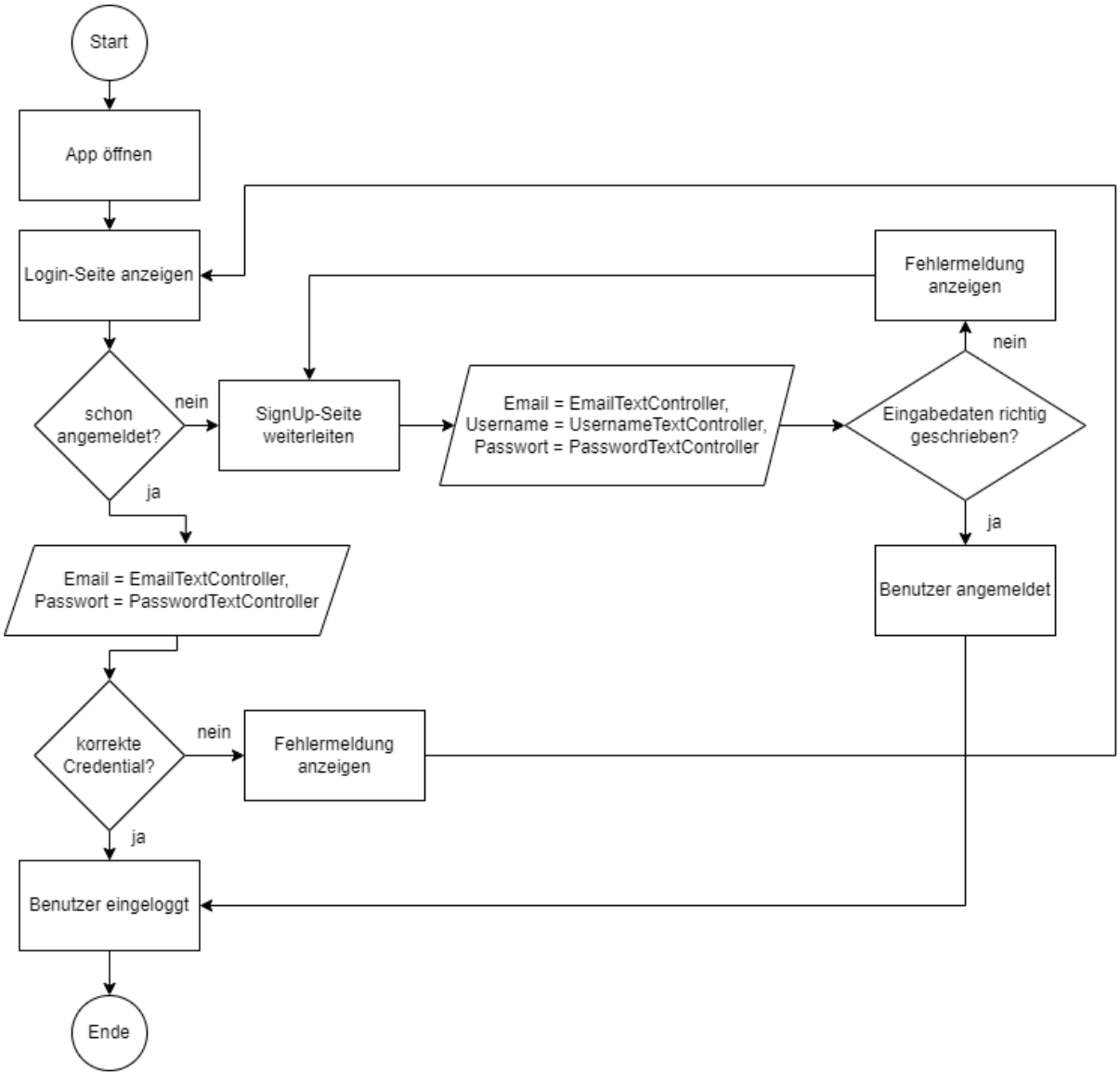

Abb. 3.12: Programmablaufplan für den Anmelde- und Registrierungsprozess

Nachdem der Benutzer das Konto erstellt hat, werden die persönlichen Daten des Benutzers in der Datenbank gespeichert. Außerdem wird der Zugriff auf die Inhalte durch den Server gewährt. Damit der Benutzer jedoch eingeloggt oder authentifiziert bleibt, wenn die App zufällig geschlossen wird, ist eine lokale Speicherung erforderlich. Daher wird das Paket Fluttersecurestorage verwendet, um das verschlüsselte JWT sicher auf dem Gerät selbst zu speichern, da dieses Paket häufig zum Speichern kritischer Daten verwendet wird. Daher muss sich der Benutzer nicht jedes Mal neu anmelden, wenn die App versehentlich geschlossen wird. Fluttersecurestorage kann durch Einfügen des Schlüssels und des Wertes der Daten verwendet werden. In diesem Fall wären die Daten das Zugriffstoken, das von JSON in einen String umgewandelt werden sollte.

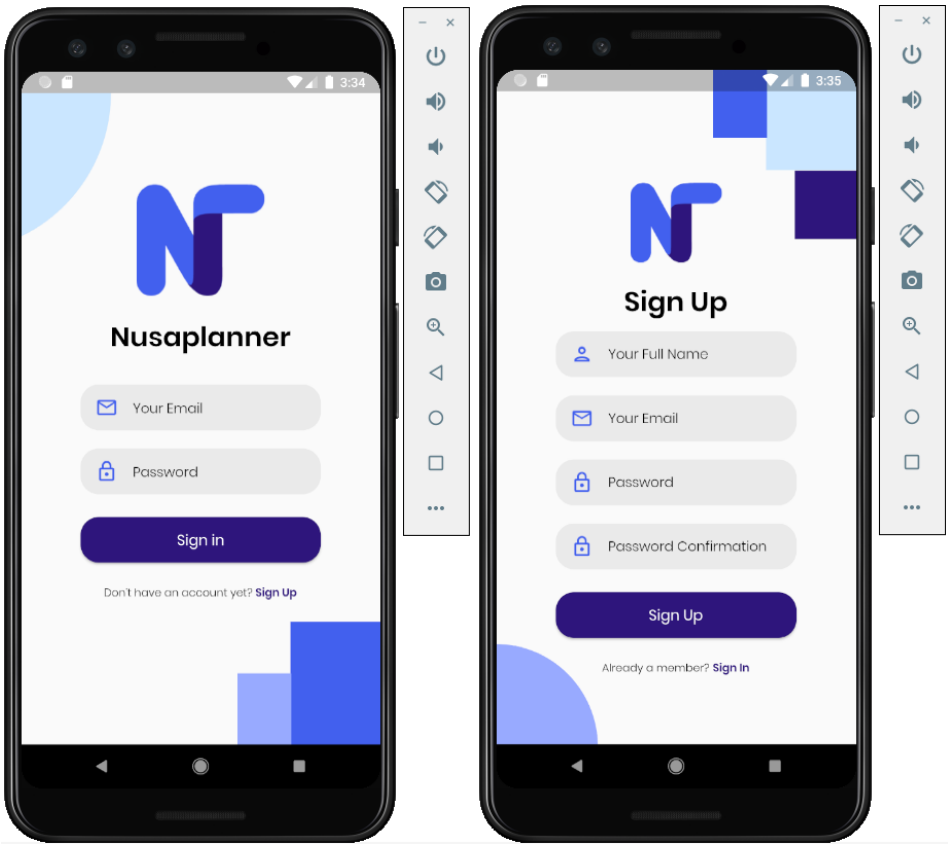

Abb. 3.13: UI von SignIn und SignUp

Ein weiterer lokaler Speicher ist das SharedPreferences-Plugin, in dem eine kleine Sammlung von Daten gespeichert wird, in diesem Fall das Enddatum der Studienvorbereitung. Um die Daten zu speichern, muss die von der Klasse SharedPreferences bereitgestellte Setter-Methode definiert werden. Diese Methode führt zwei Prozesse aus. Erstens werden die Schlüsselwertdaten gleichzeitig aktualisiert und dann können die Daten auf der Festplatte gespeichert werden [35].

Den genauen beschriebenen Prozess der Anmeldung und Registrierung ist in **Abb. 3.14** zu sehen. Das BPMN-Diagramm unten zeigt den Prozess der Benutzerauthentifizierung und -autorisierung sowie die Speicherung der erforderlichen Daten auf dem Server und lokal auf dem Gerät. Was das Design der Benutzeroberfläche betrifft, so zeigt **Abb. 3.13** das endgültige Design der Benutzeroberfläche für die Anmeldung und Registrierung auf dem Emulator.

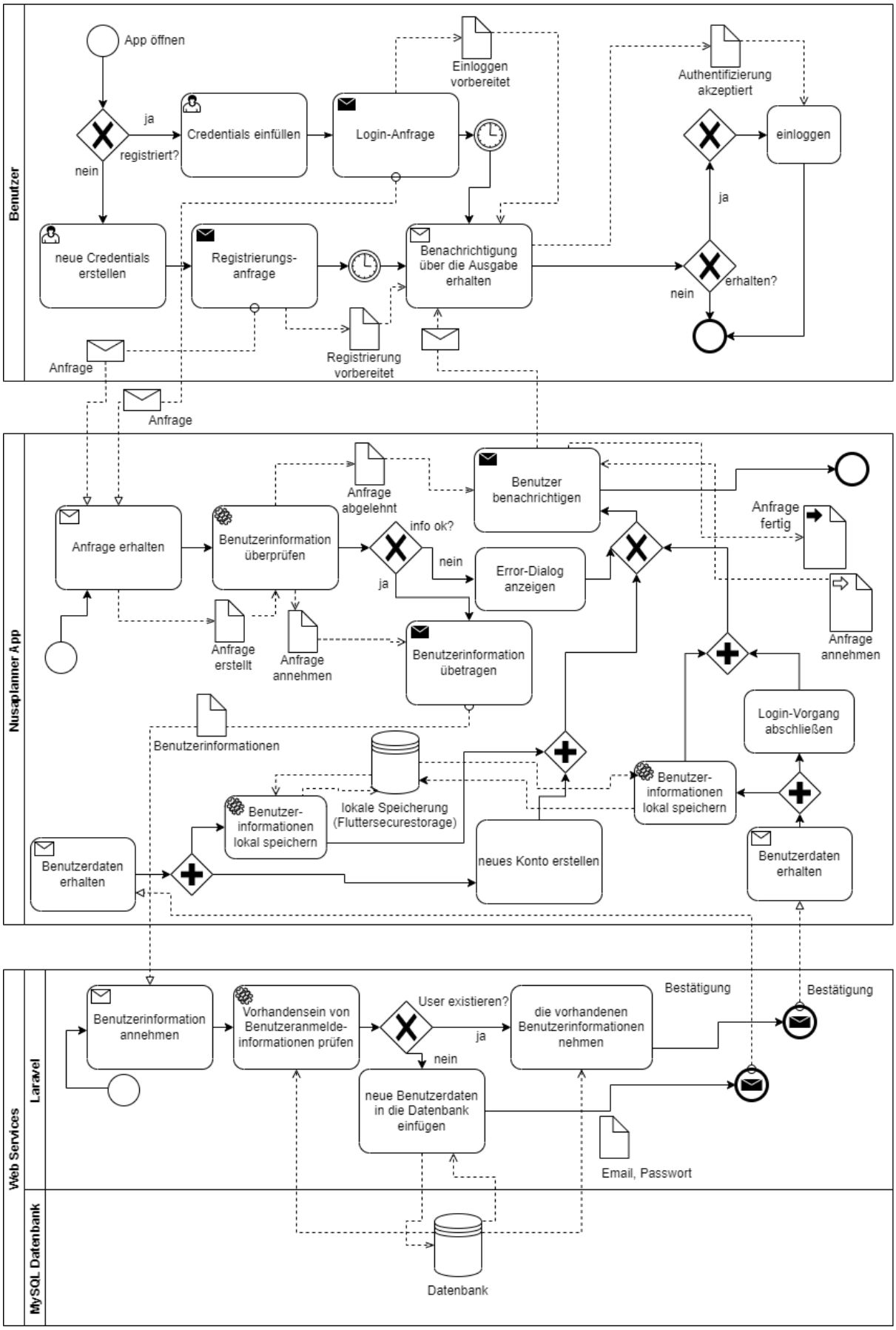

Abb. 3.14: BPMN-Diagramm von den Login- und Registrierung-Funktionen

Wie bereits erwähnt, wird der Benutzer nach der Registrierung zu den *Onboarding*-Seiten weitergeleitet. Nachdem er diese Seiten durchlaufen hat, muss das Startdatum der Studienvorbereitung eingegeben werden (siehe **Abb. 3.16**), von dem aus dem Enddatum der Studienvorbereitung geschätzt werden kann. Das Enddatum wird später in der Datenbank und auf dem lokalen Gerät mit dem SharedPreferences-Plugin gespeichert. Der Prozess des Speicherns der Daten in der Datenbank und auf dem lokalen Gerät ist unten dargestellt.

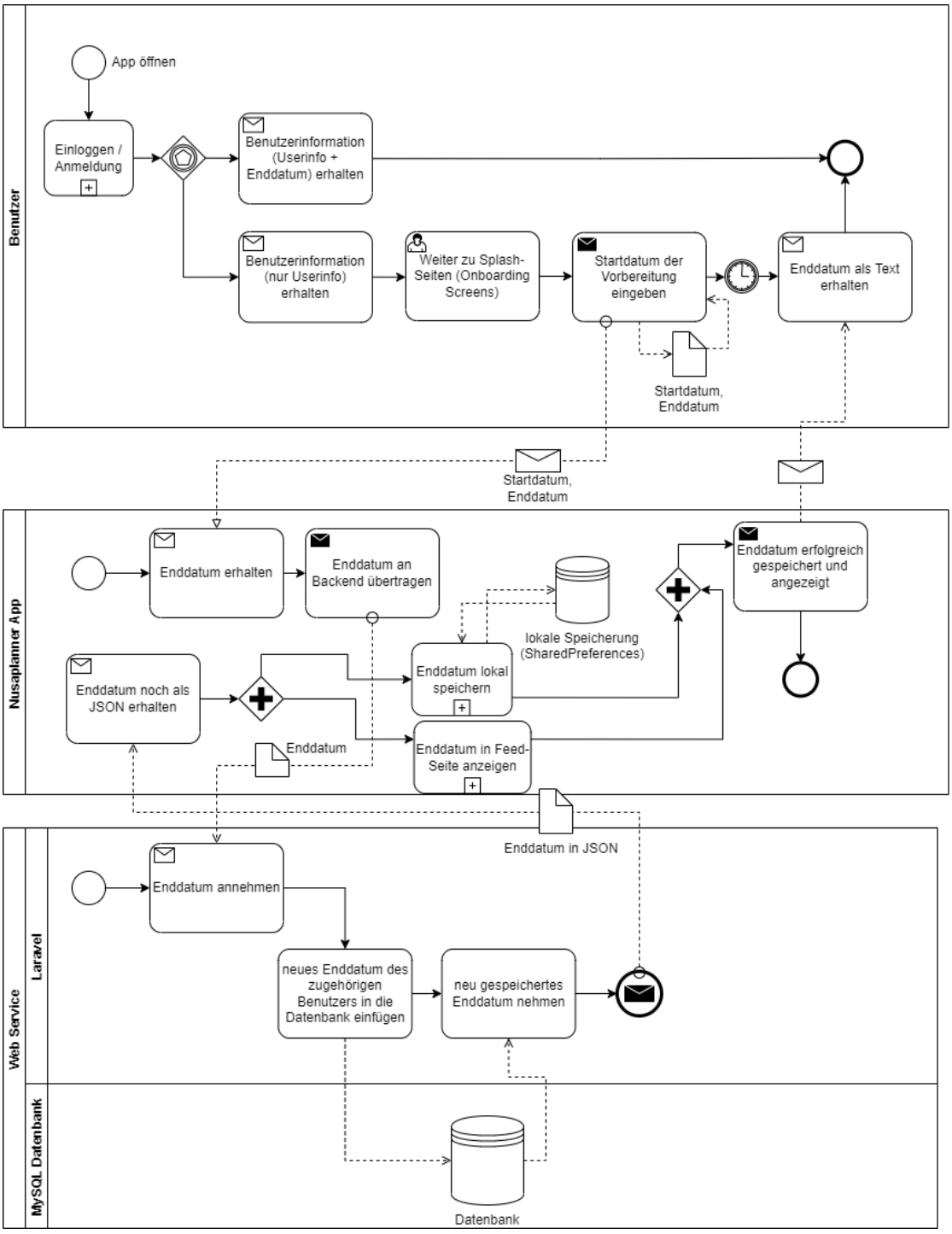

Abb. 3.15: BPMN-Diagramm von der Add and Save Date-Funktion

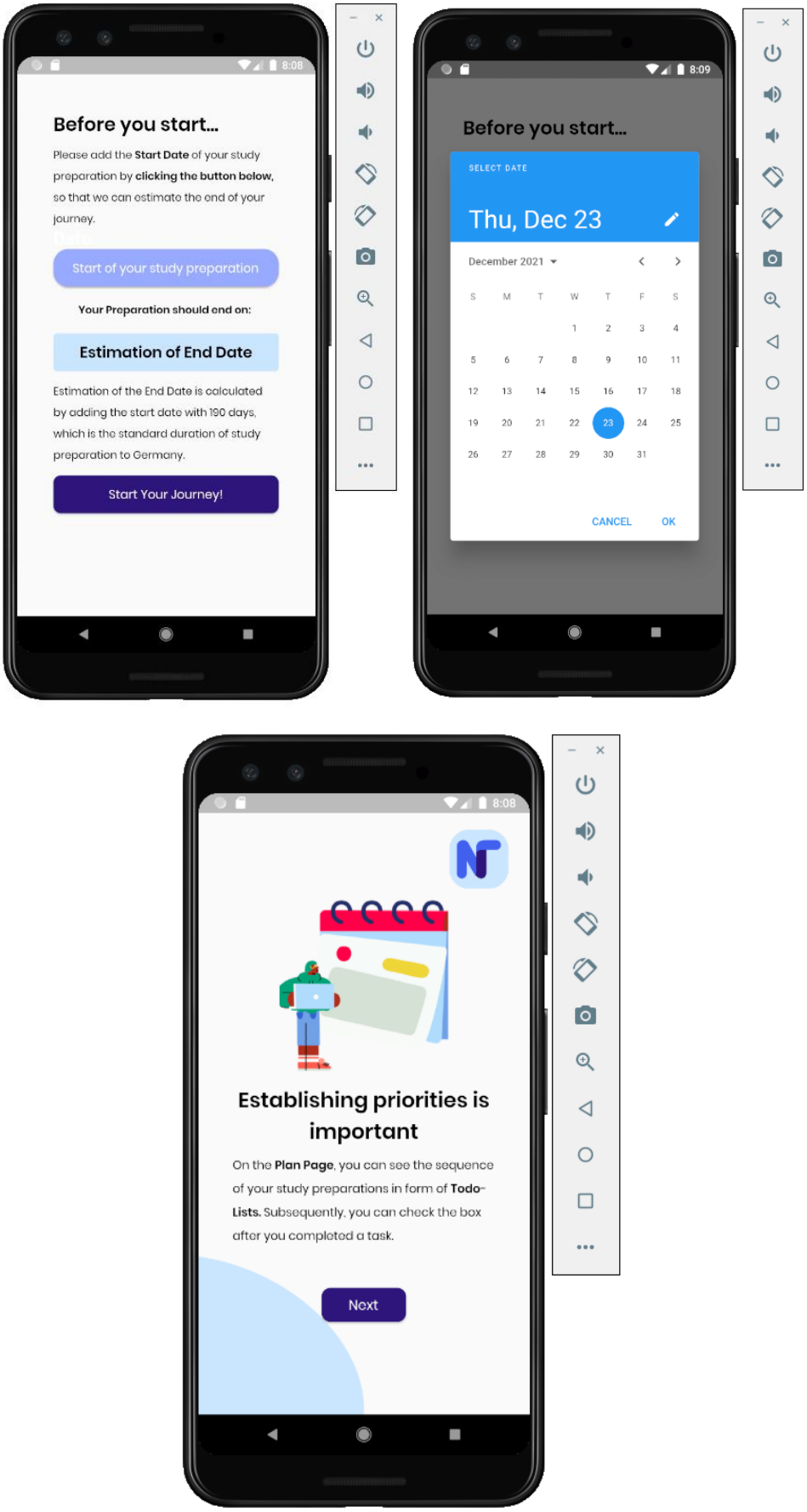

Abb. 3.16: UI von *Add Date* und *Onboarding Screen*

Nach der erfolgreichen Registrierung und der Eingabe des Startdatums für die Studienvorbereitung oder der erfolgreichen Anmeldung ist die Seite Plan die erste Hauptseite, die in der App angezeigt wird. Diese Seite enthält das voraussichtliche Abreisedatum des Benutzers, das zuvor mit dem SharedPreferences-Plugin lokal gespeichert wurde, sowie die Liste der Studienvorbereitungsschritte. Der Benutzer kann dann das Kästchen der Checkliste ankreuzen und sie wird direkt gespeichert.

Wie bereits erläutert, kann jeder Benutzer seine eigenen Kästchenwerte sowohl aus dem lokalen Speicher als auch aus der Datenbank abrufen. Der Benutzer kann nur auf seine eigenen Kontrollkästchendaten zugreifen, indem er seinen eigenen User-ID als Fremdschlüssel angibt. Der Fremdschlüssel wird dann vom Benutzer verwendet, um die Werte der Kontrollkästchen vom Server abzurufen. Anschließend wird die aktualisierte Liste der Kontrollkästchen auf der *Plan*-Seite angezeigt (siehe **Abb. 3.18**).

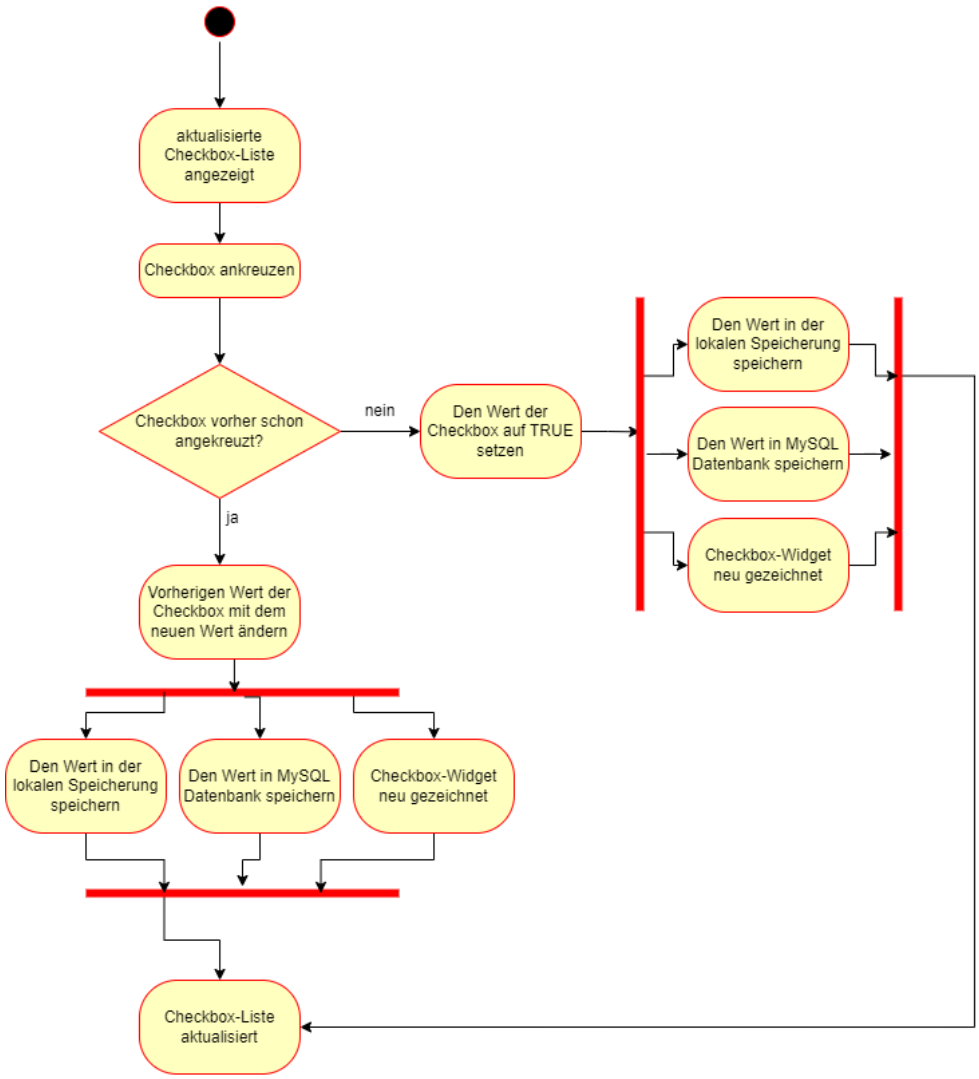

Abb. 3.17: Aktivitätsdiagramm für Modifizierung der Werte von Checklisten

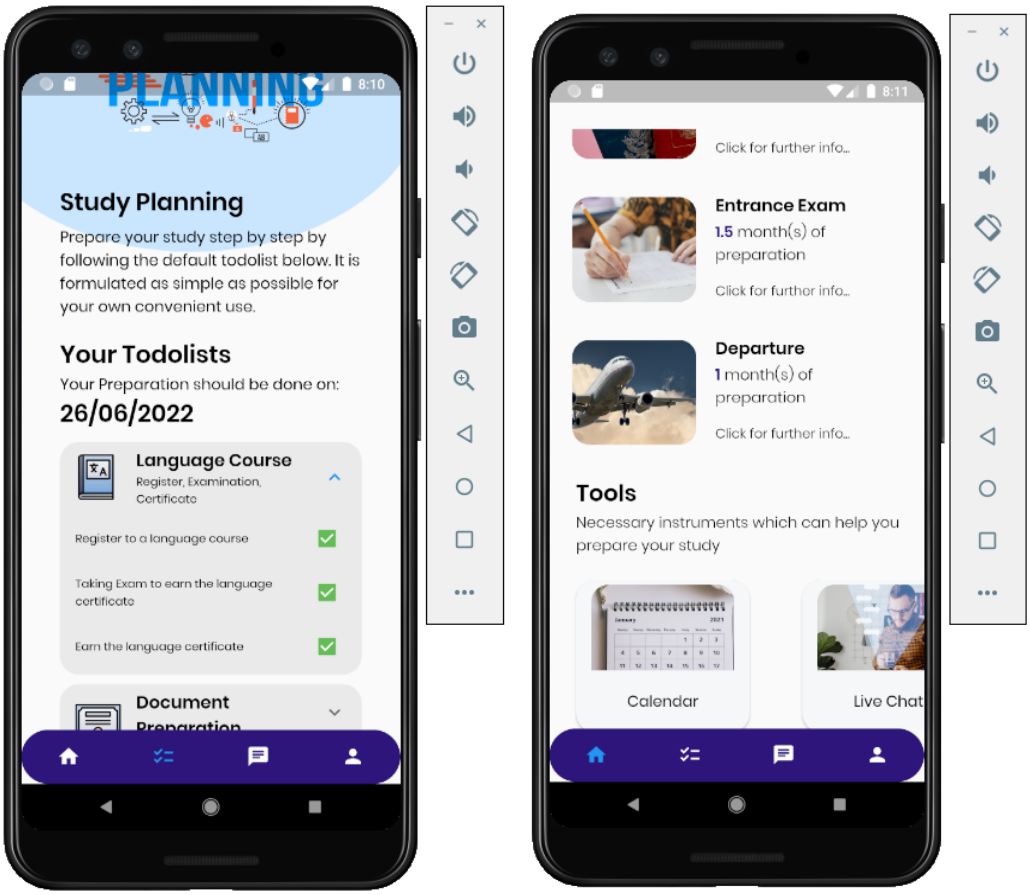

Abb. 3.18: UI von *Plan*- und *Feed*-Seiten

Die *Feed*-Seite enthält wichtige Informationen zu den Vorbereitungsschritten, die in Form einer Folge von Detailbildern dargestellt werden (siehe **Abb. 3.19**). Der Schritt Sprachkurs wird als einfache Visualisierung dieser Funktion genommen. Bevor der Benutzer den Sprachkurs besucht, sollte er sich vorher informieren, wo er sich anmelden wird. Danach ist es wichtig, auf wichtige Termine und die Einschreibung bei der entsprechenden Sprachschule zu achten. Und schließlich sollte der Benutzer bereits wissen, welchen Zertifikatsabschluss er anstreben möchte. Diese verschiedenen Schritte werden in verschiedene Detailbildschirme verpackt. Jeder Bildschirm enthält wichtige Informationen zu dem jeweiligen Schritt.

Um den Live-Chat-Dienst zu starten, wird der in **Listing 3.10** angegebene Code ausgeführt. Der Benutzer wird zunächst an den vom Plugin bereitgestellten Chatbot weitergeleitet. Wenn der Benutzer bestimmte benutzerdefinierte Fragen hat, kann er an den Kundendienst weitergeleitet werden. Der zugehörige Benutzer kann anhand der Benutzerkennung bzw. User-ID erkannt werden, die bereits bei der Registrierung angegeben wurde. Die lokal gespeicherte Benutzerkennung wird abgerufen, um einen Benutzer von einem anderen zu unterscheiden.

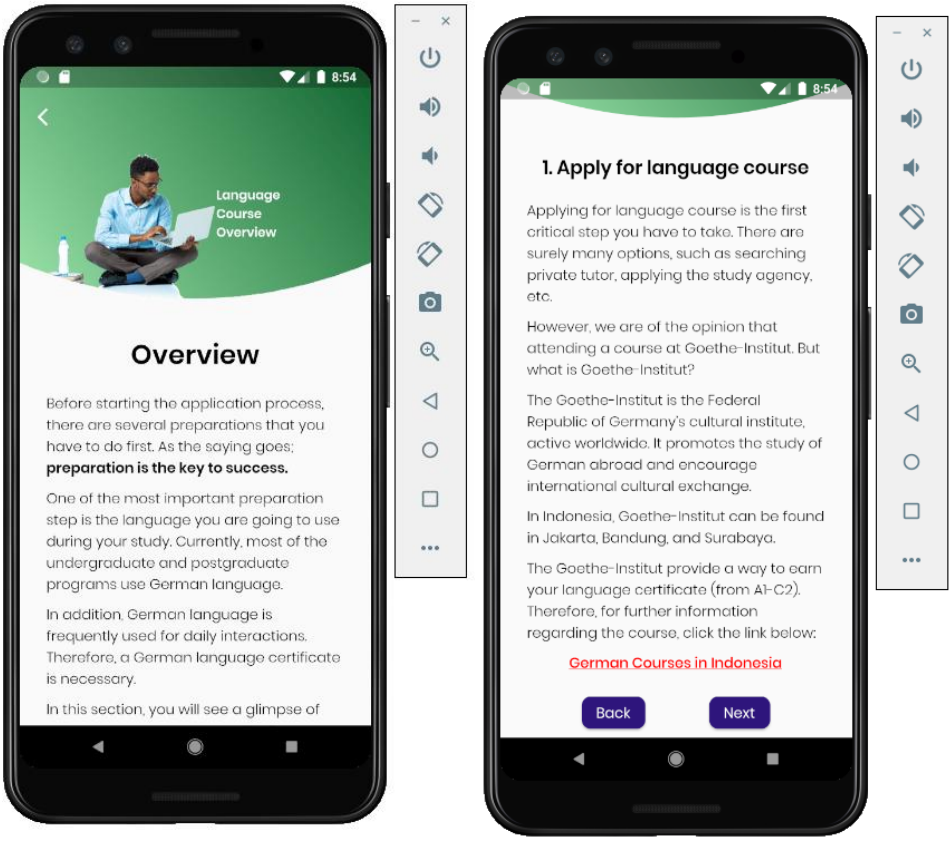

Abb. 3.19: UI von *Detail*-Seiten

```
dynamic user = {
    'userId':
        '${auth.user.id}', //Replace it with the userId of the logged in
user
  };
  dynamic conversationObject = {
    'appId':
        '34ff17301367d14868f0dff1390c0b832', // The
[APP_ID](https://dashboard.kommunicate.io/settings/install) obtained from
Kommunicate dashboard.
    'kmUser': jsonEncode(user)
  };

  KommunicateFlutterPlugin.buildConversation(
          conversationObject)
      .then((clientConversationId) {
    print("Conversation builder success : " +
        clientConversationId.toString());
  }).catchError((error) {
    print("Conversation builder error : " +
        error.toString());
  });
}
```

Listing 3.10: Code zum Starten der Kommunicate Chatbox

Wenn sich der Benutzer schließlich abmelden möchte, findet er die Abmelden-Taste auf der *Profile*-Seite. Zunächst besteht die Profilseite aus den persönlichen Informationen eines Benutzers, einer Hilfe-Taste, die Unterstützung bei der Verwendung der App in Form von Einführungsbildschirmen bietet (ähnlich den Bildschirmen, die der Benutzer nach der Registrierung vorfindet), und einer Abmelden-Taste. Aufgrund der begrenzten Zeit für die Entwicklung der App werden die persönlichen Daten des Benutzers und die Hilfebildschirme jedoch erst in Zukunft entwickelt. Die Schaltfläche zum Abmelden ist noch vorhanden, damit der Benutzer die App ordnungsgemäß beenden kann.

Die Gültigkeit des Tokens vom Benutzer wird nicht nur für die Authentifizierung und Autorisierung verwendet, sondern auch für die erneute Authentifizierung, wenn der Benutzer die App unbeabsichtigt verlässt. Um sich abzumelden, müssen die persönlichen Daten des Benutzers, wie z.B. der Token und das Enddatum der Studienvorbereitung, aus dem lokalen Speicher gelöscht werden, so dass der Benutzer nicht mehr authentifiziert ist.

```
void signOut({required Function success}) async {
    try {
      await dio().post('auth/logout');
      this._setUnauthenticated();
      KommunicateFlutterPlugin.logout();
      notifyListeners();

      success();
    } catch (e) {}
}
```

Listing 3.11: signOut-Methode

Es handelt sich um einen asynchronen Prozess, weil zuerst der lokale Speicher (in diesem Fall Fluttersecurestorage) aufgerufen wird, um den Token zu löschen, und der SharedPreferences-Speicher aufgerufen wird, um das Enddatum des Benutzers zu löschen (**Listing 3.12**). Schließlich wird der Benutzer zur Anmeldeseite zurückgeleitet und es werden Snackbars angezeigt, die besagen, dass der Benutzer sich gerade erfolgreich abgemeldet hat.

```
void _setUnauthenticated() async {
    this.authenticated = false;
    this.authenticatedUser = null;
    this.createdTodoList = null;
    await storage.delete(key: 'token');

    SharedPreferences prefs = await SharedPreferences.getInstance();
    await prefs.clear();
}
```

Listing 3.12: _setUnauthenticated-Methode

3.3.4. Endprodukt der App (.APK Daten)

Nachdem die App mehrmals getestet und beobachtet wurde, ob die App erfolgreich und fehlerfrei funktioniert und somit die zuvor genannten Anforderungen erfüllt, kann die App dann bereitgestellt werden. Bevor die App bereitgestellt werden kann, sind einige Dinge zu beachten. Erstens ist es notwendig, die App umzubenennen, denn der Name der App ist bisher der Projektname selbst. Zweitens sollte auch das Icon der App festgelegt werden. Um das benutzerdefinierte App-Icon zu erstellen, wird ein Paket namens flutter_launcher_icons verwendet. Das Paket und der Bildpfad für das Logo der App müssen in der Datei pubspec.yaml (**Abb. 3.20**) definiert werden und der Befehl in **Listing 3.13** wird ausgeführt.

```
56  dev_dependencies:
57    flutter_test:
58      sdk: flutter
59    flutter_launcher_icons: "^0.9.2"
60
61  flutter_icons:
62    android: true
63    ios: true
64    image_path: "assets/images/Nusatutor_logo_blue.png"
```

Abb. 3.20: flutter_launcher_icons-Package in pubspec.yaml file

```
flutter pub run flutter_launcher_icons:main
```

Listing 3.13: Kommandozeile zum Ändern des App-Startsymbols

Um den neuen App-Namen festzulegen, kann der Wert des android:label auf den Namen der App geändert werden, der in der AndroidManifest.xml zu finden ist.

```
27    <application
28        android:label="Nusaplanner"
29        tools:replace="android:label"
30        android:icon="@mipmap/ic_launcher">
```

Abb. 3.21: Ändern des Android-App-Namens in der AndroidManifest.xml

Um die .APK-Datei zu erstellen, wird der Befehl in **Listing 3.14** ausgeführt.

```
flutter build apk
```

Listing 3.14: Befehlszeile zum Erstellen der .APK-Datei der App

4. Auswertung

In den folgenden Unterkapiteln wird das im vorherigen Kapitel erarbeitete Ergebnis der Implementierung der App analysiert[6]. Die App wurde in verschiedenen Szenarien getestet und die Erfüllung der Anforderungen wird bewertet.

4.1. Implementierung der App

Im Rahmen der App-Tests gibt es zwei wichtige Phasen. Nachdem der Backend-Service erfolgreich mit Laravel erstellt und implementiert wird, können die Funktionen der App nur noch mit dem lokalen Webserver (XAMPP) getestet werden, um die Daten aus der Datenbank abzufragen. In dieser Phase ist es wichtig, sicherzustellen, dass alle Anforderungen der App erfüllt sind und ihre Funktionen, einschließlich des Sendens und Empfangens von Daten vom Webserver, ordnungsgemäß funktionieren. Erst wenn alle Funktionen der App einwandfrei funktionieren und der Backend-Dienst erfolgreich auf dem Web-Host bereitgestellt wurde, muss der lokale Webserver während der Funktionstests der App nicht mehr verwendet werden. In der letzten Phase wird die App auch auf verschiedenen Android-Geräten getestet.

Unabhängig vom Webserver (lokal oder online) werden verschiedene Interaktionen mit der App analysiert, die von den folgenden Testfällen abgedeckt werden:

Testgruppe	**Testfall**	**Anforderungen**	**Erwartetes Ergebnis**	**Status**
Benutzer-authentifizierung und -autorisierung	Anmeldung	F1.2	Die persönlichen Daten des Benutzers (z.B. User ID, email, Token) werden lokal gespeichert. Der Benutzer wird auf die *Plan*-Seite weitergeleitet	Erfolgreich

[6] Der Quellcode ist online in [36] zu sehen.

	Registrierung	F1.1	Der Benutzer wird zu den *Onboarding*-Bildschirmen weitergeleitet	Erfolgreich
Datum der Studien-vorbereitung	Eingabe eines neuen Startdatums	F2.2	Das neu erstellte Enddatum wird in *Plan*-Page angezeigt	Erfolgreich
	Aktualisierung des bestehenden Startdatums	F4.8	Das neu erstellte Enddatum wird in *Plan*-Page angezeigt	Erfolgreich
Bottom Navigation Bar	Auf andere Seiten wechseln	F4.1	Der Benutzer kann zu verschiedenen Hauptseiten wechseln	Erfolgreich
Plan-Seite	Das Enddatum anzeigen	F5.1	Da das Enddatum lokal gespeichert wird, kann das Enddatum in *Plan*-Seite angezeigt werden	Erfolgreich
	Modifizieren der Werte von Checkboxen (Ankreuzen von Checkbox)	F5.2; F10	Der Wert des Kontrollkästchens ist entweder TRUE (angekreuzt) oder FALSE (nicht angekreuzt).	Erfolgreich
Feed-Seite	Anzeigen der Bildschirmdetails der einzelnen Vorbereitungs-schritte	F4.3; F4.4	Der Benutzer kann den Detailbildschirm des zugehörigen Vorbereitungsschritts öffnen	Erfolgreich

	Das Öffnen der PDF-Dateien mit dem PDF-Viewer	F4.5	Die PDF-Dateien können in der App geöffnet werden	Erfolgreich
	Das Öffnen der externen Links mit URL-launcher	F4.6	Die Links können entweder in der App oder im Webbrowser geöffnet werden	Erfolgreich
Inbox-Seite	Live-Chat-Dienst starten	F6	Der Benutzer wird zum neu eingerichteten Live-Chat Service weitergeleitet	Erfolgreich
Profile-Seite	Abmeldung	F8	Die persönlichen Daten des Benutzers werden gelöscht. Der Benutzer wird auf die *SignIn*-Seite umgeleitet.	Erfolgreich

Tabelle 4.1: Testfälle

4.2. Ergebnisse der Testdurchführung

Bei der Durchführung der Tests wurden die eigenen Smartphones verwendet. Im weiteren Verlauf werden die Ergebnisse des Tests dargestellt, die nach der Reihenfolge der jeweiligen Aufgaben kategorisiert sind. Zusätzliche Informationen und Anmerkungen oder Feedback zu bestimmten Funktionen und Optimierungen der App werden ebenfalls hinzugefügt.

Aufgabe 1 – Anmeldung und Registrierung

Da beide Funktionalitäten der App unkompliziert sind, können diese Aufgaben ordnungsgemäß ausgeführt werden.

Es gibt ein paar Hinweise, die zu beachten sind. Während des Anmeldevorgangs sollte der Benutzer die Möglichkeit haben, das versteckte Kennwort sehen zu können, damit er leichter überprüfen kann, ob das eingegebene Kennwort korrekt ist oder nicht. Dies gilt auch für die Anmeldeseite. Derzeit verfügt diese App nicht über diese Funktion.

Aufgabe 2 – Anweisungen des Onboarding-Screens befolgen

Die Anweisungen auf den *Onboarding*-Bildschirmen können erfolgreich befolgt werden, da die Bildschirme nur die schriftlichen Informationen über die Verwendung der App und eine Schaltfläche "*Next*" enthalten. Bevor der Benutzer zur Hauptseite weitergeleitet wird, kann das Startdatum der Studienvorbereitung hinzugefügt werden.

Aufgabe 3 – Ankreuzen irgendeines Schrittes der Studienvorbereitung

Nach erfolgreicher Eingabe des Startdatums der Studienvorbereitung wird das Enddatum automatisch auf der *Plan*-Seite angezeigt. Die Vorbereitungschecklisten werden durch die Abfolge der verschiedenen Schritte in Form einer erweiterbaren Liste dargestellt. Die Checklisten können abgehakt und gespeichert werden.

Ein kleines Problem, das behoben werden soll, ist dass die erweiterbare Liste nicht mehr aufgeklappt wird, nachdem der Benutzer eine der Checklisten eines zugehörigen Schritts abgehakt hat. Der Benutzer muss also immer noch die zuletzt abgehakte Checkliste öffnen, um zu sehen, ob die Checkliste bereits angekreuzt ist oder nicht. Die Liste sollte auch dann noch aufgeklappt werden, wenn der Benutzer gerade eine der Checklisten markiert hat.

Aufgabe 4 – Anzeigen der Detailbildschirme der verschiedenen Schritte

Diese Aufgabe kann problemlos durchgeführt werden. Da jeder Vorbereitungsschritt mehrere Detailbildschirme enthält, beginnt jeder Detailbildschirm immer mit der Übersicht über die Vorbereitung. Auf dem Übersichtsbildschirm kann der Benutzer die Reihenfolge der kleinen Schritte sehen. Die kleinen Schritte können durch Klicken auf die Schaltfläche "*Next*" oder "*Back*" angezeigt werden. Außerdem gibt es auf jedem kleinen Schrittbild eine zusätzliche Schaltfläche, mit der der Benutzer wieder zum Übersichtsbild zurückkehren kann.

Es gibt ein paar wichtige Punkte, die beachtet werden sollen. Der Benutzer kann nur vom Übersichtsbildschirm aus zur Hauptseite, d.h. zur *Feed*-Seite, zurückkehren. Auf jedem der kleinen Schritte muss es eine Schaltfläche zum Verlassen des Bildschirms geben, damit der Benutzer, wenn er zur Hauptseite zurückkehren möchte, diese Schaltfläche verwenden kann. Außerdem wird vorgeschlagen, dass jeder der kleinen Schritte auf dem Übersichtsbildschirm einen Link enthält, der den Benutzer zum gewünschten kleinen Schrittbildschirm weiterleiten kann. Auf diese Weise muss der Benutzer nicht ständig auf die Schaltfläche „*Next*“ klicken, um zu dem gewünschten kleinen Schritt zu gelangen.

Aufgabe 5 - Änderung des Enddatums der Studienvorbereitung

Die *updateDate*-Seite enthält denselben Inhalt wie die *addDate*-Seite, auf die der Benutzer nach der Registrierung stößt. Auf dieser Seite kann das Datum erfolgreich geändert werden und das aktualisierte Enddatum wird auf der *Plan*-Seite angezeigt.

Aufgabe 6 - PDF-Dateien und externe Weblinks anzeigen

Einige der kleinen Schrittbildschirme enthalten PDF-Dateien oder externe Links. Bei den PDF-Dateien kann es sich um ein wichtiges Dokument oder einfach um zusätzliche Informationen zu dem entsprechenden Vorbereitungsschritt handeln. Die externen Links können auch notwendige Informationen zur Studienvorbereitung enthalten. Es gibt keine Schwierigkeiten beim Öffnen der PDF-Dateien und der externen Links.

Ein wichtiger Hinweis ist jedoch, dass die App, nachdem der Benutzer auf den Link zur PDF-Datei oder den externen Link geklickt hat, die PDF-Datei oder den Link laden muss. Dieser Ladevorgang nimmt etwas Zeit in Anspruch. Es wird vorgeschlagen, einen Indikator (z.B. *Circular Progress Indicator*) zu implementieren, der anzeigt, dass die App noch mit dem Laden der PDF-Datei oder des Links beschäftigt ist.

Aufgabe 7 - Beratung über den Live-Chat-Service

Der Live-Chat-Service kann entweder über die *Inbox*-Seite oder die *Feed*-Seite im Bereich Tools aufgerufen werden. Der Dienst kann erfolgreich aufgerufen werden.

Es gibt einige wichtige Punkte, die für die weitere Entwicklung beachtet werden müssen. Wenn der Live-Chat gerade gestartet wird, braucht die App Zeit, um den Dienst zu laden. Daher ist die Implementierung eines Indikators notwendig, um zu zeigen, dass die App noch mit dem Laden des Dienstes beschäftigt ist. Es ist auch wichtig, eine Anleitung zu erstellen, wie man diesen Kundenservice richtig anwendet. Bis jetzt stellt die App nur den Dienst zur Verfügung, aber wichtige Details wie z.B. welche Fragen vom Chatbot beantwortet werden können und welche nur an den Kundendienst gestellt werden können, sind noch nicht klar.

Aufgabe 8 – Abmeldung

Die Abmelde-Schaltfläche befindet sich nur auf der Profilseite. Diese Aufgabe kann ohne Probleme ausgeführt werden.

Aus der Bewertung der obigen Testimplementierung wird geschlossen, dass die weitere Entwicklung und Verbesserung der Software noch notwendig sind. Die folgenden Punkte sind einige der wichtigen Funktionalitäten, die noch in der App implementiert werden müssen:

- In einigen Teilen der App muss einen Indikator wie beispielsweise *Circular Progress Indicator* implementiert werden, damit der Benutzer nicht mehrmals auf die entsprechende Schaltfläche oder den Link klickt.
- Für die ordnungsgemäße und korrekte Nutzung des Live-Chat-Dienstes muss eine Reihe von Verfahren geschaffen werden, die sich speziell an den Benutzer richten.

Der Prototyp der App muss zwar noch verbessert werden, was die Nutzung des Live-Chat-Dienstes während der Tests angeht, aber er ist noch nicht bereit für den echten Einsatz.

4.3. Erfüllung der Anforderungen

Die gesamten Funktionen der App funktionieren wie erwartet. Auch die Verbindung zwischen der App und dem Backend-Server wird erfolgreich hergestellt. Daher können die Funktionen der App, die eine Kommunikation mit dem Server erfordern (z.B. Anmelden, Registrierung, Abrufen der Checklistenwerte usw.), fehlerfrei ausgeführt werden. Außerdem kann der externe Dienst wie der Live Chat-Dienst erfolgreich genutzt werden. Während des Tests wurden nur geringfügige Probleme mit der Benutzerfreundlichkeit festgestellt. In Tabelle 4.2 und 4.3 sind alle realisierbaren Optionen zur Erfüllung der Spezifikationen aufgeführt.

Funktionale Anforderungen	**Passende Lösungsansätze**
F1.1; F1.2	Die Dio-Bibliothek wird für die JSON-Web-Token-Autorisierung und das Parsen der JSON-Antwort verwendet, die die persönlichen Daten des Benutzers und das Zugriffstoken enthält. Die persönlichen Daten werden mit dem fluttersecurestorage-Plugin lokal gespeichert.
F2.1	Mit der StatelessWidget-Klasse von Flutter werden vier verschiedene Einführungsbildschirme erstellt. Jeder der Einführungsbildschirme enthält Informationen darüber, wie die Hauptseite der App zu verwenden ist.
F2.2; F4.8; F10	Ein Widget namens "DatePickerWidget" wird erstellt, um das Enddatum der Studienvorbereitung des Benutzers zu schätzen und speichern, nachdem das Startdatum eingegeben wurde. Um das Datum lokal zu speichern, wird das Plugin "SharedPreferences" verwendet.
F3	Vier verschiedene Hauptseiten werden mit der Flutter StatefulWidget-Klasse erstellt.
F4.1	Das BottomNavigationBar-Widget wurde implementiert, um eine einfache Navigation zwischen den Hauptseiten der App zu ermöglichen.
F4.2	Das Plugin Carousel_slider wird verwendet.

F4.3; F4.4	Die Details der einzelnen Vorbereitungsschritte werden mit der StatelessWidget-Klasse erstellt, da diese Bildschirme keinen veränderbaren Zustand benötigen.
F4.5	Das Plugin Flutter_pdfview wird zum Anzeigen von PDF-Dateien verwendet.
F4.6	Das Plugin url_launcher wird eingesetzt, um die externen Links entweder in der App selbst oder im Browser zu öffnen.
F4.7	Das ListView-Widget wurde implementiert, um die Liste der Tools zu erstellen, die horizontal gescrollt werden kann.
F4.9; F6	Das Kommunicate-Plugin wird als Live Chat-Dienst verwendet.
F5.1	Nachdem das Datum mit dem SharedPreferences- Plugin lokal gespeichert wurde, kann das Datum mit dem Text-Widget dargestellt werden.
F5.2	Dio-Bibliothek und API-Funktionen namens "fetchTodolist" sowie "updateTodolist" sind implementiert. Die fetchTodolist-Methode, die die GET-Methode enthält, wird zum Abruf der Checklistenwerte vom Backend verwendet. updateTodolist-Methode, die die POST-Methode enthält, wird zur Übertragung der aktualisierten Werte an das Backend verwendet.
F8	Die signOut-Methode wird eingesetzt, um alle lokal gespeicherten persönlichen Informationen zu löschen.
F9	Das Provider-Paket wird für die einfache Verwaltung des App-Status verwendet.

Tabelle 4.2: Erfüllung der funktionalen Anforderungen

Nicht-Funktionale Anforderungen	Passende Lösungsansätze
NF1	Im Widget Text ist die Schriftgröße-Eigenschaft auf 14 als Mindestschriftgröße eingestellt. Der Zeilenabstand zwischen den Textzeilen ist ebenfalls konfiguriert.
NF2	Das Design ist so einfach wie möglich konzipiert und enthält nur die notwendigen Widgets auf den zugehörigen Seiten.
NF3	Lange vor der Entwicklung der App wurde eine Recherche zur Studienvorbereitung von Indonesiern in Deutschland durchgeführt, um die notwendigen Informationen zu sammeln.
NF4	Der Vorbereitungsfortschritt des Benutzers wird durch eine Reihe von Checklisten dargestellt.
NF5	Die App bietet detaillierte und gründliche Erklärungen zu jedem komplexen Vorbereitungsschritt, die auf der Feed-Seite zu finden sind. Außerdem steht ein Live-Chat-Service zur Verfügung, um Unklarheiten bei der Studienvorbereitung zu beseitigen.
NF6	Die App wurde mit dem Flutter SDK implementiert.

Tabelle 4.3: Erfüllung der nicht-funktionalen Anforderungen

5. Fazit

Im Rahmen dieser Arbeit wurde eine kostengünstige und praktische Android-Mobilanwendung entwickelt, die eine unkomplizierte, aktuelle, und gründliche Anleitung zur schrittweisen Studienvorbereitung enthält. Die App fungiert als Planer, der den Studierenden hilft, den Überblick über wichtige Termine und Ereignisse der Studienvorbereitung zu behalten, wie z.B. die Einschreibung für einen Sprachkurs, die Beglaubigung der erforderlichen Dokumente usw. Der Planer gibt auch Auskunft über die übliche Zeiteinschätzung der Studienvorbereitung. Darüber hinaus bietet die App ausführliche Erklärungen zu den verschiedenen Schritten der Studienvorbereitung und fasst den komplexen Prozess in einem leicht verständlichen Leitfaden zusammen. Die App bietet auch eine On-Demand-Beratung in Form eines Live-Chat-Dienstes, um Studienbewerbern bei der Überwindung einiger Hindernisse bei der Vorbereitung zu helfen.

Die App wird mit dem Flutter SDK entwickelt. Ursprünglich sollte die App für beide mobilen Betriebssysteme, also Android und iOS, und für das Web entwickelt werden. Für die Zwecke dieser Arbeit wird jedoch nur die Android-Version der App entwickelt. Die iOS- und Webversionen werden in der Zukunft entwickelt. Der Grund für die Implementierung von Flutter SDK ist eine effiziente Entwicklung einer App mit nur einer einzigen Codebasis. Die App ist in der Programmiersprache Dart geschrieben. Für das Backend der App wird das Web-Framework Laravel eingesetzt.

Der Schwerpunkt bei der Entwicklung dieser App liegt auf der Verbesserung der Datenerfassung für die Studienvorbereitung in Deutschland und dem Einsatz der App als Plattform, die verschiedene Informationen aus unterschiedlichen, voneinander getrennten Quellen integriert. Die aus verschiedenen Quellen gesammelten Informationen werden in Form einer Abfolge von Detailbildschirmen auf einer der Hauptseiten der App, der *Feed*-Seite, umfassend visualisiert. Ein Detailbildschirm enthält die zusammengefassten Informationen eines Studienvorbereitungsprozesses und kann die Übersicht über wichtige Formulare oder Dokumente in Form von PDF-Dateien sowie zusätzliche Informationen aus externen Weblinks enthalten. Der Benutzer kann diese PDF-Dateien mit dem PDF-Viewer öffnen, der mit dem Plugin flutter_pdfview implementiert ist. Außerdem können die externen Links, die die zusätzlichen Informationen enthalten, entweder über den Webviewer in der App oder über den Webbrowser geöffnet werden.

Um das voraussichtliche Enddatum der Studienvorbereitung zu sehen, muss der Benutzer zuvor das Startdatum eingeben, das mithilfe des SharedPreferences-Plugins lokal und in der MySQL-Datenbank mit Hilfe des Laravel-Frameworks gespeichert wird. Die persönlichen Daten des Benutzers, wie Benutzer-ID, E-Mail, Zugriffstoken (JSON Web Token oder JWT) werden als sensible Daten eingestuft und daher mit dem fluttersecurestorage-Plugin lokal gespeichert. Das lokal gespeicherte Enddatum kann dann auf der Plan-Seite der App angezeigt werden. Um den Live-Chat-Dienst mit dem Kommunicate-Plugin zu starten, wird die lokal gespeicherte Benutzer-ID zur Unterscheidung der Benutzer verwendet.

Nach dem Test der App wird festgestellt, dass es einige Aspekte gibt, die noch verbessert werden müssen. Es wird dringend empfohlen, eine Anleitung zur Nutzung des Live-Chat-Dienstes speziell für den Benutzer zu erstellen, damit dieser den Dienst ordnungsgemäß und korrekt nutzen kann. Darüber hinaus sollte in einigen Bereichen der App ein Indikator implementiert werden, insbesondere an den Stellen, an denen die App mit dem Laden von PDF-Dateien oder externen Links sowie dem Starten des Live-Chat-Dienstes beschäftigt ist. Schließlich zeigt dieses Projekt das Potenzial der Entwicklung mobiler Apps mit Flutter SDK dank seiner vollständigen Dokumentation, die gründliche und detaillierte Informationen über die Verwendung von Widgets mit einfachen Beispielen bietet. Die weitere Entwicklung ist für die iOS- und Web-Version der App vorgesehen.

Literaturverzeichnis

[1] ICEF GmbH, „Recruiting from Indonesia in a context of increased competition," 18. September 2019. [Online]. Available: https://monitor.icef.com/2019/09/recruiting-from-indonesia-in-a-context-of-increased-competition/. [Zugriff am 02. Juni 2021].

[2] Expatrio Global Services GmbH, „Cooperation between Indonesia and Germany in the areas of higher education and skilled labor," 17. Dezember 2019. [Online]. Available: https://www.expatrio.com/blog/2019-12/cooperation-between-indonesia-and-germany-areas-higher-education-and-skilled-labor. [Zugriff am 02. Juni 2021].

[3] A. Scheidmeir, „Mobile App-Entwicklung mit Google Flutter," exensio GmbH, 20 August 2020. [Online]. Available: https://exensio.de/news-medien/newsreader-blog/mobile-app-entwicklung-mit-google-flutter. [Zugriff am 15 September 2021].

[4] Interaction Design Foundation, „Native vs Hybrid vs Responsive: What app flavour is best for you?," 2020. [Online]. Available: https://www.interaction-design.org/literature/article/native-vs-hybrid-vs-responsive-what-app-flavour-is-best-for-you. [Zugriff am 15 September 2021].

[5] infotechalive, „Advantages and Disadvantages of Native apps and Hybrid apps," 04. Mai 2018. [Online]. Available: https://infotechalive.com/advantages-and-disadvantages-of-native-apps-and-hybrid-apps/. [Zugriff am 17 September 2021].

[6] YML, „Native VS Hybrid Mobile Apps — Here's How To Choose," 05. Juli 2021. [Online]. Available: https://yml.co/native-vs-hybrid-mobile-apps-heres-how-to-choose. [Zugriff am 16 September 2021].

[7] T. H., „Pros and Cons of Mobile Websites and Mobile Apps," 01. September 2019. [Online]. Available: https://rubygarage.org/blog/mobile-app-vs-mobile-website. [Zugriff am 19 September 2021].

[8] S. Kidecha, „Native vs. Hybrid vs. Cross-Platform: How and What to Choose?," 27. Mai 2020. [Online]. Available: https://dzone.com/articles/native-vs-hybrid-vs-cross-platform-how-and-what-to. [Zugriff am 19 September 2021].

[9] A. Frisina, „The Pros and Cons of Native vs Hybrid App Development,“ 12 Juni 2021. [Online]. Available: https://sunlightmedia.org/native-vs-hybrid/. [Zugriff am 19 September 2021].

[10] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak und G. Ghinea, „An empirical investigation of performance overhead,“ 09. Juni 2020. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s10664-020-09827-6.pdf. [Zugriff am 25 September 2021].

[11] M. L. Napoli, Beginning Flutter, Z. Chawdhary, Hrsg., Indianapolis, Indiana: John Wiley & Sons, 2019, pp. xxi, 5.

[12] T. Sneath, „Flutter 1.0: Google’s Portable UI Toolkit,“ 04. Dezember 2018. [Online]. Available: https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html. [Zugriff am 25 September 2021].

[13] K. Kyslova und P. Horvath, „Is Google’s Flutter SDK a good choice for your app?,“ 30. Oktober 2020. [Online]. Available: https://proxify.io/articles/what-is-flutter. [Zugriff am 25 September 2021].

[14] P. Schivanchal, „Declarative and Imperative Programming in the flutter,“ 12 Mai 2021. [Online]. Available: https://medium.flutterdevs.com/declarative-and-imperative-programming-in-the-flutter-2d4c6b704a87. [Zugriff am 25 September 2021].

[15] F. Pereiro, „Declarative Programming: Is It A Real Thing?,“ 04. Mai 2016. [Online]. Available: https://www.toptal.com/software/declarative-programming. [Zugriff am 25 September 2021].

[16] Flutter, „Introduction to declarative UI,“ 2021. [Online]. Available: https://docs.flutter.dev/get-started/flutter-for/declarative. [Zugriff am Oktober 2021].

[17] Flutter, „Flutter architectural overview,“ 2021. [Online]. Available: https://docs.flutter.dev/resources/architectural-overview. [Zugriff am 23. September 2021].

[18] Surf, „Our Guide to Flutter Architecture,“ 2021. [Online]. Available: https://surf.dev/flutter-architecture-guide/. [Zugriff am 25. September 2021].

[19] Alibaba Clouder, „Exploration of the Flutter Rendering Mechanism from Architecture to Source Code,“ 08. Februar 2021. [Online]. Available: https://www.alibabacloud.com/blog/exploration-of-the-flutter-rendering-mechanism-from-architecture-to-source-code_597285. [Zugriff am 26. September 2021].

[20] E. Windmill, Flutter in Action, Shelter Island, New York: Manning Publications Co., 2020, pp. 61-62, 84.

[21] T. Ho, „Flutter: Element and RenderObject,“ 19. April 2021. [Online]. Available: https://itzone.com.vn/en/article/flutter-element-and-renderobject/. [Zugriff am 28. September 2021].

[22] R. Payne, Beginning App Development with Flutter: Create Cross-Platform Mobile Apps, Dallas, Texas: Apress Media LLC, 2019, p. 190.

[23] M. Stauffer, Laravel: Up & Running, Bd. II, Sebastopol, California: O'Reilly Media, Inc., 2019, pp. 3, 24, 117, 145.

[24] A. Burets, „Laravel vs Symfony in 2021 – Which PHP Framework Choose For Your Project?,“ 06. Januar 2021. [Online]. Available: https://scand.com/company/blog/laravel-vs-symfony-in-2021-which-php-framework-choose-for-your-project/. [Zugriff am 14 Oktober 2021].

[25] A. Kılıçdağı und H. İ. YILMAZ, Laravel Design Patterns and Best Practices, Birmingham: Packt Publishing Ltd, 2014, pp. 14-15.

[26] w3schools, „Controllers in Laravel,“ [Online]. Available: https://www.w3schools.in/laravel-tutorial/controllers/. [Zugriff am 31. Dezember 2021].

[27] E. Heidi, „A Practical Introduction to Laravel Eloquent ORM,“ [Online]. Available: https://www.digitalocean.com/community/tutorial_series/a-practical-introduction-to-laravel-eloquent-orm. [Zugriff am 14 Oktober 2021].

[28] Laravel, „Eloquent: Relationships,“ [Online]. Available: https://laravel.com/docs/8.x/eloquent-relationships#one-to-many. [Zugriff am 14. Oktober 2021].

[29] M. Summers, „Everything You Need to Know About Mobile App Development Architecture,“ 24. September 2018. [Online]. Available: https://magora-systems.com/mobile-app-development-architecture/. [Zugriff am 24 Oktober 2021].

[30] Lucidchart, „What is Business Process Modeling Notation,“ [Online]. Available: https://www.lucidchart.com/pages/bpmn. [Zugriff am 15. Dezember 2021].

[31] pub.dev, „dio 4.0.4,“ 22. November 2021. [Online]. Available: https://pub.dev/packages/dio. [Zugriff am 20. Dezember 2021].

[32] A. Kattel, „DIO IN FLUTTER,“ 08. Januar 2020. [Online]. Available: https://medium.com/@ashmikattel/dio-in-flutter-ad6ba26aee36. [Zugriff am 20. Dezember 2021].

[33] Auth0, „Introduction to JSON Web Tokens,“ [Online]. Available: https://jwt.io/introduction. [Zugriff am 22 Dezember 2021].

[34] SmartBear Software, „Bearer Authentication,“ [Online]. Available: https://swagger.io/docs/specification/authentication/bearer-authentication/. [Zugriff am 21. Dezember 2021].

[35] Flutter, „Store key-value data on disk,“ [Online]. Available: https://docs.flutter.dev/cookbook/persistence/key-value. [Zugriff am 23. Dezember 2021].

[36] A. G. Waluya, „Nusaplanner Mobile App,“ 15. September 2021. [Online]. Available: https://github.com/adibwaluya/NusaplannerMobileApp.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass

- Ich die vorliegende wissenschaftliche Arbeit selbständig und ohne erlaubte Hilfe angefertigt habe,
- Ich andere als die angegebenen Quellen und Hilfsmittel nicht benutzt habe,
- Ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe,
- Die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen hat.

Berlin, 01.07.2021

Adib Ghassani Waluya