

Entwicklung eines Prototyps einer Applikation mit flexiblen Designelementen

Maturaarbeit

Produkt mit Werkbericht



17.10.2021

Abstract

In Rahmen dieser Arbeit wurde eine interaktive Mobile Appvorlage einer «E-Commerce» App für Smartphones in «Flutter», ein UI-Entwicklungs-Kit von Google, erstellt, welches die «Dart» Programmiersprache für die Entwicklung plattformübergreifender Apps verwendet.

Die Appvorlage soll die elementaren Bausteine und Funktionen einer E-Commerce App wie Startseite und Produktinformationsseite beinhalten. Diese soll anschließend als Vorlage für die Erstellung verschiedener E-Commerce Anwendungen dienen.

Am Anfang wurde eine Entwicklungsumgebung für «Flutter» errichtet. Zunächst wurde zur Vorbereitung und Übung eine einfache Taschenlampen App mit bereits im Internet vorhandenem Code sowie mithilfe eines YouTube Tutorial entwickelt. Diese wurde für Tester auf «Google Play Store» hochgeladen und freigegeben. Als nächster Schritt wurde ein «Wireframe» (Vorführmodell) für die App in «Adobe XD» (Adobe Experience Design) erstellt und in ein interaktives «Mockup» in «Framer», einer «No-Code-Plattform», «All-In-One Design Tool» (Werkzeug) umgewandelt. Schließlich wurde die Appvorlage in «Flutter» entwickelt.

Mit einfachen Änderungen können verschiedene Designelemente wie Farbe, Typografie und Form angepasst wie auch gestaltet werden. Durch Wiederverwendbarkeit der Komponenten wird der Appvorlage ein flexibles, simples und nahezu grenzenloses Design verliehen. Die Appvorlage wurde schlussendlich zwar nicht vollständig entwickelt, aber kann als Vorlage für die Erstellung vieler verschiedener Elemente einer E-Commerce Applikation dienen. Mit ihrer Hilfe wird der Übergang von einer Idee oder einem Produkt zu einer App vereinfacht und die Entwicklungszeit gekürzt.

Vor Beginn dieser Arbeit hatte ich weder im Programmieren noch in Bereichen der Mobile App-Entwicklung oder des Mobile App Designs wirklich Ahnung.

Die Appvorlage konnte im Rahmen der Arbeit nicht vollständig entwickelt werden, aber kann in ihrer Fassung als Vorlage für mehrere Elemente einer E-Commerce App verwendet werden. Eine vollständige Version hat grosses Potenzial.

Vorwort

Meine Leidenschaft für Informatik habe ich erst vor ein paar Jahren entdeckt, aber seitdem wächst sie im rasanten Tempo und mit enormer Geschwindigkeit an. Um mein Wissen zu erweitern, habe ich an mehreren Informatik Projekten vom Gymnasium Thun teilgenommen. Angefangen bei der Projektwoche «Math and Games» in der ich erste Erfahrungen in Python sammeln konnte, bis hin zur Ergänzungsfachwoche im Ergänzungsfach Informatik «Künstliche Intelligenz und Robotik», in welcher ich im Team mit einem Lego-Bausatz namens «Mindstorms NXT» einen Roboter programmiert und konstruiert habe.

Mir war von Anfang klar, dass ich für meine Maturaarbeit etwas entwickeln möchte, was mit Informatik zu tun hat und auch viele Menschen unterstützt. Zuerst kam mir die Idee, eine Website zu programmieren, welche ich aber wieder verwarf, da ich mich mehr für die Entwicklung einer Mobile App interessierte. Um ein solches Vorhaben ohne jegliche Vorkenntnisse im Programmieren und in der Softwareentwicklung zu verwirklichen, würden viele Personen sich wohl eine simple Applikation als Ziel vornehmen. Ich habe mich allerdings dazu entschieden eine für Anfänger eher komplizierte zu entwickelnde App zu kreieren.

Die Absicht hinter meiner Maturaarbeit war in erster Linie das Programmieren zu erlernen und etwas Nützliches selbst zu erschaffen. Ein weiteres Ziel war einen UI Design (User Interfacedesign: Schnittstellengestaltung) Entwurf mit anpassbaren Elementen wie Knopffarbe – oder rahmen zu erstellen.

Ich habe aus den vielen Fehlermeldungen beim Programmieren und den immer wiederkehrenden Komplikationen eine Lektion gelernt. Nämlich sich nicht zu sehr an den auftretenden Problemen, die nicht lösbar erscheinen, aufzuregen und sich dadurch unter Druck zu setzen, da man es mit der Zeit lösen kann.

Ich möchte mich bei meinem Betreuer, Herrn A. W., und meinem Ergänzungsfach Informatiklehrer, Herrn R. B., bedanken. Ein spezieller Dank geht an meine Familie für ihre Geduld und an meine Kollegen, insbesondere an L. S., für ihre Unterstützung.

Inhaltsverzeichnis

1	Einleitung.....	5
2	Theoretische Grundlagen	5
2.1	Software	5
2.2	Arten von Softwares	6
2.2.1	Systemsoftware	6
2.2.2	Anwendungssoftware	6
2.3	Mobile App.....	6
2.3.1	Konzeptphase und Analyse	7
2.3.2	Designprozess	8
2.3.3	Programmierung und Entwicklung.....	9
2.3.4	Testen und Veröffentlichung.....	9
2.3.5	Instanthalung	9
2.4	Arten von Apps und Technologien	9
2.4.1	Native Apps	9
2.4.2	Web Apps.....	10
2.4.3	Hybride Apps.....	10
2.4.4	Cross-Plattform Apps.....	11
2.5	UX Design und UI Design	11
2.5.1	UX Design	11
2.5.2	UI Design	11
2.6	Cross-Plattform-Entwicklung	13
2.7	Dart	14
2.8	Flutter	14
2.8.1	Everything is a Widget.....	15
2.8.2	Stateless und Stateful Widgets.....	15
3	Methode.....	18
3.1	Material und Vorbereitung.....	18
3.2	Flutter Installation.....	20
3.3	Erstellung und Aufbau eines Flutter-Projekts	21
3.4	Taschenlampen App	23

3.4.1	Code Erläuterung.....	25
3.5	Taschenlampen App Version Zwei	31
3.6	Design Mobile Apps	35
3.7	E-Commerce Appvorlage	36
3.7.1	E-Commerce Appvorlage Elemente und Funktionen	36
3.7.2	E-Commerce Appvorlage Design	38
3.7.3	E-Commerce Appvorlage Struktur und Aufbau	40
3.7.4	Anpassbare Komponentenvorlagen	45
3.7.5	Erstellung eigener Widgets	49
4	Ergebnisse.....	53
4.1	Taschenlampen Apps.....	53
4.2	Appvorlage-Prototyp.....	55
4.2.1	Wireframe.....	55
4.2.2	Mockup.....	55
4.2.3	Prototyp	58
5	Fazit.....	61
6	Quellenverzeichnis	65
6.1	Internetquellenverzeichnis.....	65
6.1.1	Fussnoten.....	68
6.2	Abbildungsverzeichnis	69
6.3	Tabellenverzeichnis	71
7	Eidesstattliche Erklärung	72

1 Einleitung

Das Leben ohne ein Smartphone ist heutzutage für viele undenkbar. Im Jahr 2021 schätzt man die Anzahl aller Smartphone-Nutzer auf etwa 3.9 Milliarden, das sind rund 500 Millionen mehr als im Jahr 2019. Bis 2024 sollen es 4.5 Milliarden werden. Neben der Hardware, die physisch vorhandenen Bestandteile eines Smartphones, ist die Software, Programme oder Daten, der Hauptbestandteil. [1]

Die Entwicklung einer Applikation ist meistens nicht die Aufgabe einer einzigen Person. Besonders wenn man ein Anfänger ist, was die Software-Entwicklung betrifft. In einem kleinen Unternehmen sind sicherlich mindestens fünf Personen an der Entwicklung einer Software beteiligt. Vor Beginn dieser Arbeit hatte ich nahezu keine Vorkenntnisse im Programmieren sowie im Bereich der Mobile App-Entwicklung oder des Mobile App Designs. Da das Thema und die Umsetzung relativ kompliziert sein kann, wurde diese Arbeit so geschrieben, dass es für die Leser/innen möglich ist, selbst eine App zu erstellen.

In dieser Arbeit beschäftige ich mich mit Softwaredesign und Mobile App Entwicklung. Das Ziel der Arbeit ist eine Appvorlage für eine Mobile E-Commerce App mit einem flexiblen Design zu entwickeln. Die Anwendungssoftware soll die elementaren Bausteine und Funktionen einer E-Commerce App beinhalten. Das Design soll flexibel geplant und gestaltet werden, sodass die einzelnen Elemente der App, wie die Farbe oder der Rahmen der Knöpfe, mit einem einfachen Verändern von Parametern, Eigenschaften wie die Farbe oder der Kantenradius, zu einer umfangreichen und sichtbaren Änderung des ganzen Designs führen.

Die Appvorlage soll mit ihrem flexiblen Design dazu beitragen, den Übergang zwischen einer Idee oder einem Produkt und ihrer Verwirklichung in einer Mobile App zu vereinfachen und die Entwicklungszeit kürzen.

Die ganze Entwicklung wurde wegen der, für die «iOS» Plattform erforderlichen Entwicklungsumgebung und dem Fehlen iOS und «macOS» Geräte sowie wegen der hohen Gebühr im App Store ausschliesslich auf Android Geräten getestet. Jedoch kann der Quellcode im «Flutter» mit kleinen Änderungen für die iOS Plattform vorbereitet und entsprechend getestet werden. Die Appvorlage und das Design wurden nicht in Umfragen bewertet oder von anderen Personen getestet oder beurteilt. Sie wurde allein mit dem Wissen aus dem Internet, Vergleich mit anderen Apps und aus meinen einigen Ideen und Interpretationen und Überlegungen geplant, gestaltet und entwickelt.

2 Theoretische Grundlagen

2.1 Software

Software (engl. soft: weich, weiche Ware) ist ein Sammelbegriff für ausführbare Programme und den ihnen dazugehörigen Daten. Sie umfasst im Gegensatz zur Hardware (engl. hard: hart, fest), welche als Oberbegriff für die physischen Komponenten (elektronischen und mechanischen Bestandteil) steht, alle nicht physischen Komponenten eines computergestützten Systems. Eine Systemsoftware wie Betriebssysteme, z.B. «Android», «Windows» oder «iOS», ist genauso eine Software wie die darauf installierten Programme, also eine Anwendungssoftware wie ein Kalender- oder ein Spielprogramm. Die Software bestimmt, was ein

softwaregesteuertes Gerät tut und wie es das macht, daher wird sie oft mit einem Manuskript verglichen. In der Wirtschaftsinformatik wird oft zwischen Systemsoftware und Anwendungssoftware unterschieden. [2]

2.2 Arten von Softwares

2.2.1 Systemsoftware

Eine Systemsoftware, auch Systemprogramm genannt, ist die zentrale Software von Systemen, die sämtliche Abläufe beim Betreiben eines Rechners steuern. Sie legt den Grundstein für alle anderen Softwarearten, funktioniert als Schnittstelle zwischen Anwendungsprogrammen, was bedeutet, dass sie die Verbindung zur Hardware herstellt und die Verwendung dieser Ressourcen steuert. Sie wird auf das jeweilige Gerät vorinstalliert und mitgeliefert. In der Regel findet keine Interaktion zwischen der Systemsoftware und dem Benutzer statt. Zur Systemsoftware gehören Betriebssysteme wie «Linux», «macOS», «Android» und «Windows» sowie systemnahe Softwares. [4]

2.2.2 Anwendungssoftware

Eine Anwendungssoftware (engl. application software, app) ist ein Softwareprogramm oder Computerprogramm, also ein ausführbares Programm, welches für die Lösung von einem Benutzerproblem eingesetzt wird. Sie können eine oder mehrere Funktionen erfüllen. Anwendungssoftwares sind diejenigen, mit denen die Nutzer in Kontakt kommen und interagieren. Sie werden zum Teil als sogenannte Bundles vom Hersteller mitgeliefert oder können separat heruntergeladen und installiert werden. Beispiele für Anwendungssoftware sind Spielprogramme und Fotobearbeitungsprogramme. [5]

2.3 Mobile App

Eine Mobile App, Kurzform App, ist eine Anwendungssoftware für mobile Betriebssysteme und Mobilegeräte wie einem Smartphone oder Tablet, welche keine systemtechnische Funktionalität hat, damit ist gemeint, dass sie nicht relevant für das Funktionieren eines Systems selbst ist.

Mobile Apps werden in vier Typen aufgeteilt: Native-, Web-, Hybrid- und Cross-Plattform Apps. Native Apps sind plattformabhängig, funktionieren dementsprechend nur auf einer bestimmten Plattform wie Android oder iOS. Web-, Hybrid- und Cross-Plattform hingegen sind plattformunabhängig. [6]

Unter Mobile Entwicklung, auch App-Entwicklung genannt, versteht man der Prozess der Erstellung einer Mobile App für mobile Geräte wie Smartphones oder Tablets. Sie wird in verschiedenen Phasen unterteilt.

- Konzeptphase -und Analyse
 - App Idee

- Was soll die App können
 - Zielgruppe
 - App Typ und Funktionen
 - Ziel Plattform
- Design (Planung und Entwicklung)
 - Aufbau
 - Wireframe
 - Mockup
 - Prototypen
- Programmierung
 - Entwicklungstools
 - Framework
 - Programmiersprache
 - Programmieren
- Testen
 - Testen
 - Fehlerbeheben
- Veröffentlichung
 - Veröffentlichen
 - Marketing
- Instandhaltung
 - Aktualisieren
 - Erweitern und Anpassung
 - Dienstleistungen (wie Support)

[7]

2.3.1 Konzeptphase und Analyse

Apps und Produkte im Allgemeinen beginnen mit einer Idee oder als mögliche Lösung eines bestehenden Problems. Bei der Konzeptphase wird, wie der Begriff schon darauf hindeutet, ein Konzept erstellt. Dabei werden verschiedene Punkte berücksichtigt, Fragen beantwortet und Ziele definiert. Was soll die App können? Wer ist das Zielpublikum? Welche Möglichkeiten gibt es für die Umsetzung der Idee? Welche Chancen wird oder kann die App im Markt haben? Welche sind die Zielplattformen? Als nächster Schritt geht es zum Design. Da wird versucht, das Konzept in einer visuellen Form umzuwandeln. Dabei geht es um die Informationsarchitektur, die Navigation auf der grafischen Oberfläche sowie deren Gestaltung und ausserdem auch das Benutzererlebnis. Typische Fragen beim Designprozess sind:

Wie soll die App aussehen? Wie können die Funktionen in einer effizienten Weise genutzt werden? Wie wird durch die App navigiert? Wie sollen die einzelnen Elemente wie z.B. Text, Bilder oder Knöpfe gestaltet werden?

2.3.2 Designprozess

Als erstes wird ein sogenanntes Wireframe (engl. wire frame model, Drahtgittermodell) erstellt. Ein Wireframe ist ein schematischer, visueller Entwurf, welcher das strukturelle Modell einer grafischen Benutzeroberfläche einer Software definiert und das Grundgerüst einer App oder Webseite darstellt. Die Elemente wie z.B. Knöpfe, Text und Bilder werden in Form von einfachen Kästchen, Linien und Skizzen dargestellt. Diese können von Hand oder digital erstellt werden. Aus einem Wireframe wird dann ein «Mockup» (Vorführmodell, Attrappe, Anschauungsmodell) aufgebaut und weiterentwickelt. Mockups sind digitale Entwürfe einer App oder Webseite, die zu Präsentationszwecken eingesetzt werden. Anders als bei Wireframes sind Farben, Typographien, Bilder sowie Grafiken in ein Mockup integriert. Mockups können sowohl statisch als auch dynamisch, also interaktiv, sein. Aus den Mockups werden dann Prototypen erstellt, welche interaktiv sind und die meisten Funktionen der App beinhalten. [7, 8, 9, 10]

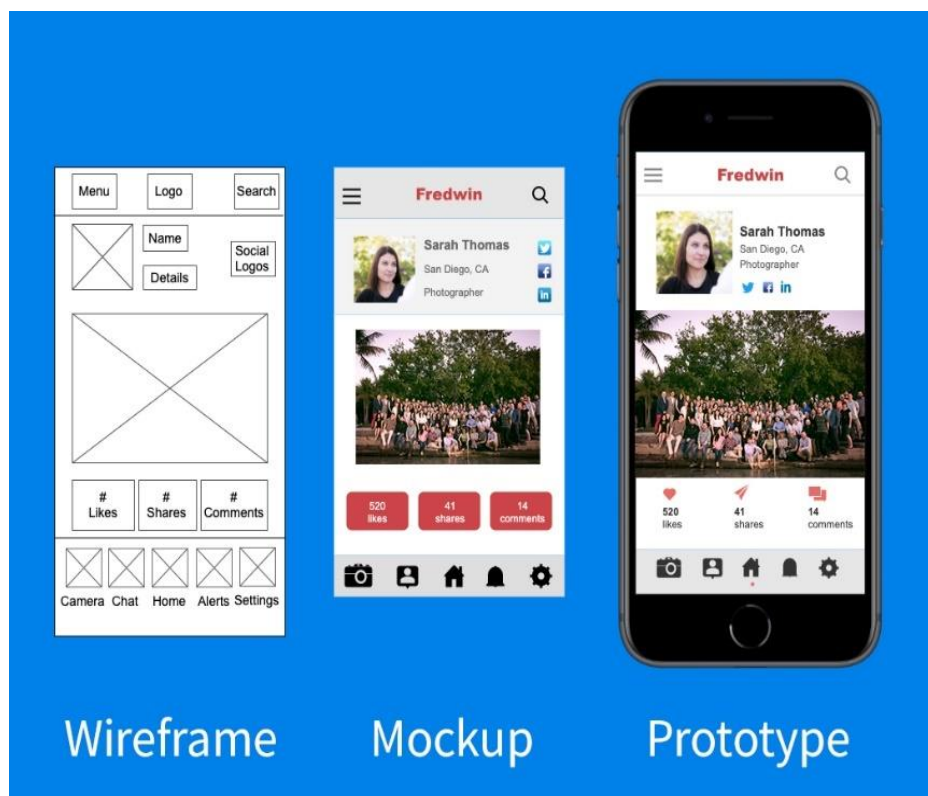


Abb. 1: Designprozess einer App: Wireframe, Mockup und Prototyp
Quelle: Aha (o. D.)

2.3.3 Programmierung und Entwicklung

Wenn das Design der App erstellt ist, kann mit der Programmierung begonnen werden. Die passenden Entwicklungstools, Frameworks¹ sowie Programmiersprachen werden ausgewählt. Während dem Programmierungsprozess können mehrere Prototypen entstehen, die immer verbessert und weiterentwickelt werden. Aus diesen entstehen dann die fast fertiggestellte App, die danach in die Testung-Phase kommt.

2.3.4 Testen und Veröffentlichung

Die App wird als erstes in einem kleinen Rahmen z.B. auf Simulation Geräten und/oder von einer kleinen Zahl an Nutzer getestet. Die Fehler werden behebt und die App wird laufend für mehr Nutzer fürs Testen freigegeben. Bei diesem Prozess entstehen verschiedene unfertige Versionen, die als Entwicklungsstadien bekannt sind, dazu gehören z.B. Alpha- oder Beta-Versionen. Die abschliessende Testversion der App Software, in welche alle Funktionen enthalten sind und alle von Testnutzer gemeldeten sowie bekannten Fehler behoben sind, wird als «Release Candidate», «Freigabekandidat» oder «Prerelease» (Vorabveröffentlichung) freigegeben. Aus dem Release Candidate wird danach die endgültige Version namens «Release» erstellt und abschliessende Kontrollen sowie Tests durchgeführt. Sie wird dann auf die geplanten Plattformen für alle Nutzer veröffentlicht. [11, 12, 13]

Viele Unternehmer beginnen schon vor dem «Release» mit dem Marketing und der Werbung ihrer Software. Dies geschieht aber auch nach der Veröffentlichung. Das Ziel der Werbung ist ein breites Spektrum potenzieller Nutzer anzulocken und somit viele Kunden gewinnen.

2.3.5 Instandhaltung

Eine Software wie eine App muss gepflegt und stetig aktualisiert werden. Sie muss fortlaufend an die sich entwickelnden Hardware und den neuen Technologien angepasst werden. Dazu muss für die Benutzer der App ein «user support/ tech support» service (Dienstleistung) eingerichtet werden, welche ihren Fragen und Anliegen nachgeht und bei der Lösung von Problem unterstützt. Dies gehört zu der Instandhaltung einer Software. [13]

2.4 Arten von Apps und Technologien

2.4.1 Native Apps

Native Apps sind in der Entwicklungssprache eines Betriebssystems programmiert. Sie können sich an die Funktionalitäten des entsprechenden Geräts anpassen, haben den Zugang zu Hardwareeigenschaften wie das «GPS»² oder die Kamera und können ausserdem im Offline-Modus verwendet werden. Native Apps haben die beste Leistung unter der App-Typen,

¹ Siehe Abschnitt Cross-Plattform-Entwicklung, Seite 13.

² Global Positioning System (GPS) Englisch für weltumspannendes Ortungssystem. (Wikipedia, 2021)

sind am sichersten und bieten eine sehr gute User Experience (Benutzererfahrung). Für die Entwicklung wird z.B. «Java» für «Android» und «Swift» für «iOS» benutzt. Da Native Apps plattformabhängig sind, muss für jedes Betriebssystem eine eigene App entwickelt werden. Native Apps sind im App Store der jeweiligen Plattform vorhanden und können von dort aus installiert werden. Ein Merkmal der Native Apps ist, dass sie meist von einem Betriebssystem zu den anderen unterschiedlich aussehen und nicht das exakt gleiche UI Design haben. Ein Nachteil der Native Apps besteht im hohen Entwicklungsaufwand und in den dadurch entstehenden Kosten. Beispiel für Native Apps sind «Ebay» und «WhatsApp». [14, 15]

2.4.2 Web Apps

Web Apps sind plattformunabhängig und von jedem Gerät oder Browser zugänglich. Sie besitzen somit eine grosse Reichweite und erreichen mehr Benutzer. Sie werden in einem Webbrowser durch eine «URL»¹ ausgeführt und passen sich an das jeweilige Gerät an, wenn sie geöffnet werden. Für die Entwicklung werden z.B. «HTML»² und «CSS»³ verwendet. Die Entwicklungskosten und der Aufwand sind viel geringer als bei Native Apps. Ein Nachteil der Web Apps sind Einschränkungen beim Zugriff auf bestimmten Hardware Komponenten der Geräte und die meist notwendige Internetverbindung. Beispiel für Web Apps sind Google Suche, Amazon und YouTube. [14, 15]

2.4.3 Hybride Apps

Hybride Apps sind, wie der Name verrät, eine Mischung aus Native- und Web Apps. Damit wird das Beste aus den beiden App-Typen kombiniert. Die Hybride Apps werden mit den Sprachen der Web Apps wie «HTML» und «CSS» programmiert und können sich wie die Native Apps an jedes Gerät anpassen. Sie haben Zugriff auf Gerätefunktionen wie die Kamera und das GPS. Sie benötigen für viele Funktionen keine Internetverbindung und können auf zuvor geladene Daten zurückgreifen wie z.B. bei «Instagram», wo die zuletzt geladenen Posts sichtbar sind, aber fürs Posten eines Bildes Internetverbindung nötig ist. Grosse Teile des Quellcodes sind für alle Plattformen gleich. Die Entwicklung ist schnell, braucht weniger Aufwand und die Kosten sind günstiger als bei Native Apps. Hybride Apps bieten ein besseres Erlebnis als bei Web Apps und sind z.B. im «App Store» oder «Google Play Store» vorzufinden. Ein bemerkbarer Nachteil der Hybride Apps ist, dass sie nicht so nah am Betriebssystem wie native Apps sind und das bei komplexen Anwendungen Probleme in der Performance auftreten können. «Instagram» ist ein Beispiel für eine Hybride App. [14, 15]

¹ Uniform Resource Locator (URL) ist Standard für das Aufrufen bestimmter Ressourcen, bspw. einer Webseite. (Wikipedia, 2021)

² Hypertext Markup Language (HTML) ist eine Sprache für die Beschreibung der Struktur von Web-Seiten. (selfhtml, 2021)

³ Cascading Style Sheets (CSS) ist eine Stylesheet-Sprache, mit der Gestaltungsanweisungen erstellt werden, bspw. für das Aussehen einer Webseite. (Wikipedia, 2021)

2.4.4 Cross-Plattform Apps

Cross-Plattform Apps werden in einer zwischengeschalteten, einheitlichen Programmiersprache entwickelt, welche nicht zum Betriebssystem eines Gerätes gehört. Sie werden mithilfe plattformübergreifender Anwendungsframeworks wie «React Native» oder «Flutter» in das jeweilige Betriebssystem kompiliert. Cross-Plattform Apps sehen wie eine echte native App aus und haben unter den verschiedenen Betriebssystemen ein einheitliches UI Design. Sie unterstützen die Hardware-Funktionalitäten und können die meisten Native-App Features wie Push Nachrichten und GPS-Funktionen ausführen.

Der grösste Teil des gesamten Programmcodes kann auf mehrere Plattformen verwendet werden. Der Source Code kann mit nativem Code ergänzt werden, zum Beispiel wenn eine Funktion nicht über den zwischen den Plattformen geteilten Source Code abgebildet werden kann. Die Entwicklungskosten sowie der Aufwand sind im Vergleich zu den anderen App-Typen gering. Cross-Plattform Apps erreichen ein ziemlich grosses Publikum und sind im App Store der jeweiligen Plattform vorhanden. Ein Nachteil der Cross-Plattform Apps ist, dass sie sich an die Konventionen und Vorschriften aller Plattformen anpassen müssen, was dafür sorgt, dass der Designprozess der App nicht frei möglich ist. Die Programmiersprachen sind je nach Framework unterschiedlich, hier sind einige Beispiele: «React Native» verwendet «JavaScript», «Flutter» verwendet «Dart» und «Xamarin» verwendet «C#». Beispiele für Cross-Plattform Apps sind «Philips Hue», «Google Assistant» und «Slack». [15]

2.5 UX Design und UI Design

2.5.1 UX Design

«UX Design» (engl. user experience design Benutzererfahrungsgestaltung) ist der Prozess der Entwicklung von einem Produktdesign, welcher darauf zielt, die Benutzer das bestmögliche Erlebnis zu liefern. Es umfasst verschiedene Aspekte wie z.B. Nutzerpsychologie, Ästhetik, Inhalt, Nutzerverhalten, Benutzerfreundlichkeit, Nützlichkeit und Attraktivität bei der Interaktion mit einem Produkt. [16, 17, 18]







2.5.2 UI Design



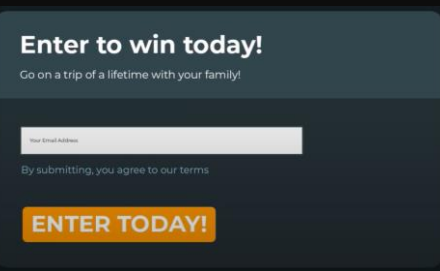

UI Design (engl. user interface design, Schnittstellengestaltung) ist ein Teilaspekt des UX Design. Darunter versteht man die Gestaltung von Benutzeroberflächen für Maschinen und Software, wie z.B. Haushaltegeräte, mobile Geräte und andere elektronischen Geräte. Es befasst sich mit der Optik und Effizienz einer Benutzeroberfläche wie z.B. eine Webseite oder eine App. Das UI Design bestimmt, wie ein Knopf oder eine Navigationsleiste in einer App aussieht. Dabei wird auch auf das praktische Nutzen dieser Elemente geachtet und berücksichtigt. [17, 18]

Beim UI Design müssen mehrere Prinzipien und Punkte berücksichtigt werden. Die wichtigsten Prinzipien sind gelistet. In **Tabelle 1** sind einige Beispiele für gute und schlechte Umsetzungen dieser Punkte:

- Farbe und Kontrast
- Komplexität gegen Einfachheit
- Visuelle Hierarchie (welche Elemente sind wichtiger und sollen hervorstechen)
- Einheitlichkeit und Kontinuität
- Grössenverhältnis der Elemente zueinander

Tabelle 1: Beispiel für schlechte und gute Umsetzung von UI Design Grundsätze

	Schlecht	Gut
Farbe & Kontrast	 <p>Schlechte Kombination von Farben Zu schwacher Kontrast Schwierig lesbar und anstrengend die Augen</p>	 <p>Guter Kontrast Einfach lesbar Gute Farbkombination</p>
Komplexität gegen Einfachheit	 <p>Caotisch und verstaut mit Effekten</p>	 <p>Schlicht und einfach</p>
Visuelle Hierarchie		

	Schlechte visuelle Unterteilung Elemente schwer erkennbar	Klare visuelle Anordnung Elemente durch Farbe und Grösse voneinander abgetrennt
Einheitlich- keit und Kontinuität	 <p>Nicht regelmässige Abstände Zu viele verschiedene Schriftarten</p>	 <p>Regelmässige Abstände Einheitliche Typografie</p>
Grössen-ver- hältnis	 <p>Unpassende Grössenverhältnisse</p>	 <p>Strukturierte und geeignete Grössen</p>

Quelle: Eigene Darstellung, Bilderquelle: Screenshots aus YouTube Tutorial, Simon (2019)

2.6 Cross-Plattform-Entwicklung

Mit Cross-Plattform-Entwicklung (plattformübergreifende Entwicklung) kann aus dem Quellcode eine App für verschiedene Betriebssysteme erzeugt werden. Dies ist mit Hilfe von Application-Frameworks, bzw. Cross-Plattform Frameworks, möglich. Ein Framework (Rahmenstruktur) ist ein Programmiergerüst, welches die Anwendung und Entwicklung vereinfacht. Es beinhaltet die Basisbausteine für ein Programm und stellt dem Entwickler die Vorlagen und Strukturen sowie Standardmodule und «API»¹ für die Anwendungsentwicklung zur Verfügung. Komponente und Software-Bibliotheken wie auch Tools vereinfachen dem Programmierer die Entwicklung und beschleunigt den Prozess. Application-Frameworks, welche zur plattformübergreifenden App-Entwicklung dienen, übersetzen den Code (Quellcode) aus einer frameworkspezifischen Sprache in die Sprache der jeweiligen Zielsysteme. Zu den bekanntesten plattformübergreifenden Appentwicklungstechnologien gehören «React Native», «Xamarin» und «Flutter». [19]

¹ Ein API (engl. application programming interface) auf Deutsch Programmierschnittstelle, regelt und ermöglicht die Kommunikation und das Interagieren zwischen Anwendungen. Sie fungieren als Vermittler wie z.B. ein die Bedienung in einem Restaurant welche die Bestellung aufnimmt und sie dem Koch übermittelt und nachher die Bestellung serviert. (Wikipedia, 2021)

2.7 Dart

Die Programmiersprache Dart ist eine sehr junge Programmiersprache, die von Google entwickelt und weiterentwickelt wird. Sie soll laut dem Hersteller eine modernere Alternative zur «JavaScript» Programmiersprache für die Verwendung in Webbrowser dastehen. Weil Webbrowser nicht native (ohne zusätzliche Zwischenschritte) mit Dart umgehen können und JavaScript in allen modernen Browsern optimal ausgeführt werden kann, wird der Compiler «Dart2js» (Dart zu JavaScript) gebraucht. Die Ähnlichkeit mit etablierten objektorientierten Programmiersprachen wie «Swift» und «Java» und die Verwandtschaft zur Syntax¹ mit der Programmiersprache «C» ermöglichen einen einfachen Einstieg in die Dart Sprache. [20]

Dart ist eine moderne, prägnante und objektorientierte Sprache. Sie verfügt über Funktionen, Klassen, Objekte, Operatoren, Variablen, Schleifen etc. Sie kann dank des open-Source-Tools «DartPad» in jedem modernen Browser ausprobiert und ausgeführt werden. Dart ist einfach lesbar, da die Syntax vom Hersteller nahe an der menschlichen Sprache gehalten wurde. Auf **Abb. 2:** «Hello World» Programm in Dart

Code-Quelle: Dart (2021), Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart

Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart

Code-Quelle: DartPad, o. D., Screenshot DartPadAbb. 2: «Hello World» Programm in Dart

Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart

Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart

Code-Quelle: DartPad, o. D., Screenshot DartPad

Abb. 2: «Hello World» Programm in Dart

Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart

Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart

Code-Quelle: DartPad, o. D., Screenshot DartPadAbb. 2: «Hello World» Programm in Dart

Code-Quelle: Dart, 2021, Screenshot DartPad

¹Regeln zur Kombinations von festgelegten Zeichen. (Wikipedia, 2021)

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPadAbb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPadAbb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPad
Abb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPadAbb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPad

Abb. 4: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Abb. 5: «Hello World», Emulator AnsichtAbb. 4: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Abb. 4: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Abb. 5: «Hello World», Emulator AnsichtAbb. 4: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Abb. 4: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Abb. 5: «Hello World», Emulator AnsichtAbb. 4: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Abb. 4: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Abb. 5: «Hello World», Emulator AnsichtAbb. 4: «Stateless Widget» Beispiel, «Hello World» ProgrammAbb. 3:
Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPadAbb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPadAbb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPad
Abb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in DartAbb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart

Code-Quelle: DartPad, o. D., Screenshot DartPad
Abb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Abb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPad
Abb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Abb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPad
Abb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Abb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPad
Abb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Abb. 2: «Hello World» Programm in Dart
Codequelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad, o. D., Screenshot DartPad
Abb. 2: «Hello World» Programm in Dart
Code-Quelle: Dart, 2021, Screenshot DartPad

Abb. 3: Beispiel eine «for-Schleife» in Dart
Abb. 2 **und** Abb. 3: Beispiel eine «for-Schleife» in Dart

Code-Quelle: DartPad (o. D.), Screenshot DartPad sind zwei einfache Beispiele der Dart Programmiersprache abgebildet.

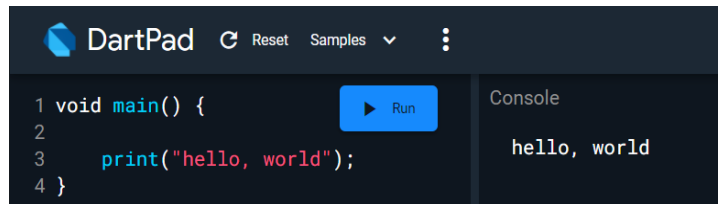


Abb. 162: «Hello World» Programm in Dart
Code-Quelle: Dart (2021), Screenshot DartPad

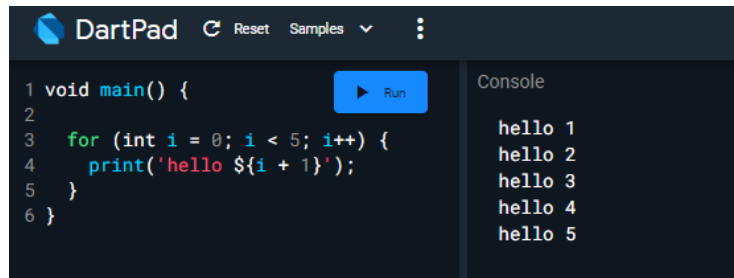


Abb. 238: Beispiel eine «for-Schleife» in Dart
Code-Quelle: DartPad (o. D.), Screenshot DartPad

2.8 Flutter

Flutter ist eine von Google erstellte Open-Source-UI-Entwicklungs-Kit. Das quelloffene Framework ermöglicht die Entwicklung von Cross-Plattform Apps in der Programmiersprache «Dart». Mit Flutter können aus einer einzigen Codebasis Apps für die verschiedenen Plattformen wie «Android» oder «iOS» hergestellt werden. Die Hersteller Flutters werben mit den kurzen Entwicklungszeiten, der schnellen Ausführungsgeschwindigkeit und einem nativeren Benutzererlebnis, welche alle das Framework ermöglichen und speziell machen. Flutter kann auf «Windows», «macOS», «Linux» sowie «Chrome OS» installiert und eingerichtet werden. [21]

2.8.1 Everything is a Widget

Im Flutter ist jedes Element ein «Widget»¹. Es ist die grundlegende Komponente in einem Flutter-Programm. Ein Knopf ist ein Widget, ein Kasten ist ein Widget, ein Text ist ein Widget, sogar eine Seite wie z.B. eine Profilseite ist ein Widget, wobei ein Widget wiederum aus Widgets besteht. Der Zustand eines Widgets wird in einem «State-Objekt» gespeichert, wodurch der Zustand des Widgets von seinem Aussehen getrennt wird. Der Zustand besteht aus Werten, die sich ändern können, wie der aktuelle Wert eines Schiebereglers oder ein Kontrollkästchen markiert ist. Wenn sich der Zustand des Widgets ändert, ruft das «State-Objekt» «setState()» auf und weist das Framework an, das Widget neu zu zeichnen.

2.8.2 Stateless und Stateful Widgets

Widgets werden in zwei Arten unterschieden, «stateless» (zustandlos) -und «stateful» (zustandsbehaftet) Widget. «Stateless Widget» sind Widgets, deren Zustand nach der Erstellung

¹ Programmelemente, die in Grafische Benutzeroberfläche (GUI, engl. graphical user interface) eingebunden werden.
«Widget» ist ein Kunstwort aus zwei Wörtern «Window» (dt. Fenster) und „Gadget“ (dt. Gerät) zusammengesetzt. (Ryte, o. D.)

nicht mehr verändert werden kann. Sie sind unveränderlich, was bedeutet, dass die Widgets nicht neu gezeichnet werden können, während die Anwendung in Aktion ist. Beispiel für «Stateless Widgets» sind Text und Icon (Bild, Piktogramm).

«Stateful Widgets» sind Widgets, deren Zustand nach der Erstellung geändert werden kann. Sie werden auch als zustandsabhängige Widgets bezeichnet. Diese Zustände sind veränderbar und können während ihrer Lebensdauer mehrfach geändert werden. «Stateful Widgets» können ihr Aussehen als Reaktion auf Ereignisse ändern, wenn sie Daten erhalten oder durch Benutzerinteraktionen ausgelöst werden. Beispiel für Stateful Widgets sind Checkboxes und Slider (Schieberegler). [22, 23, 24, 25, 26]

Tabelle 2: «Stateful Widget» und «Stateless Widget» im Vergleich

Stateful Widget	Stateless Widget
Statische Widgets; sie werden nur bei der Initialisierung aktualisiert	Sind dynamisch
Ist nicht abhängig von Datenänderungen oder Verhaltensänderungen	Sie ist abhängig und ändert sich, wenn der Benutzer interagiert
Beispiele sind Text, Icons	Beispiele sind Checkbox oder Slider
Enthält kein setState()	Hat ein internes setState()
Kann nicht während der Laufzeit der Anwendung aktualisiert werden	Kann während der Laufzeit aktualisiert werden

Quelle: In Anlehnung an Mistry (2021)

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

  @override
  Widget build(BuildContext context) {
    return Text( "hello, world" );
  }
}
```

Abb. 480: «Stateless Widget» Beispiel, «Hello World» Programm
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

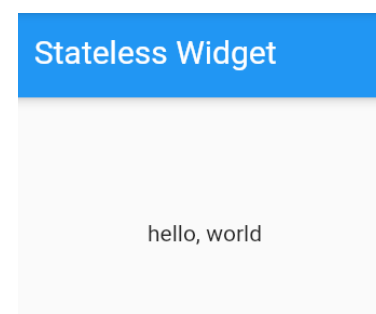


Abb. 239: «Hello World», Emulator Ansicht
Quelle: Eigene Darstellung

```

class CheckboxBeispiel extends StatefulWidget {
  const CheckboxBeispiel({Key? key}) : super(key: key);

  @override
  _CheckboxBeispielState createState() => _CheckboxBeispielState();
}

class _CheckboxBeispielState extends State<CheckboxBeispiel> {
  bool _checked = false;

  @override
  Widget build(BuildContext context) {
    return Checkbox(
      value: _checked,
      onChanged: (bool? value) {
        setState(
          () {
            _checked = value!;
          },
        );
      },
    );
  }
}

```

Abb. 585: «Stateful Widget» Beispiel, «Checkbox» Widget
Quelle: Eigene Darstellung, Screenshot Visual Studio Code

Stateful Widget



Abb. 808: «Checkbox» Widget, nicht ausgewählt, Emulator Ansicht

Stateful Widget



Abb. 969: «Checkbox» Widget, abgehakt, Emulator Ansicht
Quelle: Eigene Darstellung

Da die Benutzeroberfläche mehrere Widgets enthält, werden die Widgets zu einem «Widget tree» (Widget-Baum) kombiniert, das erklärt die Verhältnisbezeichnung «child», «children» und «parent» in Flutter. Ein Widget (Widget B), welches nach einem anderen Widget (Widget A) kommt, wird als «child» (Kind) bezeichnet. Das heisst, Widget A ist der «parent» (Vater) vom Widget B. Zu jedem erstellten Widget in Flutter wird gleichzeitig ein Element vom Flutter-Framework erstellt. Diese kombinierten Elemente werden als «element tree» (Elementbaum) bezeichnet.

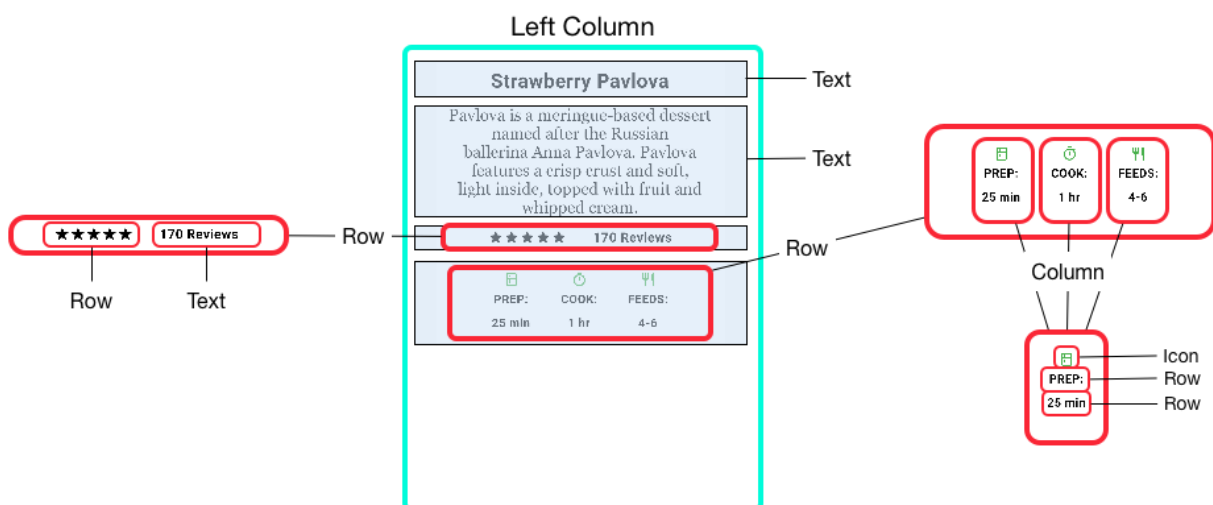


Abb. 1034: Beispiel eines Layouts, Anordnung von Widgets
 Quelle: Flutter; Layouts in Flutter (2021)



Abb. 1035: Bewertungsreihe im Widgets
 Layouts von Abb.9
 Quelle: Flutter; Layouts in Flutter (2021)

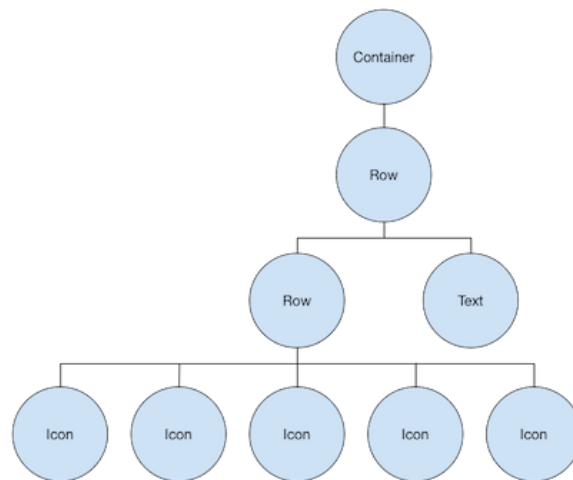


Abb. 1036: «widget tree», der Bewertungsreihe auf Abb.
 1035: Bewertungsreihe im Widgets Layouts von Abb.9

Damit enthält der «widget-tree» die gesamte Widget-Konfiguration und der «element tree» alle gerenderten Widgets auf dem Bildschirm. Es gibt noch einen weiteren Baum namens «render tree» (Renderbaum), mit dem der Benutzer aber nicht interagieren muss. Die render tree ist im Grunde ein einfaches Layoutsystem, das von den Renderobjekten erbt.

Mehrere Widgets, die unter zustandslosen Widgets zusammengefasst/kombiniert sind, werden als «stateless widget tree» bezeichnet und die entsprechenden Elemente bilden einen Elementbaum. Flutter verwendet die «createElement»-Methode, um die Elemente zu erstellen. Jedes erstellte Element hat eine eindeutige ID, die vom Flutter-Backend zugewiesen wird.

Bei dem «Stateful Widget» wird beim Aufruf der «createElement»-Methode durch das Framework ein «state object» (Zustandsobjekt) erstellt. Der Grund für die Erstellung des Zustandsobjekts ist, dass sich die Zustände bei einer dynamischen Anwendung ändern würden, während der vorherige Zustand erhalten bleibt und die Leistung verbessert wird. Es wird eine «setState»-Methode angewandt, um die Statusobjekte zu aktualisieren. Der Ausgangszustand «initial state» im Elementbaum wird aber auch nach einer Änderung des Zustands beibehalten. [24]

3 Methode

3.1 Material und Vorbereitung

Ohne Vorwissen in der Appentwicklung wendete ich mich an die Google Suche und das Internet. Im Internet wurde ich über Apps, Appentwicklung und verschiedene App-Typen

fündig. Von den verschiedenen Möglichkeiten entschied ich mich für die Cross-Plattform Entwicklung, da es meinen Ansprüche am besten entsprach. Die grosse Anzahl erreichbarer Nutzer und den im Vergleich zu den anderen App-Typen gering verbundenen Aufwand mit einem einzigen Quellcode verschiedener Plattformen zu erreichen sowie dass es nahe an eine Native App Leistung kam, machten sie zum besten Kandidaten. Unter den bekanntesten plattformübergreifenden Mobileentwicklungstechnologien musste noch die Wahl zwischen «React Native» und Flutter getroffen werden. In **Tabelle 3** werden die beiden Arten verglichen:

Tabelle 3: Flutter und React Native im Vergleich

	Flutter	React Native
Technologie	Portables UI-Toolkit zur Erstellung nativ kompilierten Anwendung	Framework zur Erstellung nativer Anwendungen
Offizielle Freigabe	Dezember 2018	März 2015
Hersteller	Google	Facebook
Programmiersprache	Dart	JavaScript
Sharing code	Dieselbe Codebasis für Android, iOS, Windows, macOS, Linux und Webanwendungen für Browser wie Chrome oder Safari	Dieselbe Codebasis für Android, iOS und Web Apps
User Interface (UI)	Apps sehen auf Android und iOS sehr ähnlich aus Apps sehen auf den Betriebssystemen genauso gut aus wie auf älteren Versionen.	Anwendungskomponente sehen genauso wie native Komponenten der jeweiligen Plattform. z.B. Schaltfläche auf einem Android Gerät sieht genauso wie eine native Android Schaltfläche
Markteinführungszeit (Time-to-Market)	Schnell, viel schneller als die native Entwicklung	In etwa so schnell wie die Entwicklung mit Flutter
Community (Gemeinschaft)	Weniger als bei React Native, aber sehr schnell wachsende Gemeinschaft die immer mehr an Popularität zunimmt.	Reife und grosse Gemeinschaft

Dokumentation	Sehr gut, mit starker Unterstützung durch das Flutter-Team	Gut, aber nicht so ausführlich und detailliert wie bei Flutter
App Leistung	Näher an Native Apps	Gut, weniger als bei Flutter Apps

Quelle: In Anlehnung an Skuza, Mroczkowska und Włodarczyk (2021)

Aufgrund der fehlenden Kenntnisse in den beiden Programmiersprachen «Dart» und «JavaScript» wie auch des erforderlichen Erlernens einer der beiden Sprachen, ist die Wahl auf die gefallen, welche am schnellsten und am einfachsten zu erlernen ist. «Dart» ist allgemein einfacher im Vergleich mit «JavaScript». Das ist auf der einfachen und der an den menschlichen Sprachen näheren Syntax von Dart zurückzuführen. Aus diesem Grund und wegen der besseren Flutter Apps Leistung wie auch den ähnlichen Benutzeroberflächen (UI) auf den verschiedenen Plattformen sowie den ausführlicheren Dokumentationen schien Flutter im Vergleich zu React Native der bessere Kandidat zu sein. Dementsprechend wurde diese Art deshalb ausgewählt. Ausserdem ist Flutter für Einsteiger in der Entwicklung von Apps simpler, was für ein Anfänger wie mich optimal war. Darüber hinaus beschleunigt Flutter die App-Entwicklung und reduziert die Kosten sowie die Komplexität der App-Produktion auf allen Plattformen. Eine kürzere Entwicklungszeit kann beim Erreichen eines meiner Arbeitsziele, den Übergang von einer Idee zu einer App, beitragen und erhöht die Chancen, die Appvorlage Prototyp in so einer kurze Frist im Rahmen der Maturaarbeit zu entwickeln. Kosten zu sparen ist natürlich auch von grossem Vorteil.

Wegen dem Fehlen eines konkreten App Konzepts meinerseits wurde nach einem Gespräch mit Herrn Weyermann die Entwicklung einer einfachen Taschenlampen App als erster Hindenstein gesetzt. Das Ziel war, eine sehr simple Taschenlampen App zu entwickeln und sie auf einen «App Store» hochzuladen. Um dies zu erreichen, musste zuerst eine Entwicklungsumgebung vorhanden sein.

3.2 Flutter Installation

Für die Flutter Installation und Einrichtung gibt es genaue Anweisungen auf der Flutter Webseite. Zuerst muss das passende Flutter-Paket für das jeweilige Betriebssystem ausgewählt werden, auf welchem Flutter installiert werden soll. Dabei kann es zwischen Paketen für die Betriebssysteme «Windows», «macOS», «Linux» und «Chrome» ausgewählt werden.

Da ich nur einen «ThinkPad L13 Yoga» Laptop habe, welcher Windows als Betriebssystem besitzt, konnte ich Flutter nur auf Windows installieren. Vorzugsweise hätte ich ein Apple Gerät für die Entwicklung genutzt, weil sich darauf die Flutter App sowohl für Android als auch für iOS entwickeln liesse. Das Betriebssystem «macOS» unterstützt die gesamte Entwicklung, was beim «Windows» Betriebssystem nicht möglich ist. Das liegt daran, dass für die Erstellung von App-Versionen wie z.B. «Release»¹ für iOS Geräte ein «Xcode»²

¹ Die fertige Software-Version, welche auch publiziert wird. (siehe Abschnitt Testen und Veröffentlichung, Seite 9)

² Von Apple integrierte Entwicklungsumgebung «IDE» (engl. integrated development environment). (Wikipedia, 2021)

erforderlich ist. Mit Xcodes können «iOS», «ipadOS» wie auch «macOS» Programme entwickelt werden.

Um Flutter auf Windows installieren und laufen können, muss die Entwicklungsumgebung die Mindestanforderungen erfüllen.

Betriebssysteme: Windows 7 SP1 oder höher (64-Bit), x86-64-basiert.

Festplattenspeicher: 1,64 GB (umfasst nicht den Festplattenspeicher für IDE/Tools).

Werkzeuge: Flutter hängt davon ab, dass diese Werkzeuge in Ihrer Umgebung verfügbar sind.

- Windows PowerShell¹ 5.0 oder neuer (ist bei Windows 10 vorinstalliert)
- Git² für Windows 2.x, mit der Option Git von der Windows-Eingabeaufforderung³ zu verwenden.

Abb. 1099: «Flutter» Mindestanforderungen für ein «Windows» Betriebssystem, Datum: 17.10.2021

Quelle: Flutter; Windows Install (2021)

Nach der Überprüfung der Kompatibilität des Laptops und das Windowsbetriebssystem wurde die Flutter SDK⁴ installiert. Ein wichtiger Hinweis dabei ist, Flutter nicht direkt in «C:\Programdateien\...» zu installieren. Wenn Flutter-Befehle in der normalen Windows-Konsole ausgeführt werden, müssen die Umgebungsvariablen aktualisiert und der Speicherort von Flutter SDK zu deren Pfaden hinzugefügt werden. Um die Installation und Lauffähigkeit vom Flutter zu überprüfen, kann der Befehl „flutter doctor“ ausgeführt und im Falle von Problemen den Anweisung gefolgt werden, um diese zu beseitigen. [27]

Um Flutter Anwendungen für Android zu erstellen, können verschiedene Texteditoren in Kombination mit Flutter «command-line tools» verwendet werden. Flutter empfiehlt jedoch die Verwendung eines ihrer Editor-Plugins⁵ wie «Visual Studio Code» für eine bessere

¹ PowerShell, engl. Power (Kraft), Shell (Hülle). ist eine plattformübergreifende Framework von Microsoft zur Aufgabenautomatisierung und Konfigurationsverwaltung. Kann unter Windows, macOS und Linux ausgeführt werden. (Wheeler & Oplrod, 2021)

² Windows-Eingabeaufforderung bekannt als «CMD», engl. Command Prompt, ist ein textbasiertes Benutzeroberfläche innerhalb des Betriebssystems. Dadurch kann der Nutzer Befehle and das Betriebssystem geben, wie z.B. ein Programm starten oder Dateien organisieren. (Ionos, 2020)

³ Git, (engl. git, in British English slang, unpleasant person) ist ein verbreitetes Versionskontrollsystem, das von Linus Torvalds, dem Linux Betriebssystem Entwickler, entwickelt wurde. (Wikipedia, 2021)

⁴ Ein Software Development Kit (SDK) ist eine Sammlung von Programmierwerkzeugen und Programmbibliotheken, die zur Entwicklung von Software dient. (Wikipedia, 2020)

⁵ Plugin (Software-Erweiterung oder Zusatzprogramm) ist eine Software-Komponente, die eine Software erweitert bzw. verändert. (Wikipedia, 2021)

Erfahrung. Diese Plugins bieten Code-Vervollständigungen, Syntax-Hervorhebungen, Unterstützung bei den Widget-Bearbeitungen, «Run» Unterstützung und «Debug» (Fehler beseitigen) Unterstützung. Aber um Flutter Android-Plattform-Abhängigkeiten bereitzustellen und Softwareversionen zu erstellen, ist es einfacher, Flutter mit der offiziellen von Google erstellten «IDE», «Android Studio»¹, welche spezifisch für die Entwicklung von Android Apps ist. Android Studio IDE enthält auch das «Android Virtual Device Manager»² (AVD Manager) Werkzeug, welches mit dem «Android Emulator» dazu dient, Android Geräte auf dem Computer zu simulieren.

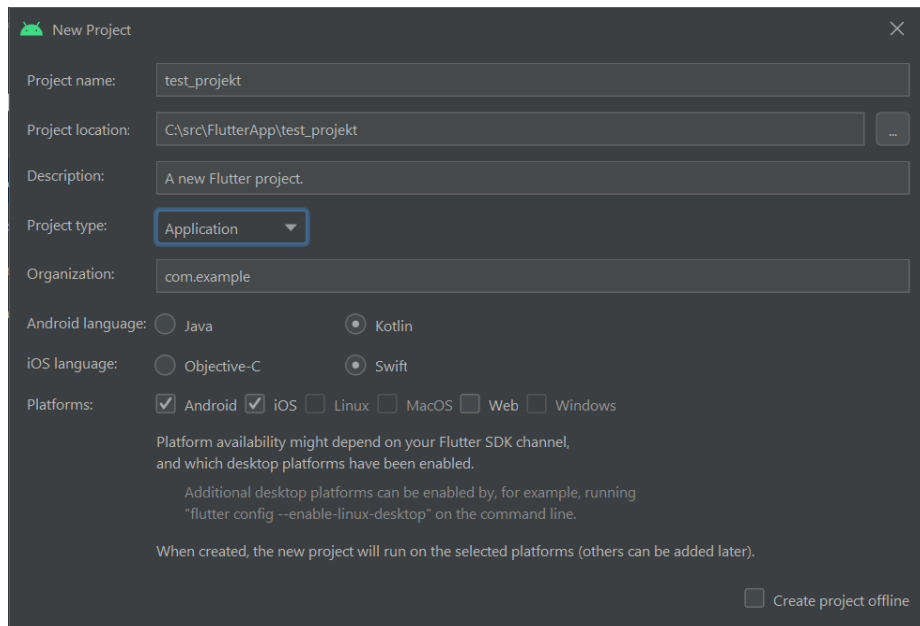
Um ein Android Emulator zu erstellen, können zwischen verschiedenen vorhandenen Geräten Vorlagen ausgewählt oder neu individualisierte erstellt werden. Dabei muss für die Erstellung ein «System Bild» ausgewählt werden, die Android Versionen und die jeweils dazugehörigen verschiedenen API-Stufen haben wie z.B. «Pie», API-Stufe 28 für «Android 9», oder «R», API-Stufe 30 für «Android 11»,. Die jeweiligen «SDK» wird dabei installiert oder kann auch manuell mit dem «SDK Manager» installiert und konfiguriert werden.

3.3 Erstellung und Aufbau eines Flutter-Projekts

Nach der Installation von Flutter und Android Studio wurden die erforderliche Flutter-Plugin: Flutter und Dart installiert und eingerichtet. Um ein Überblick über Flutter zu bekommen, wurde ein Flutter-Projekt erstellt. Für die Erstellung wird nach einem Projektnamen und der Organisation gefragt. Es kann zwischen den Programmiersprachen für die Android sowie iOS Plattform, der Projektart und den Plattformen gewählt werden (siehe **Abb. 13**). Diese können auch später geändert werden, jedoch ist es viel einfacher und es wird von Flutter empfohlen, diese am Anfang zu machen. Der Projektname und die Organisation werden als Paketname für die Android und die «Bundle-ID» für iOS verwendet, wenn die App veröffentlicht wird.

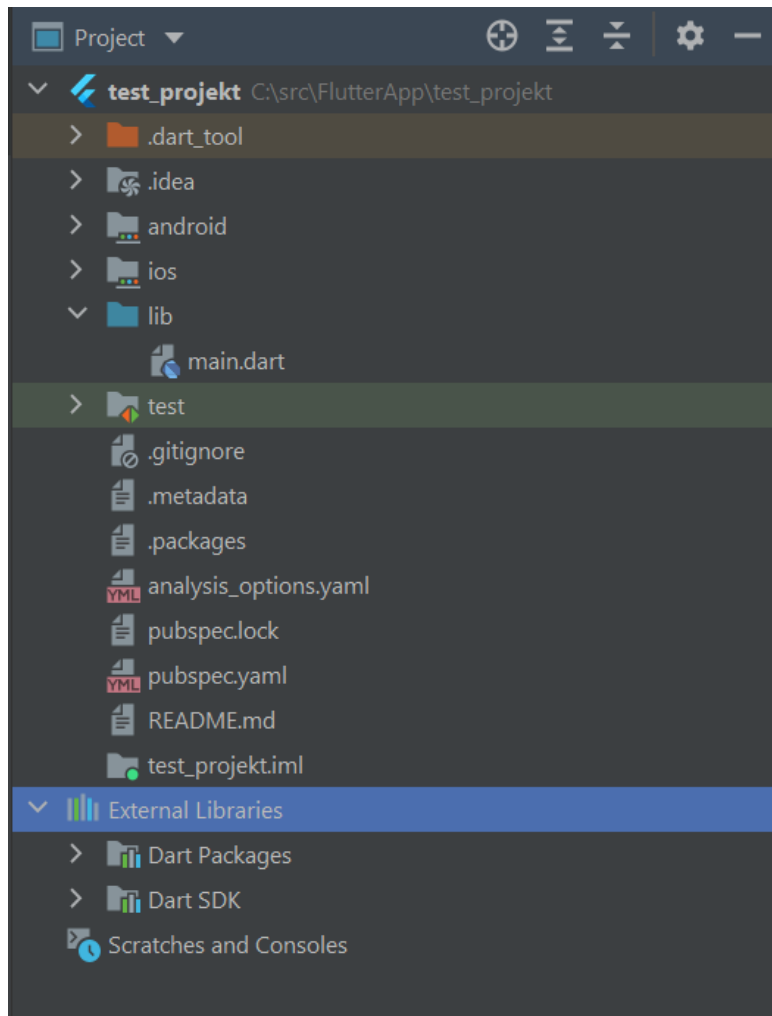
¹ IDE für die Android-Softwareentwicklung. (Google Developers, o.D.)

² AVD Manager: virtuelles Android-Gerät, ist eine Konfiguration, die die Eigenschaften eines Android-Telefons, -Tablets, -Wear OS-, -TV oder Automotive OS-Geräts definiert, welche im Android Emulator simuliert werden können. (Google Developers, o.D.)



*Abb. 1100: Auswahlmöglichkeiten bei der Erstellung eines Flutter Projekts
Quelle: Screenshot Android Studio*

Jedes Flutter-Projekt besteht grundsätzlich aus zwei Ordnern. Ein App-Ordner, der mit dem Projektnamen wie z.B. «test_projekt» benannt ist, und den gesamten Quellcode enthält sowie einen anderen Ordner namens «External Libraries», welcher alle externen Bibliotheken, die für das Projekt installiert und gebraucht werden, enthält (siehe **Abb. 14**). Der Projektordner enthält Plattformspezifische Ordner, deren Anzahl je nach Menge der bei der Erstellung des Projektes ausgewählten Plattformen variieren. Also alles, was mit der Android Plattform zu tun hat, wird im Android Ordner sein und dies gilt auch für die anderen Plattformen. Ausser diesen Plattformspezifischen Ordnern gibt es einen sogenannten «lib» Ordner, worin die meiste Programmierung entwickelt wird, und somit enthält er die meisten Projektcodes. In diesem «lib» Ordner befindet sich eine «main.dart» Datei, welche als Einstiegspunkt dient und beim Starten des Projekts als erstes ausgeführt wird. Neben den erwähnten Ordner gibt es im Projekt-Ordner einen «test»-Ordner, der Testdateien enthält, welche für das Testen der Anwendung zuständig sind. Neben allen anderen Ordnern und Dateien im Projekt-Ordner befinden sich darin auch mehrere Konfigurationsdateien. In **Abb. 14** wird ein Beispiel einer Flutter-Projekts Ordnerstruktur dargestellt.



*Abb. 1101: Ordnerstruktur eines Flutter Projekts
Quelle: Screenshot Android Studio*

3.4 Taschenlampen App

Nach der Flutter Einrichtung wurde nach Beispielen für eine Taschenlampen App im Internet gesucht. Es wurde auf «YouTube» wurde ein Video mit dem Titel «Flashlight App In Flutter» von einem YouTuber namens «Geeky Bharat» als Orientierung für mein Vorhaben gewählt und der Quellcode vom Youtuber «GitHub Repository» als Basis für die Entwicklung der Taschenlampen App verwendet. Die Anleitung des YouTube Videos wurde Schritt für Schritt mit Schneckentempo nachgemacht. In **Abb. 15** und **Abb. 16** ist der Code der App abgebildet und erläutert. [28, 29]

```

1  import 'package:flutter/material.dart';
2  import 'package:flashlight/flashlight.dart';
3
4  void main() => runApp(MyApp());
5
6  class MyApp extends StatefulWidget {
7    @override
8    _MyAppState createState() => _MyAppState();
9  }
10
11  class _MyAppState extends State<MyApp> {
12    var _hasFlashlight = false;
13    var isON = false;
14
15    checkLight() async {
16      bool hasFlash = await Flashlight.hasFlashlight;
17      setState(() {
18        _hasFlashlight = hasFlash;
19      });
20    }
21
22    @override
23    initState() {
24      super.initState();
25      initFlashlight();
26    }
27
28    initFlashlight() async {
29      bool hasFlash = await Flashlight.hasFlashlight;
30      print("Device has flash ? $hasFlash");
31      setState(() {
32        _hasFlashlight = hasFlash;
33      });
34    }
35
36    @override
37    Widget build(BuildContext context) {
38      return MaterialApp(
39        home: Scaffold(
40          appBar: AppBar(
41            elevation: 0.0,
42            backgroundColor: Colors.orangeAccent,
43            title: const Text('Flashlight'),
44            centerTitle: true,
45          ),

```

Abb. 1102: Code der fertigen Taschenlampen App (Teil 1)
 Code-Quelle: In Anlehnung an Tiwari (2021), Screenshot Visual Studio Code

3.4.1 Code Erläuterung

1. material Paket wird importiert

2. flashlight Paket wird importiert

import-Anweisung: Die import-Anweisung wird verwendet, um die Bibliotheken zu importieren, die vom Flutter-SDK bereitgestellt werden. Hier haben wir die Datei material.dart importiert. Wir können alle Flutter-Widgets verwenden, die das Materialdesign implementieren, indem wir diese Datei importieren.

4. main Funktion ruft den MyApp Konstruktor.

main()-Funktion: Wie viele andere Programmiersprachen hat auch Dart eine main-Funktion, in der die Anweisungen eingetragen werden müssen, die beim Start der App ausgeführt werden sollen. Der Rückgabotyp der main-Funktion ist 'void'.

runApp(Widget widget) Funktion: Die void runApp(Widget widget) nimmt ein Widget als Argument und setzt es auf einen Bildschirm. Sie gibt dem Widget die Beschränkungen, damit es in den Bildschirm passt. Es macht das angegebene Widget zum «root» (Wurzel) Widget der App und andere Widgets zu dessen Unterelementen. Die MaterialApp wird als Wurzel-Widget verwendet, in dem die anderen Widgets definiert werden.

6. Der Name des Stateful Widgets ist MyApp, das von runApp() aufgerufen wird und ein Stateful Widget erweitert.

Mit dem Schlüsselwort extends kann es von einer Klasse geerbt oder erweitert werden. Auf diese Weise können Eigenschaften und Methoden zwischen Klassen ausgetauscht werden, die zwar ähnlich, aber nicht genau gleich sind.

7. In der Klasse MyApp wird die Funktion createState überschrieben.

@override weist lediglich darauf hin, dass die Funktion auch in einer Vorgängerklasse definiert ist, aber neu definiert wird, um in der aktuellen Klasse etwas anderes zu tun.

8. Die createState() Funktion wird verwendet,

um einen veränderbaren Zustand für dieses Widget an einer bestimmten Stelle im widget tree (Baum) zu erzeugen. Diese Methode gibt eine Instanz für die betreffende Unterklasse des Zustands zurück.

11. Die andere Klasse, _MyAppState, erweitert den Status,

der alle Änderungen im Widget verwaltet.

12. Eine Variable mit dem Namen _hasFlashlight wird definiert und zu falsch gesetzt.

Die Unterstriche « _ » in den Variablen-, Klassen- und Methodennamen machen sie privat, das bedeutet, dass sie nur in der «.dart-Datei» verfügbar sind, in der sie definiert wurden.

13. Eine Variable mit dem Namen isON wird definiert und zu falsch gesetzt.

15-20. Eine asynchrone Methode mit dem Namen checkLight wird definiert.

Asynchrone Funktionen:

Sie bilden die Grundlage der asynchronen Programmierung. Diese asynchronen Funktionen haben async-Modifikatoren in ihrem Körper. Async und await in Dart sind anderen Programmiersprachen sehr ähnlich. Wenn eine asynchrone Funktion aufgerufen wird, wird sofort ein Future (Zukunft) zurückgegeben und der Körper der Funktion wird später ausgeführt. Während der Körper der asynchronen Funktion ausgeführt wird, wird der vom Funktionsaufruf zurückgegebene Future zusammen mit seinem Ergebnis abgeschlossen.

16. Ein Boolean „hasFlasch“ wird definiert.

Ein Boolean kann nur die Werte true oder false annehmen.

Es wird überprüft, ob das Gerät eine Taschenlampe hat oder nicht.

17-19. Die setState Methode setzt den Wert von der Variable _hasFlashlight

neu zu dem Wert von `hasFlash`.

Je nachdem ob es eine Taschenlampe am Gerät gibt (`hasFlash = wahr`) oder (`hasFlash = falsch`) wird der Wert von «`_hasFlashlight`» neu zum Wert von «`hasFlash`» gesetzt.

23-26. Die «`initState`» wird gerufen.

Die Methode `initState()` wird einmal aufgerufen, wenn das zustandsabhängige Widget in den Widget-Baum eingefügt wird. Diese Methode wird im Allgemeinen überschrieben, wenn irgendeine Art von Initialisierungsarbeit geleistet werden muss, wie z.B. die Registrierung eines listeners, da diese Methode, anders als `build()`, nur einmal aufgerufen wird.

`initState()` ruft zuerst `super.initState()` und dann die `initFlashlight` Methode

28-34. Eine asynchrone `initFlashlight` Methode wird definiert.

29-30. Der Boolean `hasFlash` wird definiert.

Es wird auf den Wert von `hasFlash` gewartet und in der Console gedruckt,

z.B. Wenn der `hasFlash = wahr` ist, wird «Device has flash? Wahr» in der Console gedruckt.

31-33. Die `setState` Methode setzt den Wert von `_hasFlashlight` neu zum Wert von `hasFlash`.

36-27. Innerhalb der `_MyAppState` Klasse wird die `build`-Funktion überschrieben, die den `BuildContext` als Parameter erhält.

Diese Build-Funktion gibt ein Widget zurück, indem die Benutzeroberfläche der App gestaltet wird. Da es sich um ein `StatefulWidget` handelt, wird die `build`-Funktion viele Male aufgerufen, wodurch die gesamte Benutzeroberfläche mit allen Änderungen erneut erstellt wird.

Jedes Widget in Flutter wird durch die Build-Methode erstellt. Die Build-Methode nimmt einen `BuildContext` als Argument. Dies hilft der Build-Methode, das zu zeichnende Widget zu finden und die Position des zu zeichnenden Widgets im Widget-Baum zu bestimmen.

Wenn ein Widget einige untergeordnete Widgets hat, dann haben sowohl das übergeordnete als auch die untergeordneten Widgets ihren eigenen `BuildContext`. Nun wird der `BuildContext` des Eltern-Widgets der Elternteil des Build-Contextes seiner direkten Kinder-Widgets sein. Z.B. die `MaterialApp` hat sein eigenes `BuildContext` und ist der Vater von `Scaffold`.

`Scaffold` hat sein eigenes `BuildContext` und ist der Vater von `Container` und so weiter ...

38. Die `build` Methode gibt `MaterialApp` zurück.

`MaterialApp()`-Widget: `MaterialApp`-Klasse ist eine vordefinierte Klasse in Flutter. Sie ist wahrscheinlich die Haupt- oder Kernkomponente von Flutter. Wir können auf alle anderen Komponenten und Widgets zugreifen, die vom Flutter SDK bereitgestellt werden. `Text`-Widget, `AppBar`-Widget, `Scaffold`-Widget, `StatelessWidget`, `StatefulWidget`, etc.

39. Innerhalb von `Scaffold`-Widget können verschiedene Eigenschaften wie `home`, `appBar`, `body`, `floatingActionButton`, `backgroundColor`, etc definiert werden.

40-45. Zum Beispiel in der `AppBar`-Eigenschaft wird das `AppBar()`-Widget verwendet,

indem mit einem `Text`-Widget der Titel «Flashlight» übergeben wird, der dann oben in der Anwendung in der `AppBar` angezeigt wird.

```

46  body: Container(
47    decoration: BoxDecoration(
48      image: DecorationImage(
49        image: AssetImage("assets/images/1.png"),
50        fit: BoxFit.cover,
51      ),
52    ),
53    child: _hasFlashlight
54      ? Center(
55        child: Transform.rotate(
56          angle: -3.14 / 2,
57          child: Transform.scale(
58            scale: 4.0,
59            child: Switch(
60              activeColor: Colors.orangeAccent,
61              value: isON,
62              onChanged: (value) {
63                setState(
64                  () {
65                    isON = !isON;
66                    isON ? Flashlight.lightOn() : Flashlight.lightOff();
67                  },
68                );
69              },
70            ),
71          ),
72        ),
73      )
74      : Text(
75        "You Don't have a flash",
76        style: TextStyle(
77          color: Colors.redAccent,
78          fontSize: 20.0,
79        ),
80        textAlign: TextAlign.center,
81      ),
82    ),
83  ),
84 );
85 }
86 }

```

Abb. 16: Code der fertigen Taschenlampen App (Teil 2)
Code-Quelle: In Anlehnung an Tiwari (2021), Screenshot Visual Studio Code

46-52. In der Eigenschaft body wird ein Container-Widget verwendet, die wie verschiedene Eigenschaften wie decoration, image, color etc hat.

Der image-Eigenschaft wird ein AssetImage-Widget mit dem Speicherort vom Bild übergeben. Dieses Bild wird dann als Hintergrund in der Anwendung gezeigt.

Assets (manchmal auch Ressourcen genannt). Ein Asset ist eine Datei, die gebündelt und bereitgestellt wird und zur Laufzeit zugänglich ist.

Sie werden in pubspec.yaml, einer der Konfigurationsdateien, welches sich im Stammverzeichnis des Projekts befindet (siehe Abb. 17). Die pubspec.yaml Datei gibt Abhängigkeiten an, die das Projekt benötigt, wie z.B. bestimmte Pakete (und ihre Versionen), Schriftarten oder Bilddateien. Sie spezifiziert auch andere Anforderungen, wie z.B. Abhängigkeiten von Entwicklerpaketen (wie Test-Pakete) oder bestimmte Einschränkungen für die Version des Flutter-SDK.

53-54. Der child-Eigenschaft wird ein Center-Widget übergeben, Sie zentriert ihr Inhalt und Kinder-Widgets im Zentrum des Bildschirms.

53-82. Je nachdem ob der Wert vom _hasFlashlight true oder false ist, also ob es eine Taschenlampe gibt oder nicht, wird entweder ein Switch-Widget (Shalter) wenn (hasFlashlight = true) oder ein Text «You Don't have a

flash»

wenn(`_hasFlashlight = false`) im Zentrum des Bildschirms angezeigt (siehe **Abb. 1104**).

Im Fall vom `Switch`-Widget wird der Schalter dank dem `Transform.rotate`-Konstruktor, welcher ein Widget erzeugt, welches sein untergeordnetes Element durch eine Drehung um den Mittelpunkt transformiert. Der `Transform.scale`-Konstruktor, erzeugt ein Widget, das sein Kind gleichmäßig skaliert. Er kann den Schalter einerseits rotieren, andererseits skalieren.

- 60-69. Das `Switch`-Widget hat mehrere Eigenschaften, darunter `activeColor` (Schalterfarbe, wenn er aktiv ist), `value` (hier wird der Wert auf der `isON` Variable gesetzt, welcher oben gleich `false` gestzt ist) und `onChanged`.

`OnChanged` ist ein Callback Funktion (Rückruffunktion).

Callback ist im Grunde eine Funktion oder eine Methode, die wir als Argument an eine andere Funktion oder Methode übergeben, um eine Aktion durchzuführen. Einfach gesagt, Callback oder `VoidCallback` werden verwendet, wenn Daten von einer Methode an eine andere gesendet werden oder anders herum.

- 64-67. Je nachdem ob die Taschenlampe an oder aus ist, wird beim Tippen der Schalter der aktuelle Wert in den entgegengesetzten Wert geändert.

- 74-81. Das `Text`-Widget wird erstellt und gestaltet. `Style`-Eigenschaft ist für Gestaltung, `Color`-Eigenschaft ist für die Textfarbe, `fontSize` ist für die Textgrösse und `textAlign` ist für die Ausrichtung.

```
! pubspec.yaml X
C: > src > FlutterApps > Cocu > ! pubspec.yaml
1  name: flashlight_app
2  description: A new Flutter application.
3  version: 1.0.0+1
4
5  environment:
6    sdk: ">=2.7.0 <3.0.0"
7
8
9  dependencies:
10   flutter:
11     sdk: flutter
12
13   flashlight: ^0.1.1
14   flutter_automation: ^1.4.0
15
16  dev_dependencies:
17   flutter_test:
18     sdk: flutter
19
20  flutter:
21    assets:
22      - assets/images/1.png
23
24    fonts:
25      - family: Schyler
26        fonts:
27          - asset: fonts/Schyler-Regular.ttf
28          - asset: fonts/Schyler-Italic.ttf
29            style: italic
30
```

Abb. 1103: «pubspec.yaml» Datei der fertigen Taschenlampen App
Quelle: Anlehnung an Tiwari, 2021, Screenshot Visual Studio Code

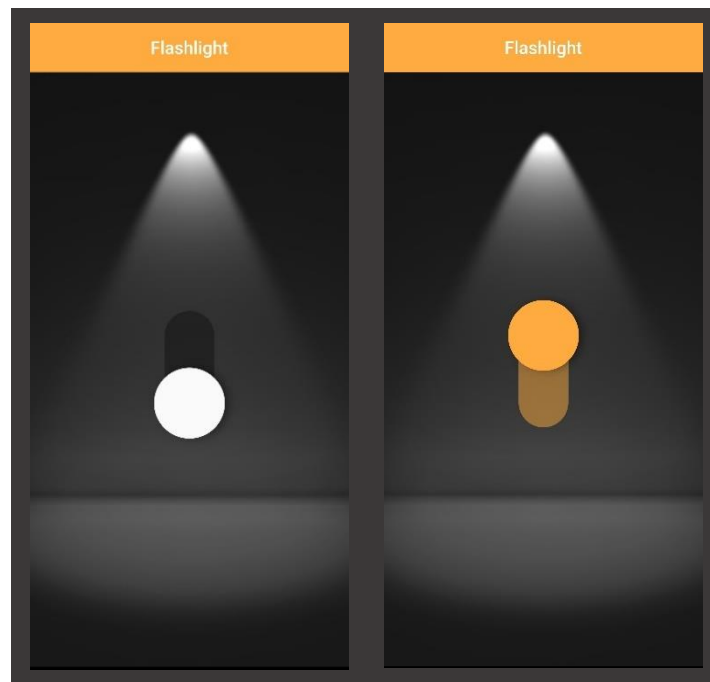


Abb. 1104: Ansicht der ersten Version der Taschenlampe App,
links: Lampe deaktiviert, rechts: Lampe aktiviert
Quelle: Eigene Darstellung, Screenshot Android Emulator

Nachdem die App im Android Emulator und auf verschiedenen Geräten, einem «OnePlus 7 Pro» mit einem Android Version 10 und einem «Samsung S7 Edge» mit Android Version 8, getestet wurde, konnte es für die Veröffentlichung der App für weitere Tester vorbereitet werden. Doch bevor die App veröffentlicht werden konnte, wurde ein App Icon erstellt. Dieser Schritt ist einer von mehreren, um eine App zu individualisieren. Das App Icon **Abb. 19** wurde mit dem kostenlosen online App Generator «Android Asset Studio» erstellt und in der App integriert. [30]



Abb. 1105: Icon der Taschenlampe App

Für die Veröffentlichung der Mobile App wurde im Internet recherchiert und ich bin zu folgenden Kenntnissen gekommen: Um Apps auf einem App Store zu publizieren, sind in der Regel ein Entwicklerkonto und ein Anmeldegebühr erforderlich. In **Tabelle 4** werden die beiden Plattformen «iOS» und «Android» verglichen.

Tabelle 4: App Publizierung bei iOS und Android Plattform im Vergleich

	iOS	Android
App Store	App Store/Apple Store	Google Play Store
Konto	Apple Developer Account	Google Developer Account

Gebühr	99 Dollar, jährlich	25 Dollar, einmalig
---------------	---------------------	---------------------

Quelle: Eigene Darstellung

Fürs Publizieren von Moblie Apps allgemein, darunter auch Flutter Apps, muss der/die Entwickler/in nachweisen, dass er oder sie der/die Eigentümer/in der jeweiligen App ist. Deshalb muss im Falle von Android das APK (Android application package) oder AAB (Android App Bundle) digital signiert werden, bevor sie hochgeladen werden kann. Jede App muss von einem sogenannten «Keystore», eine «jks»-Datei, digital signiert werden, bevor sie zu Google oder Apple hochgeladen wird. Dabei kann ein «Keystore» für mehrere Apps verwendet werden. Wenn neuere Versionen bereitgestellt werden, muss die digitale Signatur, um die Authentizität des «Uploads» zu überprüfen, verwendet werden. Wichtig ist, dass die «jks-Datei» nicht verloren geht, denn wenn dies geschieht, muss eine neue erstellt werden und bei Google sowie Apple überprüft wie auch genehmigt werden. Sie soll auch nicht in die Versionsverwaltung¹ gespeichert werden, denn wenn eine «Keystore-Datei» gestohlen wird oder in den Besitz von anderen kommt, kann die jeweilige Person die App mit einer völlig anderen Datei aktualisieren und fatale Probleme verursachen. Am besten wird eine Kopie davon an einem sicheren Ort bewahrt. Für die Erzeugung der «Keystore-Datei» ist ein «storePassword» (Speicherkenntwort) und ein «keyPassword» (Schlüsselkenntwort) erforderlich.

Die Schritte zur Erzeugung des Keystores sind für «Android» und «iOS» sehr ähnlich. Hier ein Beispiel für eine Android Version auf dem Windows Betriebssystem:

🖥 Folgenden Befehl im «Terminal» ausführen, «USER_NAME» und «APP_NAME» anpassen:

```
keytool -genkey -v -keystore C:/Users/"USER_NAME"/"APP_NAME".jks
-storetype JKS -keyalg RSA -keysize 2048 -validity 10000 -alias "APP_NAME".
```

🖥 Es wird nach einem «storePassword» und einem «keyPassword» gefragt.

- "Eigene eingeben"

🖥 Erstellen der Keystore-Eigenschaftendatei:

- Eine neue Datei «key.properties» im {project-root}/android/key.properties erstellen und mit folgendem Inhalt mit den Angaben aus dem vorherigen Schritt ersetzen und ergänzen. Der Speicherort kann beliebig gewählt werden.

```
storePassword=XXXXXXXX
keyPassword=XXXXXXXX
keyAlias=XXXXXXXX
storeFile= C:\Users\USER_NAME\XXXXX\"APP_NAME".jks
```

💀 Diese Datei soll NICHT in die Versionskontrolle eingefügt werden.

Der nächste Schritt nach der Erstellung eines «Keystore» ist, die «build.gradle»-Datei zu aktualisieren. Darin müssen die «compileSdkVersion», «minSdkVersion» (Mindest SDK Version) und «targetSdkVersion» (Ziel SDK Version) etc. definiert und kontrolliert werden. Genaue Dokumentation gibt es auf der Flutter Webseite. Dort wird dies Schritt für Schritt erklärt. [31]

¹ Versionsverwaltung oder Versionskontrolle ist ein System, das Änderungen an einer Datei oder einer Reihe von Dateien im Laufe der Zeit aufzeichnet, so dass Sie bestimmte Versionen später wieder aufrufen können. (Wikipedia, 2021)

Der folgende Schritt ist ein APK (Android application package) oder ein AAB (Android App Bundle) der App zu erstellen und zu Google Play hochladen. Dies kann auf diese Weise erreicht werden: In der Kommandozeile «projekt root» die Befehle ausführen, *“flutter clean”* und *“flutter build appbundle-release”*. Dieser Prozess ist je nach der Grösse der App unterschiedlich lang.

Wenn alle Schritte bearbeitet und gemacht worden sind, kann die App auf dem jeweiligen «App Store» ab sofort für Tester, aber erst für alle wenn es den Standards der Plattform entspricht und vom jeweiligen «App Store» genehmigt wird, veröffentlicht werden.

Aufgrund der hohen Gebühr und dem Fehlen eines Apple Geräts für die Entwicklung der Schritte des «iOS» Teils der Taschenlampen App ist die Wahl der Publizierung auf den «Google Play Store» gefallen. Die App wurde für eine kleine Gruppe von Testern freigegeben. Dies wurde mit der Erstellung eines Google Entwicklerkontos mit einem neuen «Google Konto» und einer neuen E-Mail-Adresse, welche auch für weitere Entwicklungen und Programmierzwecke gebraucht wird, bewerkstelligt.

3.5 Taschenlampen App Version Zwei




Um den zweiten Hürdenstein zu überwinden, wurde die Verwaltung des Projekts durch eine Software-Versionsverwaltung Plattform, die Individualisierung der Taschenlampen App und das Integrieren sowie Aufbewahren der Codes in einer «Code Repository» festgelegt. «Code Repository» (engl. repository, Lager, Quelle) ist, wie der Name schon sagt, ein Speicherort, in welchem Projekte oder Codes gespeichert und verwaltet werden. Mit ihrer Hilfe kann der Code online gespeichert und von mehreren Nutzern z.B. einem Entwicklerteam aus verschiedenen Geräten über dem Internet zugegriffen werden. Sie funktionieren ähnlich wie eine «Cloud», wie z.B. «Google Drive», «OneDrive» oder «iCloud», sind aber auf Software-Entwicklung und Versionsverwaltung spezialisiert. Zu den bekanntesten Versionsverwaltungsdiensten für Softwareentwicklungen gehören «GitHub», «Bitbucket» und «GitLab». In **Tabelle 5** werden sie miteinander verglichen.


Diese Plattformen sind auf «Git» basiert und teilen viele Eigenschaften. Zu den wichtigsten Funktionen bei «Git» gehören «git commit» sowie «push»(ziehen)- und «pull»(stossen)-Anfragen. «git commit» erzeugt einen Commit, der wie ein Schnappschuss des Repositories ist. «Commits» enthalten Informationen über Datei-Änderungen, die an der neu übertragenen Version vorgenommen wurden. «push» ist der Prozess der Übertragung von Dateien in das entfernte Repository. Er wird verwendet, um den «Commit», der in einem lokalen Branch auf Ihrem Computer vorgenommen wird, an ein entferntes Repository wie «GitHub» oder «GitLab» zu übertragen oder zu pushen. Mit «pull» Anfragen können z.B. Entwickler andere Entwickler über die Aktualisierungen der Codebasis informieren. [32, 33]

Für die Struktur einer Applikation ist es am optimalsten, dass nur die nötigen Elemente und Ordner im Projekt eingebaut und geordnet werden. Unbrauchbares und kommentierte Code-Zeilen führen zu vielen Codezeilen und erschweren die Lesbarkeit. Regelmässige «push» Anfragen und saubere Dokumentation der Prozesse bei den «Commit» Nachrichten

ermöglichen eine bessere Verwaltung von den Projekten und sichern das Speichern der Arbeit.

Tabelle 5: GitHub, GitLab und Bitbucket Plattformen im Vergleich

Plattform	GitHub	GitLab	Bitbucket
Kostenlose, private Repositories	 Haben keine solche Funktion für kostenlose, offene Repositories.	+	+
Open Source	- Ist nicht quelloffen. Die Benutzer hat nur begrenzte kostenlose Funktionen.	+	 Ist eine Open-Source-Plattform. Die Benutzer können alle Funktionen nutzen, ohne dafür bezahlen zu müssen.
Navigatorsche Benutzerfreundlichkeit	+	+	 Die Benutzerfreundlichkeit der Navigation ist in der GitHub-Plattform gegeben. Die Navigation ist benutzerfreundlich.
Integration von Drittanbieter-Tools	+	+	+

Projektana-lyse	 Besitzt keine solche Funktion, und die Funktionen sind für den Benutzer eingeschränkt.	+ Benutzer kann das Burndown-Diagramm ¹ und das Projektanalyse-diagramm auf der GitLab-Plattform sehen.	+ Benutzer können das Projektanalysedia-gramm und das Burndown-Diagramm auf der Bitbucket-Plattform anzeigen.
------------------------	---	---	--

Quelle: In Anlehnung an Gupta, o. D. und GeeksforGeeks, 2021

Um das Projekt und den Code zu sichern, immer auf dem neusten Stand zu sein und das Verlieren einer Datei oder grösseren Arbeitsaufwand zu vermeiden, macht es Sinn, einen Software-Entwicklungsdienst im Entwicklungsprozess zu integrieren und zu nutzen. Beim Vergleichen der Plattformen bietet «GitLab» mehr Möglichkeiten. Diese Plattform wurde wegen ihrer guten, kostenlosen und benutzerfreundlichen Oberfläche, ihren Eigenschaften sowie ihren Funktionen ausgewählt. Es wurde für GitLab ein Konto eingerichtet und in der Entwicklung der ersten mehr personalisierten Taschenlampen App integriert. Die Anweisungen auf der GitLab sind sehr ausführlich und einfach zu befolgen.

Da die erste Taschenlampen App-Version zum grössten Teil «copy-paste» und dabei fast kein Wissen über Dart und Flutter erforderlich war, wurde der Lernprozess der Dart Programmiersprache sowie dem Flutter-SDK erst für die Entwicklung der zweiten Taschenlampen App-Version geplant und mit ihm begonnen. Die Grundlagen von «Flutter» und «Dart» mussten erst erlernt und verstanden werden. Dies ist nicht sehr einfach. Die Mengen zu erlernender Informationen sind sehr gross. Für den Einstieg wurde zu Beginn ein Teil der Dokumentation «A tour of the Dart language» auf der Dart Webseite gelesen und durchgearbeitet. Als nächstes wurden Videos auf YouTube in einer Tutorial Playlist namens «Dart Programming Language Complete Tutorial 2021» von einem Softwareentwickler mit Namen «Cas Van Gool» auf dem Kanal «Flutter Expert» angeschaut und danach wurde mit dem «DartPad»-Tool wie auch diesen Videos gearbeitet. Weitere gute Tutorials über die «Dart» Programmiersprache sind auf der «Dart» Webseite zu finden. [22, 23, 24, 34, 35, 36, 37]

Der Vorgang des Lernprozesses war bei Flutter ähnlich wie bei Dart. Zuerst wurde die Dokumentation von «Flutter basics» durchgelesen und zum Teil Tutorials bearbeitet. Danach wurde die Playlist «Flutter Tutorial for Beginners» auf YouTube von einem Kanal aus dem Vereinigten Königreich namens «The Net Ninja» aufgerufen und später ein grosser Teil der Tutorials in «Android Studio» ausgeführt. Diese Tutorials waren auf Englisch. Eine gute Playlist auf Deutsch, ebenfalls auf YouTube, heisst «Wie du deine eigene App programmierst» und ist vom Kanal «STARTUP TEENS», welche von den Gründern der «Simpleclub» online Lernplattform, die ihre Mobile App «Simpleclub» auch mit Flutter entwickelt haben, erstellt wurde. Eine Strukturierung, Fragen und Notizen zu den recherchierten Themen wie Dart oder Flutter wurden in der Notizenapp «OneNote» festgehalten. [24, 38, 39, 40]

¹ Ein Burndown-Diagramm zeigt den verbleibenden Aufwand, der im Laufe der Zeit noch zu erledigend ist, an. (microTool, o. D.)

Die Personalisierung der Taschenlampen App basierte auf nur ein paar kleinen Elementen.

- ❖ In der «Appbar» wurden zwei Bilder hinzugefügt und der Titel, der Inhalt sowie die Gestaltung geändert.



Abb. 1106: «AppBar» Ansicht der zweiten Taschenlampenversion
Quelle: Eigene Darstellung, Screenshot Android Emulator

- ❖ Die Schalterfarbe ist auch bearbeitet worden. Es wurden ein neues «inactiveColor» (Schalterfarbe, wenn die Taschenlampe deaktiviert ist) und ein neues «activeColor» (Schalterfarbe, wenn die Taschenlampe aktive ist) hinzugefügt.
- ❖ Ein Knopf mit einer Schriftrolle wurde hinzugefügt. Er zeigt einen kleinen Abschnitt über die Erfindung der Taschenlampe und deren Erfinder, welcher von Wikipedia übernommen wurde (Wikipedia, 2021), an. (siehe **Abb. 1107**).



Abb. 1107: Ansicht der zweiten Version der Taschenlampe App,
links nach rechts: inaktiv, aktiv, Text eingeblendet.
Quelle: Eigene Darstellung, Screenshot Android Emulator

Die Implementierung des Interaktiven «History» Knopfes und das Layout der Widgets kann einige Schwierigkeiten bereiten. Das Einbauen von den anderen Elementen ist nicht sehr kompliziert.

Die «main.dart»-Datei in der ersten Appversion bestand aus 86 Codezeilen. Die zweite aus 356 Codezeilen. Das gesamte Projekt wurde auf «GitLab» gespeichert und verwaltet. Die

zweite Version der App wurde, wie auch die erste, im Android Emulator und auf zwei verschiedenen Geräten, einem «OnePlus 7 Pro» mit Android Version 10 und einem «Samsung S7 Edge» mit Android Version 8, getestet. Sie wurde nicht veröffentlicht oder für Tester freigegeben.

3.6 Design Mobile Apps

Nachdem die Grundprinzipien über Dart, Flutter und dem Entwicklungsprozess sowie der Veröffentlichung von Mobile Apps erlernt wurden, war es an der Zeit, mit der Design-Vorlage der E-Commerce App zu beginnen. Für das Design von UI und UX wurde wieder das Internet als Quelle gewählt. Bevor mit dem Design angefangen werden konnte, mussten die verschiedenen Konzepte von UI/UX Design und die Phasen des Designprozess recherchiert wie auch verstanden werden. Dabei tauchten viele Fragen auf: Was ist der Unterschied zwischen UI und UX Design? Was macht ein gutes Design aus? Wie muss man vorgehen? Wie kann ein flexibles Design erreicht werden?

Im Grunde lässt sich der Unterschied wie folgt erklären: Beim UX Design geht es um das Gefühl und die Erfahrung bei der Nutzung der App. Beim UI Design geht es um die Gestaltung der Benutzeroberfläche und die Interaktion mit der App. Das UI Design trägt zum Erfolg des UX Designs bei.

Ein grosser Teil der UX wird schon bei der Erstellung der ersten App Design Phase, der Erstellung des «Wireframes», gemacht. In dieser Phase wurde nur in schwarz-weiss geplant und die Informationsarchitektur sowie die Funktionsweise wie auch das Interaktionsdesign der App festgelegt.

- ♦ Eine Struktur für die App schaffen.
- ♦ Informationen sinnvoll, logisch und verständlich organisieren.
- ♦ Information leicht zugänglich und einfach machen.
- ♦ Funktionen logisch und effizient verteilen.
- ♦ Die Elemente wie Buttons, Scrolls sowie Navigationsleiste konsistent und vorhersehbar platzieren.
- ♦ Das Seitenlayout zweckmässig und nach Wichtigkeit strukturieren.
- ♦ Schnittstellen wie Navigationskomponente (Suchfeld, Schieberegler), Eingabesteuerung (Textfelder, Optionsfelder, Dropdown-Liste) einfach halten.
- ♦ Die räumlichen Beziehungen zwischen den Elementen auf der Seite berücksichtigen.

Bei Wireframes werden die Informationen strukturiert wie auch logisch und effizient verteilt. Dabei wird nach dem Motto «Einfach und Effizient» geplant.

In der zweiten Phase, der Erstellung von «Mockups», werden die Wireframes farbig. Die Elemente und Informationen werden gestaltet und es wird für das Erreichen der vorherigen gesetzten Zielen gesorgt. Hier wird nach dem Motto «Schlicht und Sauber» gearbeitet. Dies kann mit folgenden Schritten erreicht werden:

- ♦ Farbe und Textur strategisch einsetzen.
- ♦ Aufmerksamkeit auf Elemente lenken oder von ihnen ablenken.
- ♦ Farbe, Licht, Kontrast und Textur zum Vorteil nutzen.

- ♦ Mit Schatten und Kontrast dem Element eine räumliche Dimension geben.
- ♦ Farbvarianten von einem oder bis drei Grund-Farbtöne verwenden.
- ♦ Hierarchie und Klarheit mit Typografie schaffen.
- ♦ Schriftgrößen, Schriftarten und die Anordnung des Textes sorgfältig und für einfache Lesbarkeit einsetzen. [41]

Das Ziel der Gestaltung von Benutzeroberflächen ist es, die Interaktion des Benutzers so einfach und effizient wie möglich zu gestalten, um dessen Ziele zu erreichen. In der dritten Phase wird aus dem «Mockup» ein interaktiver Prototyp erstellt. Die Funktionalitäten der App und die meisten Interaktionen sind darin beinhaltet.

3.7 E-Commerce Appvorlage

3.7.1 E-Commerce Appvorlage Elemente und Funktionen

Für die Suche nach Inspiration und die Recherche für das Design der E-Commerce App wurde die Netzwerk Plattform für digitale Designer und Kreative, «dribbble», verwendet. Dribbble ist eine der größten Plattformen für Designer und dient als Plattform für Designportfolios. Die Webseite «Google Design» verfügt über zahlreiche Fallstudien und Anleitungen über Designs, welche sehr hilfreich waren. [42, 43]

Bei der Recherche über die Design Grundkonzepte, Analyse und Vergleich verschiedener E-Commerce Apps wie z.B. «IKEA», «Just Eat» und «Zalando» sowie Vergleichen vieler Design auf «dribbble», wurden viele Idee gestrichen und einige Funktionen neu durchdacht und definiert. In «dribbble» wurden verschiedene Sammlungen angelegt, welche nach den einzelnen Seiten der App unterteilt und geordnet wurden, wie z.B. «Homepage», «Profile» oder «Produkt-Information». Die Design-Vorlagen wurden analysiert, zusammen verglichen und anschliessend Elemente ausgewählt. Für die Erstellung der E-Commerce App wurden verschiedenen Kriterien berücksichtigt.

- ❖ Design-Prototyp-Vorlage muss über die elementaren Bausteine, Eigenschaften und Funktionen einer E-Commerce Applikation verfügen
- ❖ Es müssen die herkömmlichen Hauptseiten vorhanden sein
 - ↳ Registrierung/Anmeldeseite
 - ↳ Startseite (Homepage)
 - ↳ Suchseite (Discover)
 - ↳ Profileseite
 - ↳ Einstellungsseite
 - ↳ Bibliothek
- ❖ Der Design-Prototyp soll flexibel wie auch individuell anpassbar sein und anschliessend als Vorlage verschiedener E-Commerce Applikationen dienen.

In **Tabelle 6** ist eine Liste der Elemente und der Struktur, welche nach den Hauptseiten geordnet wurden, dargestellt.

Tabelle 6: Liste der Elemente nach Hauptseiten der App geordnet

Seite	Unterseiten	Elemente
Willkommen	Registrierung/Anmeldeseite	Hintergrund (Bild/Farbe) Logo (und Markenname) Slogan Start (Button)
Registrierung/Anmeldeseite	LOGIN SIGN UP	Login (Button) Sign in (Button) Textfelder (E-Mail + Passwort)
Startseite (Homepage)	Seite fürs Produkt (Produkt-Information-Seite) ❖ Anschluss zu Warenkorb ❖ Zahlungsprozess (Seiten)	Profilbild (oben) Suchleiste (oben) Filter (oben) Produkte (Scroll-Ansicht, horizontal und vertikal) Navigationsleiste (unten)
Profilseite	Liste (Unterseiten): *je nach App ❖ Konto ❖ Zahlungsmethoden ❖ Favoriten/Gespeichert ❖ ...	Profilbild und Name (oben) Logout Navigationsleiste (unten)
Suchseite (Discover)	TabBar: *je nach App ❖ Maps ❖ Suche (Produkte)	Suchleiste (oben) Filter (oben) Produkte (Scroll-Ansicht, horizontaler + vertikaler) Navigationsleiste (unten)
Einstellungsseite	Liste (Unterseiten), *je nach App ❖ Sprache	Suchleiste (oben) Navigationsleiste (unten)

	<ul style="list-style-type: none"> ❖ Benachrichtigung ❖ Support ❖ ... 	
Bibliothek	Liste (Unterseiten), *je nach App <ul style="list-style-type: none"> ❖ Bestellungen, Favoriten, ... ❖ Listen, *je nach App Kalender	Liste: (Schroll-Ansicht: horizontal) Liste: (Schroll-Ansicht: horizontal) Kalender (Monat/Wochen/Tags-Ansicht) Navigationsleiste (unten)
Produktinformati- onsseite	Produkt Bilder (oben) Name + Preis TabBar: Produkt- <ul style="list-style-type: none"> ❖ Beschreibung ❖ Material ❖ Information ❖ ... 	In Warenkorb (IconButton) Favorit (IconButton) Bilder (Scroll-Ansicht) TabBar: (Scroll-Ansicht, Tabs: horizontal, Inhalt: vertikal) Stepper (Anzahl Produkte wählen) Zu Warenkorb (Button) Navigationsleiste (unten)

Quelle: Eigene Darstellung

Um eine gute Struktur für die App zu erhalten, wurde die Informationsarchitektur sorgfältig geplant. Die Elemente wurden nach ihrer Wichtigkeit geordnet.

Die Interaktion und Funktionen sind logisch und effizient verteilt. Es wurden nur die notwendigen und wichtigen Funktionen aufgenommen und über einfache, benutzerfreundliche Schnittstellen wie die Navigationskomponente und die Eingabesteuerung zugänglich gemacht. Dies wurde durch Elemente wie die Navigationsleiste und die in der App verteilten Suchfelder erreicht. Dadurch wird der Zugang zu Informationen erleichtert und gewährleistet. Es wurde genug Raum für die Komponente geplant und Wert auf die räumlichen Beziehungen zwischen den Elementen gelegt. Mit genug Abständen zwischen den Elementen wurde der Weissraum einheitlich und gleichmässig verteilt (siehe **Abb. 22**).

Die Benennung der verschiedenen Elemente ist ein wichtiger Faktor, der zur Lesbarkeit und zum Verständnis für den Programmierer beiträgt.

3.7.2 E-Commerce Appvorlage Design

Als eine Struktur für das Design bereit war, wurde das Wireframe in «Adobe XD» erstellt. Adobe Experience Design ist eine vektorbasierte Grafiksoftware, die sich optimal zum Entwurf von grafischen Benutzeroberflächen für Mobile- und Web Apps eignet. Für die

Einführung in «Wireframing» in «Adobe XD» wurde das Tutorial «How to wireframe in Adobe XD» auf «Adobe» als Quelle gewählt. Ich ging diesen Anweisungen nach. [44]

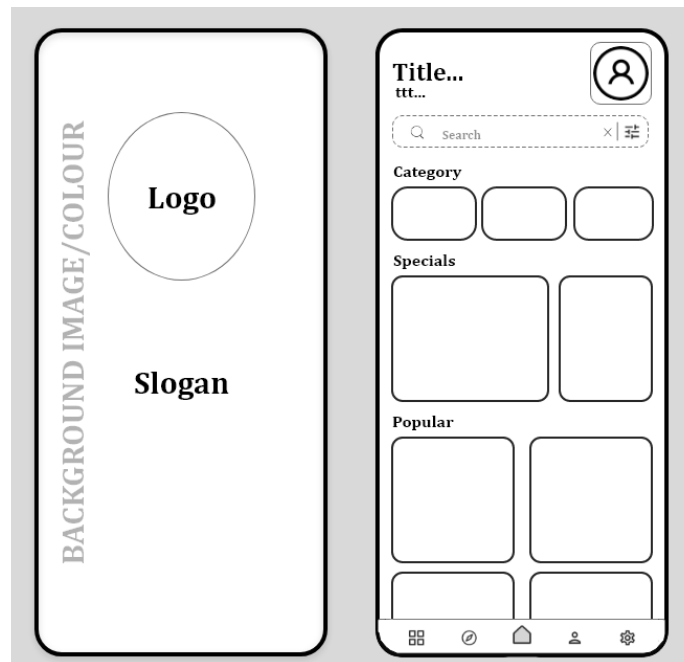


Abb. 1108: Wireframe, links: «Splashscreen», rechts: «Homepage»
Quelle: Eigene Darstellung, Adobe XD

Nach der Fertigstellung des Wireframes wurde es in ein Mockup eingebaut und weiterbearbeitet. Die Erstellung des Mockups wurde in «Framer» gemacht. Framer ist eine «No-Code-Plattform»¹, Das «All-In-One Design Tool» wurde mir von einem Kollegen «K» empfohlen. Im Gegensatz zu herkömmlichen Programmen können in Framer interaktive Designs mit Animationen erstellt werden. Framer bietet Einsteiger-Tutorials, wie z.B. «Learn the basics», an, mit deren Hilfe das Mockup erstellt wurde. [45]

Bei der Erstellung der Benutzeroberfläche wurden die erlernten Tipps und das gesammelte Wissen über Mobile App Design und UI Design angewendet. Die Aufmerksamkeit wurde durch Grösse und Farbe auf das wichtige Element gelenkt.

Durch Kontrast und Schatten können den Elementen Tiefe und Form verliehen werden, wie z.B. bei Schaltflächen und Behälter. Zusätzlich zu Farbe und Grösse wurde durch die Schriftgrössen sowie der Schriftform die visuelle Hierarchie verstärkt wie auch die Lesbarkeit erleichtert. Mit dem Einsetzen von zwei- oder drei Grund-Farbtönen wurde das Design schlicht und einfach gehalten. Error! Reference source not found. zeigt die Seite «Checkout, Order-Completed».

¹ No-Code-Entwicklungsplattformen (NCDPs) ermöglichen die Erstellung von Anwendungssoftwares über grafische Benutzeroberflächen bis hin zur Konfiguration ohne Programmierung. (Wikipedia, 2021)

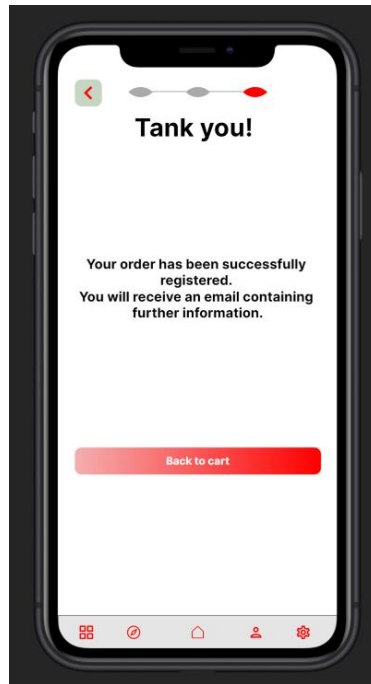


Abb. 1109: Mockup, «CheckoutOrderCompleted»

Quelle: Eigene Darstellung, Framer

3.7.3 E-Commerce Appvorlage Struktur und Aufbau

Anders als bei den zwei kleinen Taschenlampen Projekten handelt es sich bei der Appvorlage um ein grösseres Projekt. So müssen viele Aspekte berücksichtigt werden, wie z.B. die Projektstruktur, die Code-Lesbarkeit, die Benennungen der Komponenten, der Umgang mit Daten und Listen wie auch Parameter, Variablen und Eigenschaften sowie die effiziente Verwaltung und Aufbewahrung vom Code und vieles mehr.

Nachdem alle Vorbereitungsschritte abgeschlossen waren und eine Designvorlage festgelegt wurde, konnte die Programmierung beginnen. Die Verwandlung der Designvorlage in ein funktionierender Programmcode musste erst geplant werden. Die Elemente, denen das Design eine anpassbare Eigenschaft und Flexibilität verleihen sollte, mussten erst ausgewählt und definiert werden. Dabei wurden die folgenden Richtlinien definiert:

Komponentenvorlagen erstellen für:

- ❖ Seitenlayout
- ❖ Listen
- ❖ Schroll-Ansichten
- ❖ Knöpfe
- ❖ Icons
- ❖ Textfelder
- ❖ Formulare

Anpassbare Elemente und Eigenschaften:

- ❖ Form
 - ▽ Grösse
 - ▽ Kanten (Rahmenradius, Kantendicke)
 - ▽ Tiefe (Schatten)
 - ▽ Inhalt (Verschiedene Elemente wie Bilder oder Text)
- ❖ Farbe
 - Grund-Farbtöne
 - Formfarbe
 - Knöpffarbe
 - Formkanten Farbe
 - Schatten Farbe
- ❖ Typografie
 - ↯ Schriftart, Schriftgrösse, Schriftform etc.
- ❖ Anzahl Elemente
 - ≡ z.B. in Listen oder Schroll-Ansichten

Google stellt eine breite Anzahl an Bibliotheken und Werkzeugen für Design und App Entwicklung bereit. Zu den schon erwähnten Designsprachen «Google Design», Android Studio und Flutter, gehört auch «Material Design» dazu. Material Design ist eine Designsprache, die auf kartenähnlichen, materialartigen Flächen und gitterbasierten Layouts sowie Übergängen wie auch Animationen basiert ist. Der Gestaltungsstil von Material Design ist durch sein flaches Design und seinem Minimalismus bekannt.

Dabei ist «Material» ein anpassungsfähiges System von Richtlinien, Werkzeugen und Komponenten, welches die besten Praktiken für die Gestaltung von Benutzeroberflächen unterstützt. «Material» ist quellenfrei und bietet Designanleitungen und Code-Vorlagen für viele Grundbausteinelemente einer App, wie z.B. Navigationsleiste, Karten, Datentabellen, Formulare, Knöpfe, Textfelder und vieles mehr. Die Codevorlagen sind für die Plattformen Android, iOS, Web, aber auch Flutter geeignet. [46, 47]

Zusammen mit Anleitungen, Werkzeugen und Widgets von Flutter bilden «Material» Designanleitungen mit den Codevorlagen eine inhaltsreiche und gute Grundlage für die Programmierung und Entwicklung der E-Commerce App Komponentenvorlagen sowie die benutzerdefinierten Widgets in Flutter. Bevor mit der Programmierung angefangen wurde, wurden mehrere Vorbereitungen getroffen:

- ♦ Ein neues Repository auf GitLab für die Verwaltung und Synchronisierung der Programmcodes wurde erstellt.
- ♦ Ein neues Projekt in «Flutter» wurde erstellt.
- ♦ Eine Struktur für die verschiedenen Ordner und Dateien wurde definiert.

Die Ordnerstruktur ist in **Abb. 1110** abgebildet. Aus praktischen Gründen, und weil der Prototyp der App als Vorlage dienen soll, wurde die Namensgebung auf Englisch vorgenommen.

Aufbau und Ordnerstruktur der Appvorlage



Abb. 1110: Ordnerstruktur der Appvorlage (... = Ordner)

Quelle: Eigene Darstellung

¹ Theme (Themen) ist ein Bündel, welche Designelementen wie Layout, Typografie, Farben sowie Icons und Buttons enthält. (Ryte, o.D.)

Um der App eine Struktur zu verleihen, wurde für jede Seite (screen) ein Ordner erstellt, welcher folgendes enthält:

- ♦ Ein «..._screen.dart» (Seiten-Datei) -> für das Seitenlayout («AppBar», Körper («..._body.dart»)).
- ♦ Ein «..._body.dart» (Körper-Datei) -> für den Körper in «..._screen.dart»
- ♦ Ein «body» Ordner -> für die Komponenten und «Widgets», welche in «..._body.dart» verwendet werden.

Der Ordner «assets» enthält alle verwendeten Bilder, Icons und Schriftarten-Pakete.

Diese müssen in der «pubspec.yaml» Datei hinzugefügt werden, um gebraucht werden zu können (siehe **Abb. 17**).

Der Ordner «Komponente» enthält alle Vorlagen der individualisierten und selbst hergestellten «Widgets».

Der Ordner «config» enthält alle für die App nötigen und verwendeten Befehle wie:

- ♦ Themen (Regeln und Richtlinien über das Aussehen und die Gestaltung von Knöpfen oder Typografie)
- ♦ Datenmodelle (z.B. Benutzernamen).
- ♦ Farbpaletten

Der «test» Ordner ist für das Testen neuer «Widgets» und Projekte, welche von Tutorials abgeschrieben werden.

Diese Ordnung und Verteilung erleichterten das Programmieren allgemein und sind speziell für die Vereinfachung des Erstellens eines flexiblen Designs mit anpassbaren Elementen erstellt worden.

Um dies zu erklären, kann als Beispiel die Teilung eines Seitenordners wie der «Welcome» Ordner genommen werden. Der wird in drei Teile unterteilt. Eine «..._screen.dart» Datei, eine «..._body.dart» Datei und einen «body» Ordner. Da in Flutter alles ein «Widget» ist, kann ein separates «Widget» für das Seitenlayout «..._screen.dart» und ein separates «Widget» für den Körper «..._body.dart» erstellt werden.

Beim Trennen der Körper vom Seitenlayout wird der Code besser lesbar und weniger gestaut. Dazu, und weil der Körper im Seitenlayout als «Widget» übergeben wird, können verschiedene Körper in das Seitenlayout übergeben und somit verschiedene Designs ausprobiert wie auch implementiert werden.

Das gleiche Prinzip kann überall in der App verwendet werden auch für den Körper und seine Komponente. Der «body» Ordner enthält alle «Widgets», welche im Körper verwendet werden. Im Körper können demnach unterschiedliche «Widgets» integriert und ausgetauscht werden. Das trägt zu einem grossen Teil zur Verständlichkeit der Codes und der Behebung von Fehlern bei. Darüber hinaus werden die einzelnen Komponenten in mehreren, einfach zu programmierenden Teilen unterteilt. Auf diese Weise sind die verschiedenen Seiten sehr

flexibel und können leicht bearbeitet werden. In **Abb. 1112, 27** werden Beispiele mit Codes und Bildern abgebildet.

Auf der rechten Seite ist ein Beispiel einer Seite dargestellt. Die Seite besteht aus zwei Hauptkomponenten:

- ❖ Ein «AppBar» Widget, das ein Icon Widget und ein Text Widget enthält
- ❖ Ein «Column» (Säule) Widget, das drei Text Widgets enthält.

**(Die Untere Navigationsleiste ist vom Emulator)*

In **Abb. 1112, 27** werden sie mit einer schlechten Implementierung und einer guten Implementierung wiedergegeben.

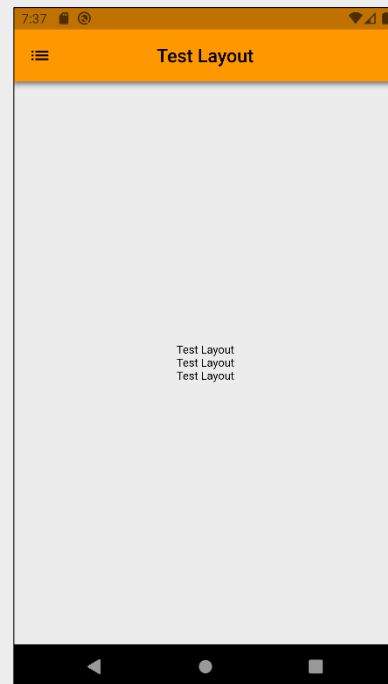


Abb. 1111: Rechts: Beispiel eines Seitenlayouts mit einem Titel in «AppBar» und ein Text im Körper
Quelle: Eigene Darstellung, Screenshot Android Emulator

```
lib > TEST > test_screen.dart > ...
1  import 'package:flutter/material.dart';
2
3  class TestScreen extends StatelessWidget {
4    const TestScreen({Key? key}) : super(key: key);
5
6    @override
7    Widget build(BuildContext context) {
8      return Scaffold(
9        appBar: AppBar(
10         leading: Icon(Icons.list),
11         title: Text("Schlechter Layout"),
12         centerTitle: true), // AppBar
13        body: Column(
14         mainAxisAlignment: MainAxisAlignment.center,
15         children: [
16           Center(
17             child: Column(
18               children: [
19                 Text("Schlechter Layout"),
20                 Text("Schlechter Layout"),
21                 Text("Schlechter Layout"),
22               ],
23             ), // Column
24         ], // Center
25       ), // Column
26     ); // Scaffold
27   }
28 }
29 }
```

Abb. 1112: Beispiel einer schlechter Code-Implementierung des Layouts aus Abb. 1111
Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android Emulator

In diesem Fall werden alle Komponenten in einem einzigen «TestScreen» Widget hinzugefügt. Insgesamt sind es 29 Codezeilen.

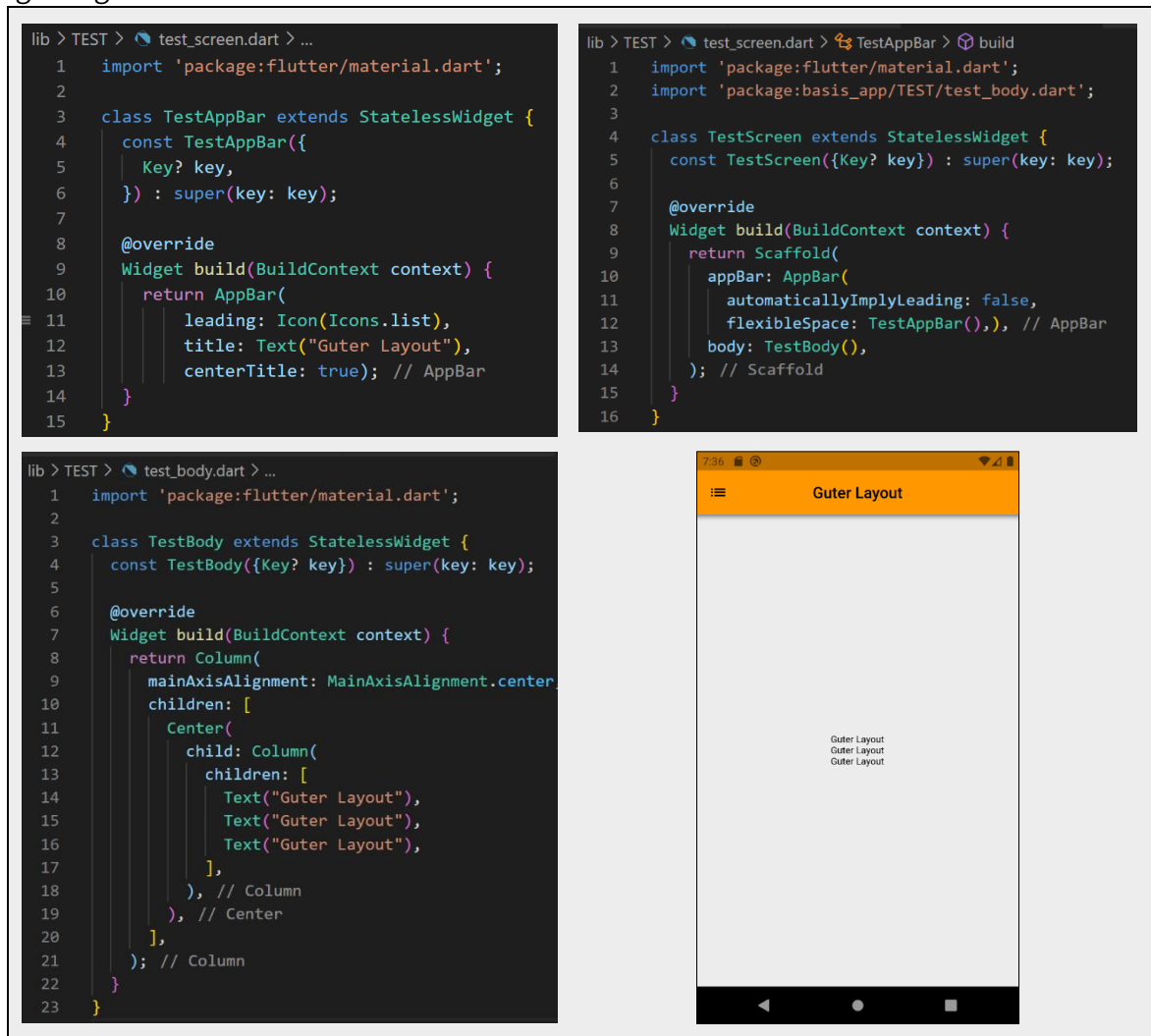


Abb. 1113: Beispiel einer guten Code-Implementierung des Layouts aus Abb. 1111
 Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android

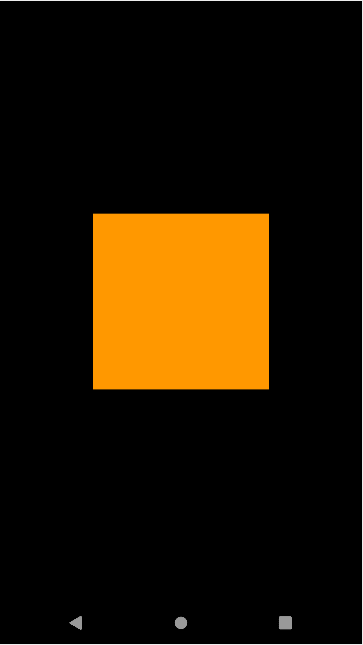
In diesem Fall werden die Seitenlayout, die «AppBar» und der Körper in drei separate «Widgets» geteilt. Insgesamt sind es 54 Codezeilen. Obwohl es in **Abb. 1113** fast doppelt so viele Codezeilen wie in **Abb. 1112** hat, trägt sie zu einer besseren Codelesbarkeit, einfacheren Bearbeitung und einem flexibleren Design bei. Das Seitenlayout besteht nun aus zwei Teilen, «AppBar» und Körper, die beliebig und sehr einfach verändert werden können. Ausserdem können die einzelnen Widgets, «TestAppBar» und «TestBody» in anderen Seitenlayouts sowie Widgets implementiert werden. Zudem ist die Anzahl der Codezeilen für das Seitenlayout von 29 auf 15 gesunken. Da es sich hier nur um eine Seite handelt, hat die erste Variante weniger Codes. Jedoch werden im Falle von mehreren Seiten und Widgets ein massiver Aufwand und unnötig viele Codes verhindert.

3.7.4 Anpassbare Komponentenvorlagen

Um leicht bearbeitbare Elemente zu erstellen, ihre Eigenschaften anzupassen und zu verändern, sollten die gewünschten Eigenschaften sehr leicht zugänglich sein. Um den Prozess der

Änderungen zu vereinfachen und zu verkürzen, sollten die Änderungen am besten nur an einer oder wenigen Stellen vorgenommen werden. Neue Komponenten und Widgets können in Flutter erstellt werden, diese lassen sich auch individualisieren und wiederverwenden. Die Eigenschaften der Widgets können mit Parametern bestimmt werden. Anstatt die Farbe des «Containers» im Widget selbst einzugeben, kann es als Parameter eingegeben und von außerhalb des Widgets festgelegt oder geändert werden. In **Abb. 1114**, **Abb. 1115**, **Abb. 1116**, **Abb. 1117** werden einige der Möglichkeiten mit und ohne der Übergabe von Parametern und Eigenschaften an Widgets in Flutter anhand eines «Container» dargestellt und erklärt.

```
lib > TEST > test_screen.dart > EigenschaftenAlsParamater > build
1  import 'package:flutter/material.dart';
2
3  class EigenschaftenAlsParamater extends StatelessWidget {
4    const EigenschaftenAlsParamater({Key? key}) : super(key: key);
5
6    @override
7    Widget build(BuildContext context) {
8      return Center(
9        child: Container(
10          height: 200,
11          width: 200,
12          color: Colors.orange,
13        ), // Container
14      ); // Center
15    }
16  }
```



Container ohne Paramater

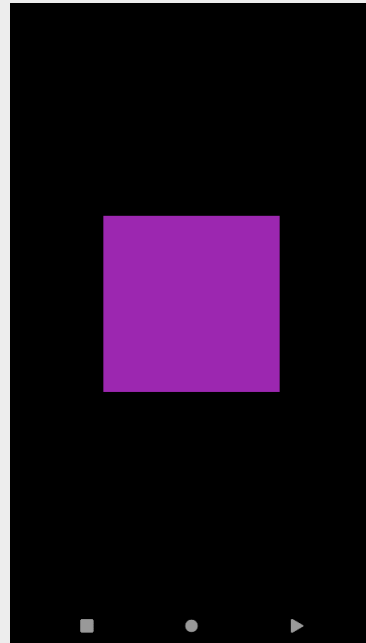
Die «Container» Farbe wird im «Container» selbst definiert. (Zeile 12).

*Abb. 1114: Beispiel eines «Container» Widgets ohne Übergabe von Parameter und Eigenschaften in Flutter
Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android Emulator*

```

lib > TEST > test_screen.dart > EigenschaftenAlsParamater
1  import 'package:flutter/material.dart';
2
3  const Color xFarbe = Colors.purple;
4
5  class EigenschaftenAlsParamater extends StatelessWidget {
6    const EigenschaftenAlsParamater({
7      Key? key,
8    }) : super(key: key);
9
10   @override
11   Widget build(BuildContext context) {
12     return Center(
13       child: Container(
14         height: 200,
15         width: 200,
16         color: xFarbe,
17       ), // Container
18     ); // Center
19   }
20 }

```



Container mit «lokalem» Parameter

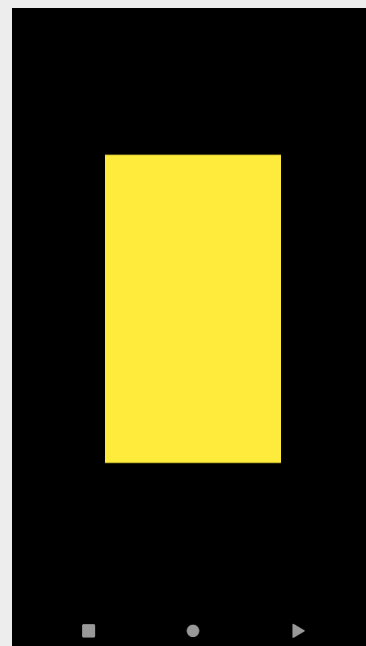
Die «Container» Farbe wird in derselben Datei definiert (xFarbe = Colors.purple) und der Eigenschaft «color» übergeben. (Zeilen: 3, 16).

Abb. 1115: Beispiel eines «Container» Widget mit Übergabe von Farbeigenschaft in Flutter
Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android Emulator

```

lib > TEST > test_screen.dart > ...
1  import 'package:flutter/material.dart';
2
3  const Color xFarbe = Colors.yellow;
4  const double xKastenhohe = 350;
5
6  class EigenschaftenAlsParamater extends StatelessWidget {
7    const EigenschaftenAlsParamater({
8      Key? key,
9    }) : super(key: key);
10
11   @override
12   Widget build(BuildContext context) {
13     return Center(
14       child: Container(
15         height: xKastenhohe,
16         width: 200,
17         color: xFarbe,
18       ), // Container
19     ); // Center
20   }
21 }

```



Container mit einem Farb- und Höhenparameter

Die «Container»-Farbe und -Höhe wird in derselben Datei definiert (xFarbe = Colors.purple) und der Eigenschaft «color» und «height» übergeben.

(Zeilen: 3, 4, 15, 17).

Abb. 1116: Beispiel eines «Container» Widget mit Übergabe von Farbeigenschaft und Größenparameter in Flutter
Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android Emulator

1.

```
lib > TEST > test_color_palette.dart > ...
1 import 'package:flutter/material.dart';
2
3 // Farben
4 const Color xHauptFarbton = Colors.blue;
5 const Color xContainerFarbe = Colors.green;
```

2.

```
lib > TEST > test_container_parameters.dart > ...
1 // Form: Höhe und Breite
2 const double xContainerHohe = 150;
3 const double xContainerBreite = 300;
```

3.

```
lib > TEST > test_screen.dart > ...
1 import 'package:basis_app/TEST/test_color_palette.dart';
2 import 'package:basis_app/TEST/test_container_parameters.dart';
3 import 'package:flutter/material.dart';
4
5 class EigenschaftenAlsParamater extends StatelessWidget {
6   const EigenschaftenAlsParamater({
7     Key? key,
8   }) : super(key: key);
9
10  @override
11  Widget build(BuildContext context) {
12    return Center(
13      child: Container(
14        height: xContainerHohe,
15        width: xContainerBreite,
16        color: xContainerFarbe,
17      ), // Container
18    ); // Center
19  }
20 }
```



«Container» mit Übergabe Farbeigenschaft und Gröszenparameter aus Datenmodellen in Flutter

Die «Container»-Farbe, -Höhe und Breite werden jeweils in einer separaten Datei definiert (1. und 2.).

Diese Dateien werden dann importiert (3. Zeilen: 2, 3) und der «Container»-Eigenschaft «color» und «height» übergeben. (Zeilen: 14, 15, 17).

Abb. 1117: Beispiel eines «Container» Widget mit Übergabe von Farbeigenschaft und Gröszenparameter aus Datenmodellen in Flutter

Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android Emulator

Im zweiten und dritten Beispiel (siehe **Abb. 1115** **Abb. 1116**) werden die Farbe und die Höhe des «Containers» in derselben Datei wie dem «Container» definiert. Diese Möglichkeit ist schnell umsetzbar und kann in kleinen, aber auch in grossen Projekten verwendet werden. So können die Parameter in derselben Datei gespeichert und keine weiteren Dateien importiert werden. Für ein grosses Projekt mit vielen Dateien, hunderten von Widgets und ihren Eigenschaften, wird es mühsam die Übersicht zu behalten. Wegen grösserem Aufwand wie auch langer Bearbeitungszeit ist es auch unpraktisch. Deshalb ist es besser, separate Datenmodelle zu erstellen und die Parameter an einem Ort zu speichern (siehe **Abb. 1117**). Auf diese Weise können Änderungen schnell und einfach vorgenommen werden.

Für den UI-Design-Prototyp wurden verschiedene Datenmodelle erstellt. In ihnen wurden Farben, Datenlisten, Abstände und viele andere Eigenschaften definiert, wie z.B. eine Farbpalette für die Grundfarbtöne, Schrift- und Knopffarben.

3.7.5 Erstellung eigener Widgets

Die Einsetzung von Parameter ist bei allen Widgets möglich. Da aber die Widgets unterschiedliche Eigenschaften haben und die Umsetzung für viele unterschiedlich gemacht werden, müssen immer wieder Informationen über die verschiedenen Widgets und ihren Eigenschaften recherchiert werden. Der offizielle YouTube Kanal von Flutter, «Flutter Community», ist dabei eine hilfreiche Quelle für die Vorstellung und Erklärung der Eigenschaften der Widgets gewesen.

Um für mehr Flexibilität des Designs zu sorgen, wurden benutzerdefinierte Widgets erstellt. Diese sollten eine Art Sammlung von Vorlagen werden, um mehr Auswahlmöglichkeiten für verschiedene App Design Versionen anzubieten. Damit sollte eine Breitenbande der verschiedenen E-Commerce App Designs, aber auch andere Apps allgemein gewährleistet sein. So können diese auch in anderen Projekten verwendet werden. Einige Beispiele sind: eine Suchleiste, eine Listen Scroll-Ansicht und das «CardTemplate» Widget. Die Übergabe von Parametern und Eigenschaften ist je nach Komponenten und Widgets anders oder begrenzt möglich. So musste bei den «Stateful Widgets» und den «Stateless Widgets» anders vorgegangen werden. Da die Listen Scroll-Ansicht ein starkes Merkmal der UI-Design-Vorlage ist und oft verwendet wird, wurde eine neues Widget dafür erstellt. Das Widget wurde wie folgt erstellt:

- ♦ Zuerst wurde eine neue Klasse erstellt und darin verschiedene Eigenschaften und Parameter wie Titel, Preis, Icon, Foto etc. definiert.
- ♦ Eine Liste dieser Klassenobjekte wurde erstellt und für die jeweiligen Parameter, Werte und Widgets übergeben.
- ♦ Diese Klasse wurde dann in ein Widget importiert, um ihre Eigenschaften nutzen zu können.
- ♦ Die Eigenschaften wurden danach in einem anderen Widget verwendet, um eine Vorlage zu erstellen.
- ♦ Eine leere Liste dieser Widget Vorlage wurde erstellt und mit einer Methode definiert, mit der genau so viele Widget erstellt werden, wie Klassenobjekte in der ersten Liste vorhanden sind. Die Methode wird bei der Initialisierung des Widget «CardScreenList» aufgeführt.
- ♦ Die Widgets in der neuen Liste werden mit einem List-Ansicht Konstruktor zu einer List Scroll-Ansicht umgeändert. In **Abb. 1118** **Abb. 1119** **Abb. 1120** sind die Schritte und der Code abgebildet.

```

lib > TEST > test_screen.dart > ...
1  import 'package:flutter/material.dart';
2  import 'package:basis_app/config/HomePage/animated_iconButton.dart';
3
4  class CardComponents {
5    CardComponents({
6      required this.title,
7      required this.secondaryText,
8      required this.tertiaryText,
9      required this.price,
10     required this.icon1,
11     required this.icon2,
12     required this.imageData,
13     required this.tabTitle,
14     required this.tabsecondaryText1,
15     required this.tabtertiaryText1,
16   });
17   final icon1;
18   final icon2;
19   final AssetImage imageData;
20   final String price;
21   final String secondaryText;
22   final String tertiaryText;
23   final String title;
24   final String tabTitle;
25   final String tabsecondaryText1;
26   final String tabtertiaryText1;
27 }
28
29 final List<CardComponents> allcardComponents = <CardComponents>[
30   CardComponents(
31     title: "title1 ",
32     secondaryText: "ttttttttt tttttt tttttt",
33     tertiaryText: "CHF",
34     price: '00.00',
35     icon1: FavoriteButton(
36       valueChanged: (_) {},
37     ), // FavoriteButton
38     icon2: ShoppingBagButton(
39       valueChanged: (_) {},
40     ), // ShoppingBagButton
41     imageData: AssetImage(
42       "lib/assets/images/Pic.png",
43     ), // AssetImage
44     tabTitle: 'tabTitle1',
45     tabsecondaryText1: 'tabsecondaryText1',
46     tabtertiaryText1: 'tabtertiaryText1',
47   ), // CardComponents
48 > CardComponents( // CardComponents ...
67 > CardComponents( // CardComponents ...
85 > CardComponents( // CardComponents ...
103 > CardComponents( // CardComponents ...
121 > CardComponents( // CardComponents ...
139 ]; // <CardComponents>[]

```

Neue Klasse «CardComponents» erstellt

Zeilen: 4 - 27

Verschiedene Eigenschaften und Parameter, wie Titel, Preis, Icon, Foto etc. definiert.

Zeilen: 6 - 26

Liste «allcardComponents» der Klassenobjekte «CardComponents» erstellt

Zeilen: 29 - 139

Für die jeweilige Parameter Werte und «Widgets» übergeben.

Z.B. Zeilen: 31 - 46

Abb. 1118: Erstellung einer benutzerdefinierten Listen Scroll-Ansicht Widget (Teil 1)
Quelle: Eigene Darstellung, Code: Screenshot Visual Studio

```

lib > TEST > test_screen.dart > ...
1 import 'package:basis_app/Screens/Homepage/SubjectOrProductPage/card_screen.dart';
2 import 'package:basis_app/Screens/Homepage/SubjectOrProductPage/class_card_components.dart';
3 import 'package:flutter/material.dart';
4
5 class CardTemplate extends StatelessWidget {
6   final CardComponents cComponents;
7
8   const CardTemplate({
9     Key? key,
10    required this.cComponents,
11  }) : super(key: key);
12
13  @override
14  Widget build(
15    BuildContext context,
16  ) {
17    return GestureDetector(
18      onTap: () {
19        Navigator.push(
20          context,
21          MaterialPageRoute(
22            builder: (context) => CardDetailsScreen(
23              cComponents: cComponents,
24            ), // CardDetailsScreen // MaterialPageRoute
25          ),
26        );
27      },
28      child: Container(
29        height: 200,
30        width: 160,
31        decoration: BoxDecoration( // BoxDecoration ...
32          child: Column(children: [
33            Expanded(flex: 4,
34              child: Container(
35                margin: EdgeInsets.only(bottom: 5),
36                decoration: BoxDecoration(
37                  borderRadius: BorderRadius.circular(10),
38                  border: Border.all(),
39                  image: DecorationImage(
40                    image: cComponents.imageData, fit: BoxFit.fill)), // DecorationI
41            Expanded(flex: 1,
42              child: Container(
43                padding: EdgeInsets.only(
44                  left: 8,
45                  right: 8), // EdgeInsets.only
46                child: Row(
47                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
48                  children: [Expanded(flex: 2,
49                    child: Container(
50                      alignment: Alignment.centerLeft,
51                      child: Text(
52                        cComponents.title,
53                        style: Theme.of(context).textTheme.subtitle2,
54                      ), // Text // Container // Expanded
55                    ), Expanded( // Expanded // Row // Container // Expanded ...
56                      // Expanded ...
57                    ), // Expanded ...
58                  ], // Column // Container // GestureDetector
59                ), // Column // Container // GestureDetector
60            ], // Column // Container // GestureDetector
61          ), // Column // Container // GestureDetector
62        ), // Column // Container // GestureDetector
63      ), // Column // Container // GestureDetector
64    ), // Column // Container // GestureDetector
65  );
66
116

```

Diese Klasse wurde dann in ein Widget «CardTemplate» importiert, um ihre Eigenschaften nutzen zu können.

Zeile: 2

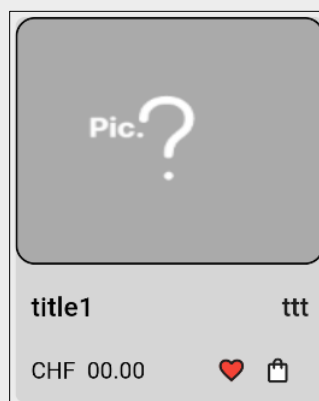
Eine Instanz der Klasse in dem Widget definiert. «CardComponents cComponents»

Zeilen: 6, 10

Auf diese Weise können die Eigenschaften dieser Klasse benutzt werden.

Wie z.B. «cComponents.imageData» und «cComponents.title»

Zeilen: 37, 49



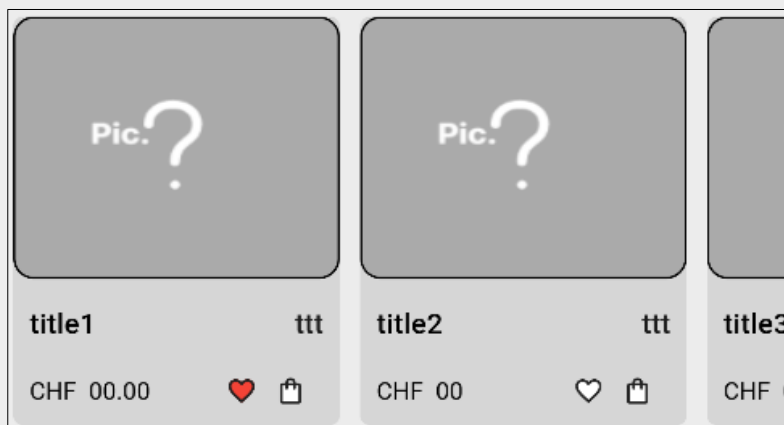
«CardTemplate» Widget

Abb. 1119: Erstellung einer benutzerdefinierten Listen Scroll-Ansicht Widget (Teil 2)
Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android Emulator


```

lib > TEST > test_screen.dart > ...
1 import 'package:basis_app/Screens/Homepage/SubjectOrProductPage/card_template.dart';
2 import 'package:basis_app/Screens/Homepage/SubjectOrProductPage/class_card_components.dart';
3 import 'package:flutter/material.dart';
4
5 class CardScreensList extends StatefulWidget {
6   const CardScreensList({Key? key}) : super(key: key);
7
8   @override
9   _CardScreensListState createState() => _CardScreensListState();
10 }
11
12 class _CardScreensListState extends State<CardScreensList> {
13   GlobalKey _listKeyCardScreensList = GlobalKey();
14   List<Widget> cardScreenComponents = [];
15
16   @override
17   void initState() {
18     super.initState();
19     _addCardComponents();
20   }
21
22   void _addCardComponents() {
23     allcardComponents.forEach((CardComponents cardComponents) {
24       cardScreenComponents.add(_buildScreens(cardComponents));
25     });
26   }
27
28   Widget _buildScreens(CardComponents cComponents) {
29     return CardTemplate(
30       cComponents: cComponents,
31     );
32   }
33
34   @override
35   Widget build(BuildContext context) {
36     return Container(
37       height: 200,
38       width: MediaQuery.of(context).size.width * 1,
39       child: ListView.separated(
40         separatorBuilder: (BuildContext context, int index) {
41           return SizedBox(
42             width: 10,
43           ); // SizedBox
44         },
45         scrollDirection: Axis.horizontal,
46         key: _listKeyCardScreensList,
47         itemCount: cardScreenComponents.length,
48         itemBuilder: (context, index) {
49           return cardScreenComponents[index];
50         }, // ListView.separated
51       ); // Container
52     ); // Container
53   }
54 }

```



«CardScreenList» «Widget»

Eine leere Liste dieser Widget-Vorlage ist erstellt

Zeile: 14

Eine Methode «_addCardComoponents» definiert, dass genau so viele Widgets erstellt werden, wie Klassenobjekte in der ersten Liste vorhanden sind.

Zeilen: 22 -26

Diese Methode wird bei der Initialisierung des Widgets «CardScreenList» aufgeführt

Zeile: 19

Die Widgets in der neuen Liste «cardScreenCompo-nents» werden mit einem «ListView.separated» Konstruktor zu einer List Scroll-Ansicht erstellt

Zeilen: 40 - 51

Scroll-Richtung ist horizontal

Zeile: 46

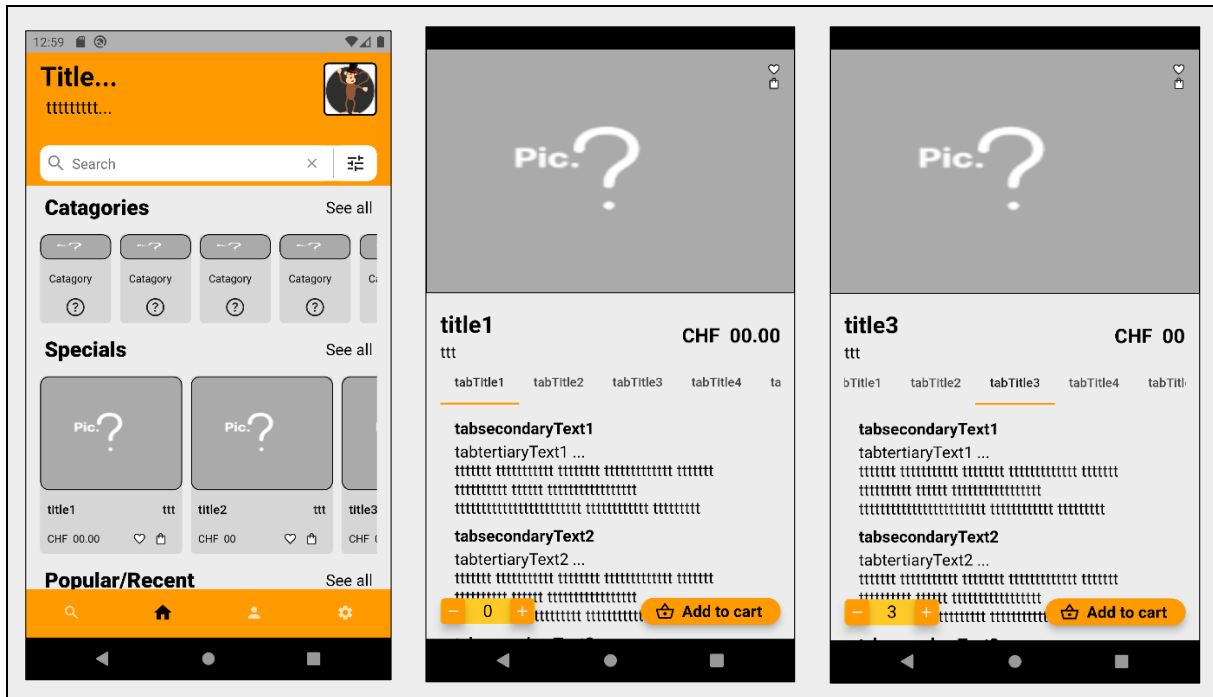
Listlänge es genau wie Anzahl «Widgets» in Liste «cardScreenCom-ponents»

Zeile: 48

Abb. 1120: Erstellung einer benutzerdefinierten Listen Scroll-Ansicht Widget (Teil 3)
Quelle: Eigene Darstellung, Code: Screenshot Visual Studio, Bild: Screenshot Android Emulator

Mit diesem Vorgang können Listen von Daten in Widgets aufgebaut und die Informationen via Eigenschaften an nächste Widget und Seiten weitergegeben werden. Dies wurde in der «CardScreenList» «Widget» eingesetzt und für jedes Klassenobjekt «CardComponents» in der «allcardComponents» Liste jeweils eine eigene «Produkt-Information-Seite» erstellt. Alle Eigenschaften können weitergegeben werden, darunter «title» oder «tabTitle1» (siehe **Abb. 1121**).

Die Suchleiste in der «Homepage» ist ebenfalls eine benutzerdefiniertes «Widget» und einer der Merkmaleigenschaften der UI Designvorlage (siehe **Abb. 1121**).



*Abb. 1121: Ansicht des Appvorlagen-Prototypen, links: «Homepage»,
Mitte und rechts: «Produkt-Information-Seite»,
Quelle: Eigene Darstellung, Screenshots Android Studio*

4 Ergebnisse

4.1 Taschenlampen Apps

Die erste Version der Taschenlampen App, in **Abb. 1104** abgebildet, wurde für Tester auf «Google Play Store» freigegeben. Die App besteht aus einem Schalter, mit dem die Taschenlampe des Geräts an- und ausgeschaltet werden kann. Bei allen anderen Tester läuft sie einwandfrei. Nur ein Fall wurde gemeldet, in der die App nicht auf die Taschenlampe zugreifen konnte.

Die zweite Version der Taschenlampe, in **Abb. 1107** abgebildet, basiert auf der ersten Version und besteht aus zusätzlichen Komponenten. Neben dem Schalter, welcher neu beim Aus- und Anschalten die Farbe ändert, zwei Bildern und einer neuen Schriftart für den Titel wurde ein Knopf hinzugefügt. Der Knopf zeigt durch Tippen einen kurzen Textabschnitt über die Erfindung der Taschenlampe. Der Code wurde in einem Code Repository in GitLab

aufbewahrt und verwaltet. Diese Version wurde nicht für Tester freigegeben, aber auf zwei privaten Geräten getestet. Sie funktioniert auf diesen beiden Geräten einwandfrei.

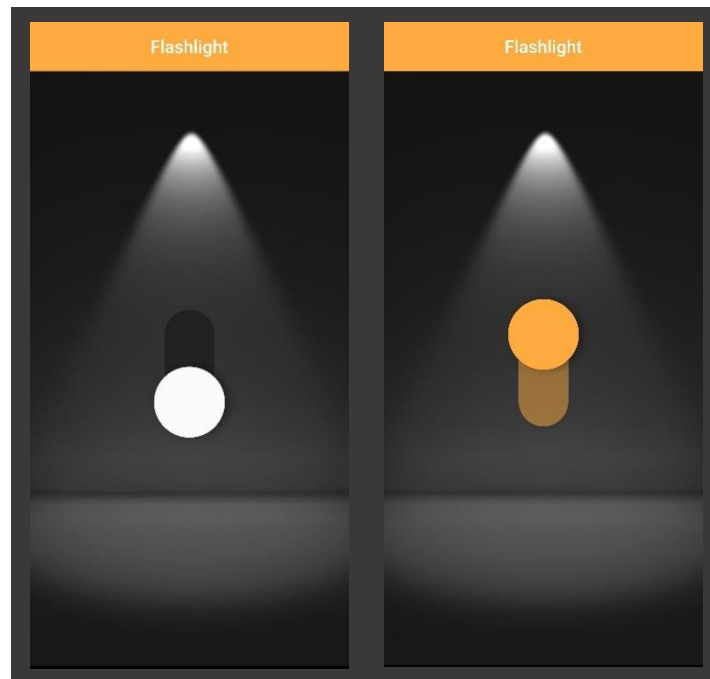


Abb. 18: Ansicht der ersten Version der Taschenlampe App,
links: Lampe deaktiviert, rechts: Lampe aktiviert
Quelle: Eigene Darstellung, Screenshot Android Emulator



Abb. 21: Ansicht der zweiten Version der Taschenlampe App,
links nach rechts: inaktiv, aktiv, Text eingeblendet.
Quelle: Eigene Darstellung, Screenshot Android Emulator

4.2 Appvorlage-Prototyp

4.2.1 Wireframe

Das Wireframe, in **Abb. 1122** abgebildet, ist nicht vollständig und enthält nicht alle App-Komponenten. Es wurden nur die Seiten «Homepage», «Profile», «Welcome» und «Authentication» vollständig entworfen. Mit dem «Wireframe» der App-Designvorlage ist eine gute Struktur für die App definiert. Die Informationen sind logisch geordnet und die Schnittstellen effizient wie auch benutzerfreundlich verteilt. Die Elemente haben genug Raum und die Abstände sind einheitlich sowie gleichmässig verteilt.

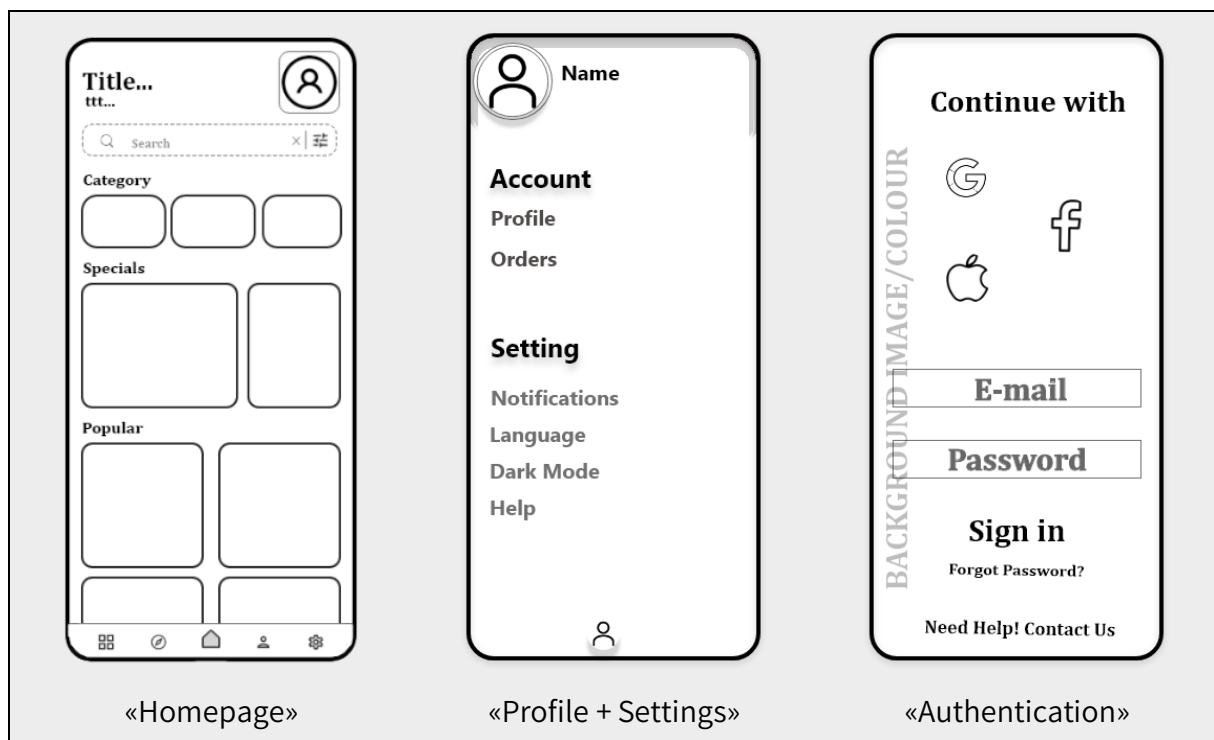
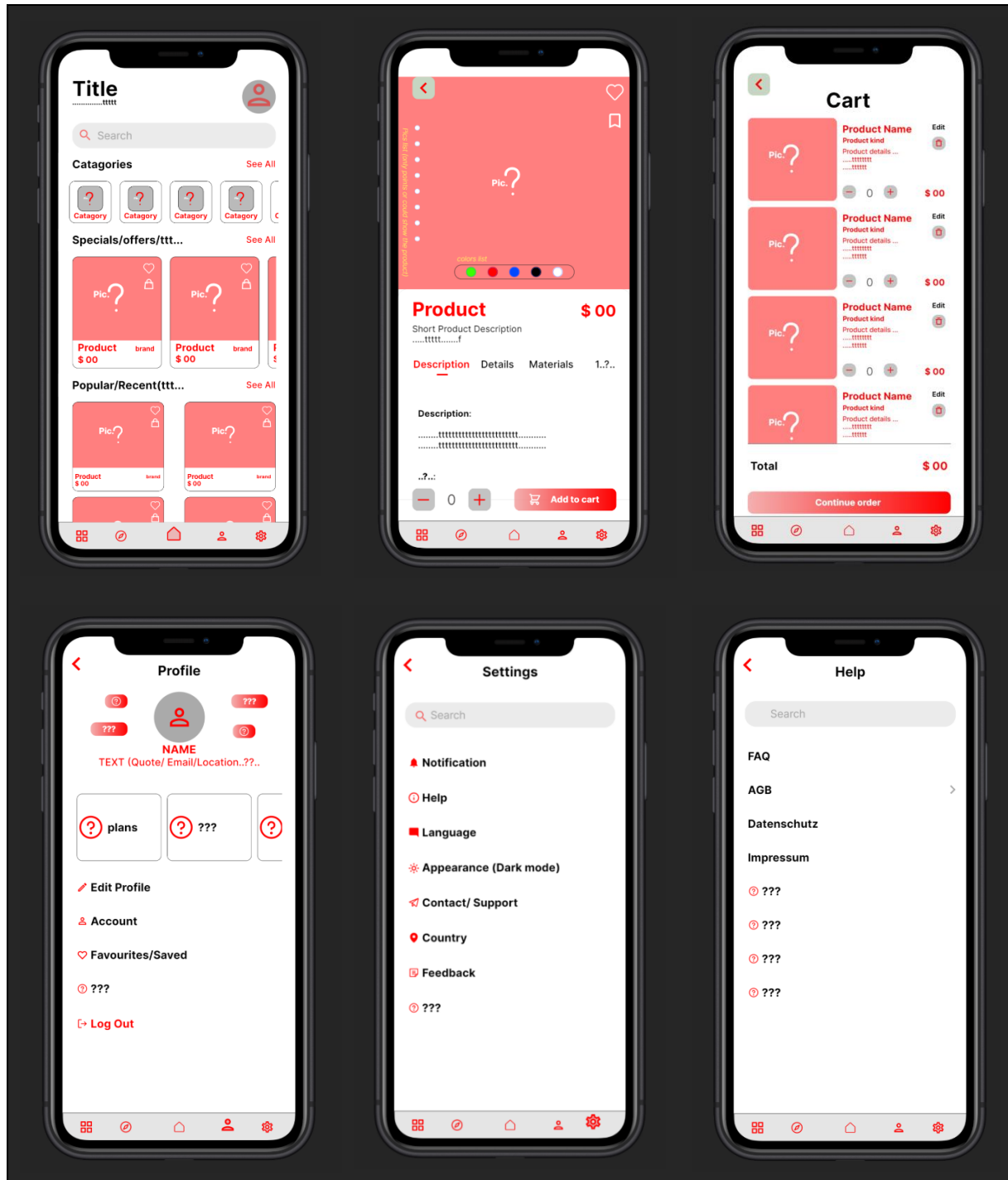


Abb. 1122: Wireframe Endversion, links-rechts: «Homepage», «Profile + Settings», «Authentication»
Quelle: Eigene Darstellung, Adobe XD

4.2.2 Mockup

Das Mockup basiert auf das Wireframe. Viele Elemente wurden aus dem unvollständigen Wireframe geändert, verbessert und es wurden neue hinzugefügt. Im «Framer» entstand ein hochwertiger, digitaler Entwurf der App (siehe **Abb. 1123**). Die Benutzeroberfläche wurde mit Farben, Typografie, Icons, etc. ergänzt. Die Aufmerksamkeit wurde mit Kontrast, Schatten und Farbe auf das wichtige Element gelenkt, damit es vom Unwichtigen ablenkt. Eine visuelle Hierarchie wurde mit dem Einsetzen verschiedener Schriftgrößen und Formgrößen erschaffen. Dank Framer Animationen und Verlinkungen können die Schnittstellen erst überprüft und danach verwendet werden. Das Mockup vermittelt ein gutes Nutzererlebnis und es kann einfach navigiert werden. Das leblose Wireframe wurde zu einer gestalterischen Fläche und einem interaktiven Design, welches sich optimal für Präsentationszwecke eignet. Auch bei dieser Phase wurden nicht alle Funktionen und Elemente vollständig entworfen. Die

Seiten «Discover» und «Bibliothek», in **Abb. 1124** abgebildet, wurden zwar erstellt, aber nicht vollständig mit Farben und interaktiven Schnittstellen ausgestattet.



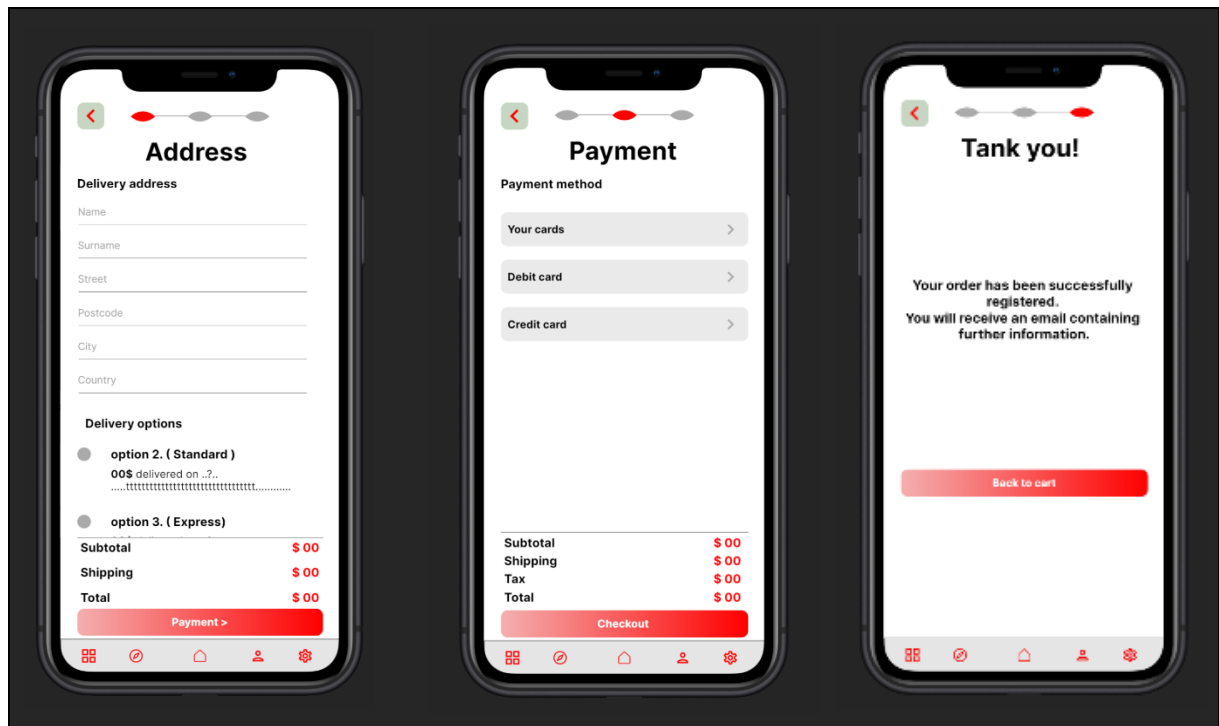


Abb. 1123: Mockup Endversion, von links oben nach rechts unten: «Homepage», «ProductScreen», «Cart», «Profile», «Settings», «Settings, Help», «Checkout, Information», «Checkout, Payment», «Checkout, OrderCompleted»
Quelle: Eigene Darstellung, Framer

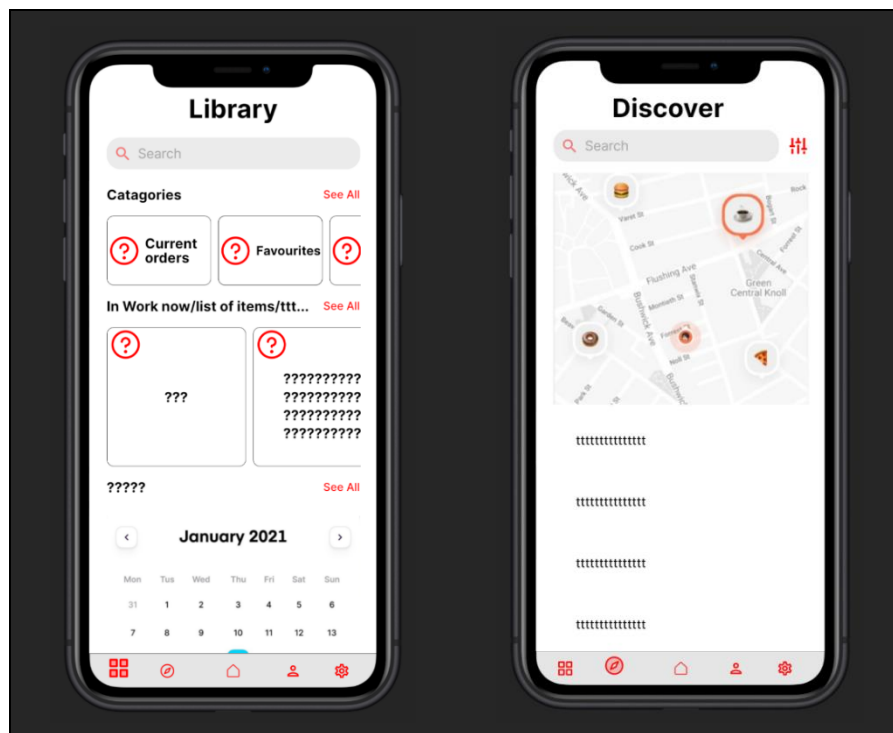


Abb. 1124: Mockup Endversion, links: «Library», rechts: «Discover»
Quelle: Eigene Darstellung, Framer

4.2.3 Prototyp

Bei der Entwicklung des Appvorlagen-Prototypen (siehe **Abb. 1121****Abb. 1125****Abb. 1126****Abb. 1127**) wurde der grösste Teil der Komponenten programmiert.

Die Appvorlage ist aber nicht vollständig. Sie enthält nicht alle Elemente der Designvorlage der E-Commerce App aus dem Mockup. Viele Komponenten sind neu und benutzerdefiniert komplett entwickelt, aber einige Funktionen und Komponenten sind unvollständig, wie z.B. die «Discover» (siehe **Abb. 1128**) oder «Authentication» Seiten. Ausserdem fehlen Funktionen wie der Zahlungsprozess oder die «Library» Seite.

Die strategische Strukturierung des Projektes in Flutter, die Übergabe von verschiedenen Eigenschaften als Parameter und die Erstellung neuer Widgets-Vorlagen für die verschiedenen App Komponenten machen die Appvorlage leicht bearbeitbar. Die Unterteilung der App-Seiten in die drei Teile, Seitenlayout, Körper und Körperkomponente, verleiht dem Quellcode eine klare Übersicht und trägt zur guten Lesbarkeit des Codes sowie zur schneller Bearbeitung von Fehlern bei.

Darüber hinaus können einzelne unterschiedliche Körper und Seitenlayouts miteinander kombiniert werden, was die Erstellung unterschiedlicher Vorlagen vereinfacht. Mit dem Erstellen von Datenmodellen, Listen, Farbpaletten und der Definition sowie Verwendung von Eigenschaften und Parametern wird es ermöglicht, in wenigen Schritten und Änderungen eine umfangreiche und sichtbare Veränderung des gesamten Designs zu erstellen. Mit benutzerdefinierten Widgets wurde die Erstellung spezifisch und für die gewünschten Komponenten des jeweiligen Designs leicht wie auch effizient gemacht. Diese Implementierungen und Praktiken kürzen den Prozess des Designs wie auch der Entwicklung und senkt den Aufwand massiv.

Obwohl die App nicht vollständig entwickelt ist und die Codeimplantierung nicht überall sauber wie auch effizient ist, wurden alle vorgenommenen Ziele und Grundsätze der Erstellung einer flexiblen sowie anpassungsfähigen Designvorlage erreicht. Sie wurde für keine Tester freigegeben. Der Code wurde zwar grösstenteils für die Android Plattform entwickelt, aber die Implementierung für die Android und iOS Plattform ist sehr ähnlich.

Mit der plattformübergreifenden Flutter Plattform und dem flexiblen Design soll die Erstellung neuer Mobile Apps mit einer vollständigen E-Commerce Appvorlage schnell und für einen breiten Nutzerkreis möglich werden. Mit ihrer Hilfe verläuft der Übergang von einer Idee oder einem Produkt zu einer App nahtlos und schnell.

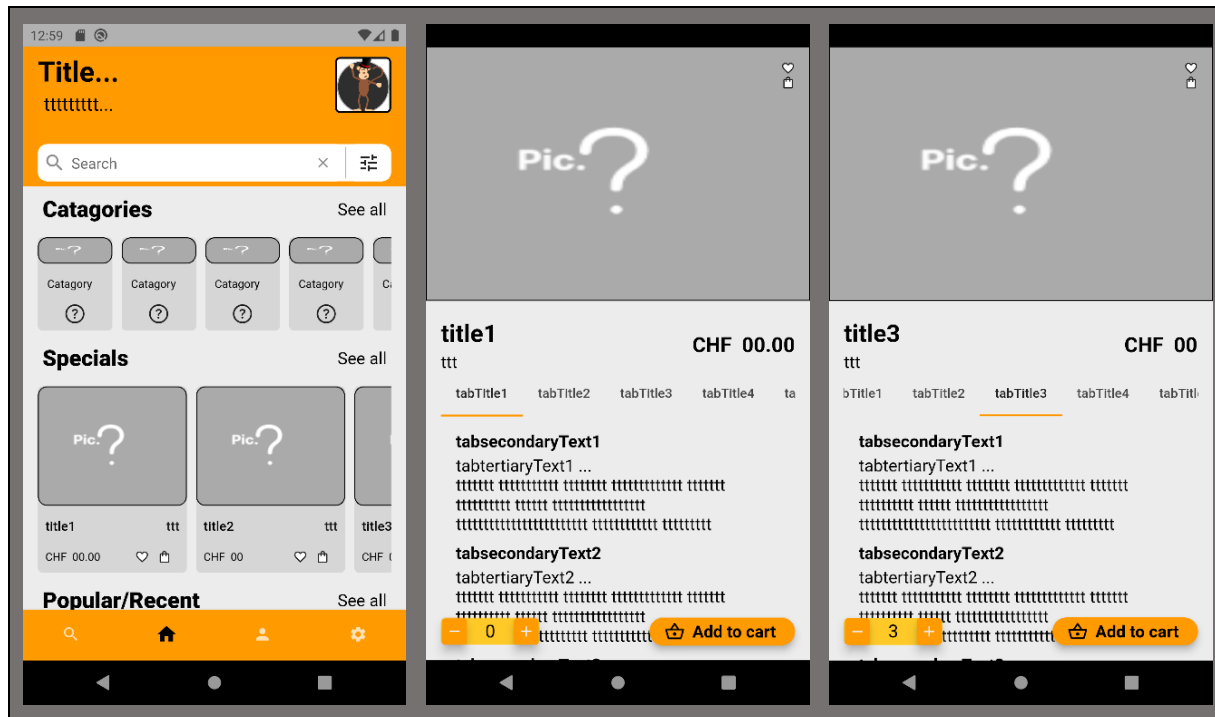


Abb. 35: Ansicht des Appvorlagen-Prototypen, links: «Homepage»,
Mitte und rechts: «Produkt-Information-Seite»,
Quelle: Eigene Darstellung, Screenshots Android Studio

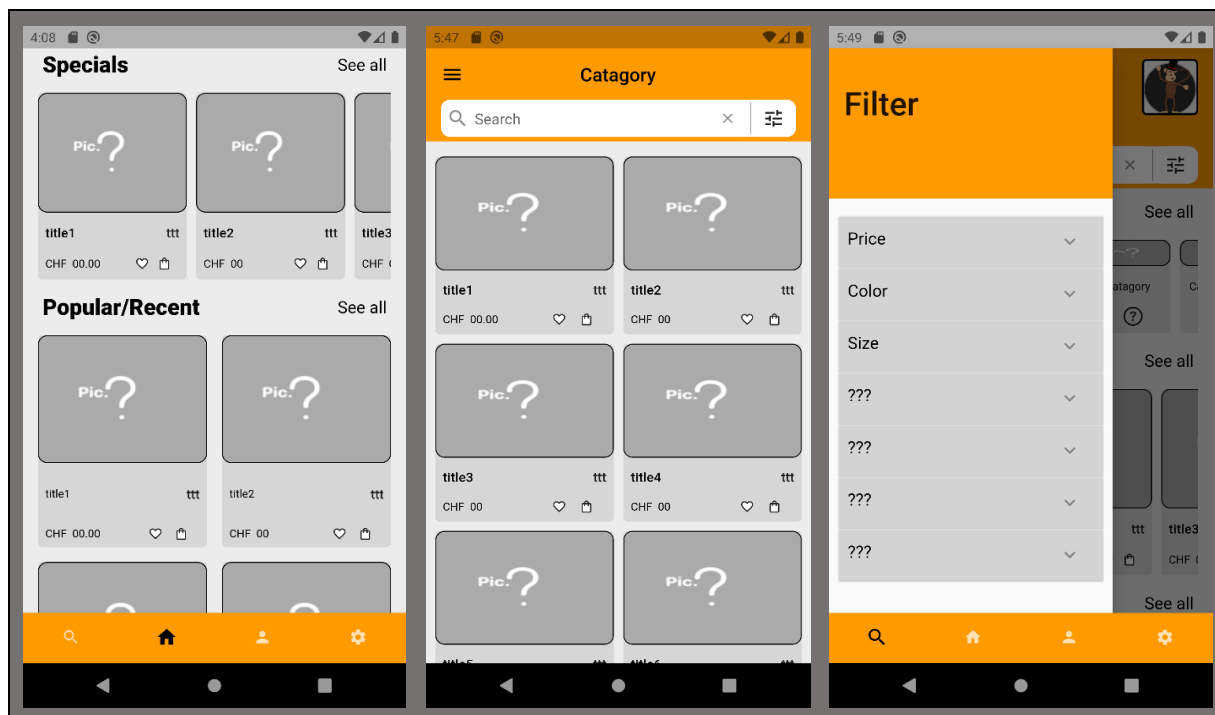


Abb. 1125: Appvorlage-Prototyp Endversion, Seiten: links-rechts:
«Homepage, Scroll» «CatagoryScreen», «FilterDrawer»
Quelle: Eigene Darstellung, Screenshots Android Studio

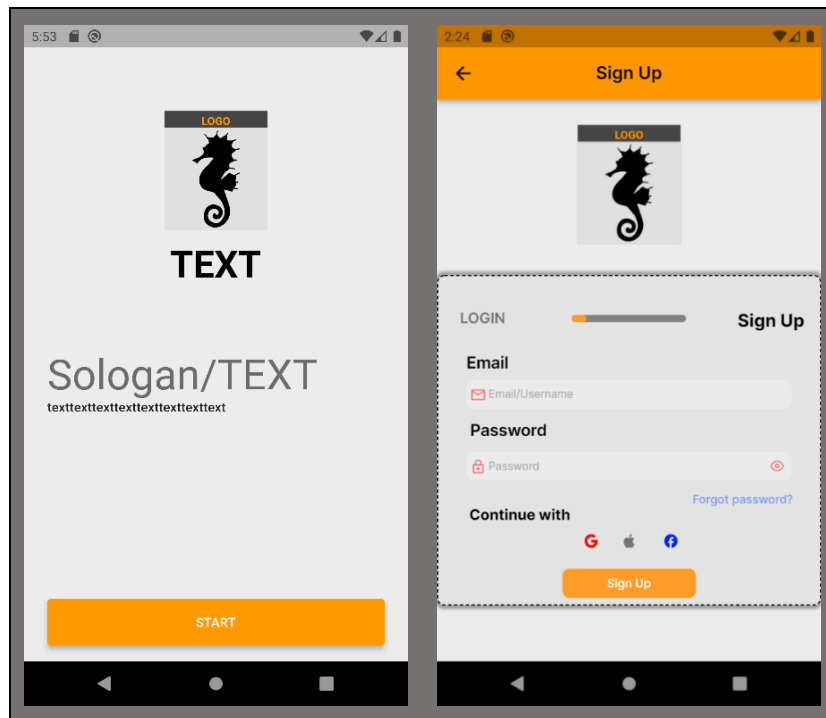


Abb. 1126: Appvorlage Prototyp Endversion, links-rechts:
 «Welcome», «Authentication», «CheckoutProcess»
 Quelle: Eigene Darstellung, Screenshots Android Studio

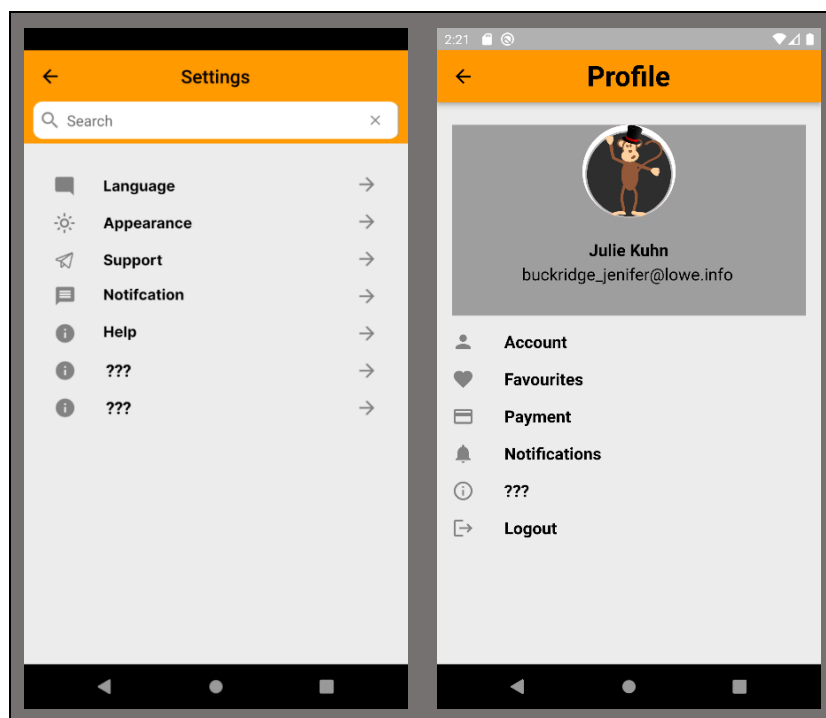
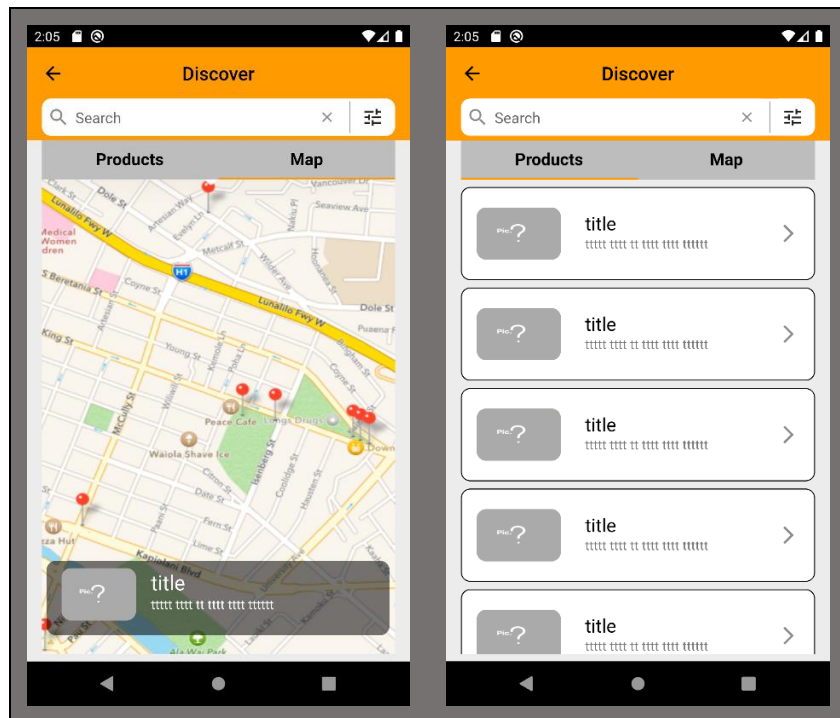


Abb. 1127: Appvorlage Prototyp Endversion, Seiten: links-rechts:
 «Settings», «FilterDrawer», «Profile»
 Quelle: Eigene Darstellung, Screenshots Android Studio



*Abb. 1128: Appvorlage Prototyp Endversion, Seiten: links-rechts:
«DiscoverMap», «FilterDrawer», «DiscoverProducts»
Quelle: Eigene Darstellung, Screenshots Android Studio*

5 Fazit

Die erste Version der Taschenlampen App ist gelungen, sie wurde erfolgreich erstellt. Für die Freigabe der App für Tester müsste der ganze Prozess von Grund auf gelernt werden. Da der erste Hürdenstein vom Hochladen der App auf einen App Store erreicht war und die App bei den meisten Testern funktionierte, wurde dem Problem mit dem Zugreifen auf die Taschenlampe des Gerätes bei einem Tester nicht nachgegangen und blieb ungelöst. Vermutlich lag das Problem an den unterschiedlichen Konfigurationen der verschiedenen Geräte und dem Kommunizieren der Software mit der Hardware.

Die vorgenommenen Ziele, die Integrierung einer Softwareentwicklung der Versionsverwaltungsdienste für die zweite Version und die Individualisierung der App, wurden erreicht. Die Integrierung der «Code Repository» brauchte mehrere Versuche, bis sie funktionierte. Die Testung der App erfolgte auf zwei Geräten, einen «OnePlus 7 Pro, Android Version 10» und einen «Samsung S7 Edge, Android Version 8». Die App ist bis jetzt auf der «OnePlus 7 Pro, Android Version 11» funktionsfähig.

Durch den gesamten Entwicklungsprozess, besonders bei der Erstellung der Wireframes und Mockups, mussten hunderte wenn nicht sogar tausende von Entscheidungen getroffen werden. Es musste über jede einzelne Kleinigkeit entschieden werden. Es tauchte eine Frage nach der anderen auf. Auch Sachen, die erst logisch und klar waren, mussten durchdacht und geplant werden. Dies führte manchmal zu einer Arbeitsunterbrechung von ein bis zwei Wochen. Damit können die unvollständigen Wireframe und Mockup Modelle erklärt werden. Dazu kann der Grund der Unvollständigkeit der gleichen Seiten, in denen sich das Wireframe,

das Mockup, aber auch die finalen Appvorlage befinden, mit der Tatsache, dass immer bei jeder Phase mit den gleichen Elementen und Seiten wie die «Homepage» Seite gearbeitet wird, erklärt werden. Dies führte zu ungleichmäßigem Fortschritt und Lösungen von Problemen. Auf diese Weise wurde nur ein Teil der Arbeit verbessert und schritt voran, während anderes unbearbeitet und am Ende unvollständig blieb.

Das Appvorlagen Projekt und dessen Code wurden durch klare Einteilungen von: individuellen Seiten, Ordern, Datenmodellen und Komponenten Dateien gut strukturiert. Jedoch müssen unbrauchbare Dateien teilweise gelöscht und neu sortiert werden. Dies könnte zukünftig besser gemacht werden, indem die unbrauchbaren Dateien sofort gelöscht werden und nicht unbearbeitet im Code bestehen bleiben. Das bedeutet auch, dass die Entwicklung von jedem Element zuerst abgeschlossen werden soll, bevor mit einer anderen begonnen wird. Es wurden sechs «commit» und «push» bei diesem Projekt erstellt. Dies sollte öfters gemacht werden, damit eine bessere Dokumentation der Entwicklung des Prozesses erreicht werden kann. In der gesamten Appvorlage wurde versucht, den Widgets, den Methoden und den Parametern logische wie auch passende Namen zu geben. Dies wurde aber nicht überall umgesetzt, da es manchmal schwierig war, gute Bezeichnungen zu finden. Immer passende Benennungen zu finden ist schwierig zu erreichen. Das Problem kann manchmal nicht völlig verhindert werden.

Die Arbeit mit Listen hat mir Schwierigkeiten bereitet, so wurden für die Erstellung der Listen Scroll-Ansicht Widgets «cardScreensList» und das Übergeben der Parameter und Eigenschaften zwei Tage investiert. Das Formatieren und Ordnen von Widgets wurde bei der Ansetzung bis hin zu den noch nicht verwendeten Widgets wie die «SliverAppBar» in der «Homepage» Seite schwierig und kompliziert. Dabei musste immer wieder die Eigenschaften und Umsetzung neuer Widgets gelernt werden. Die Implementierungsfehler der Widgets können nur vermieden werden, wenn man die vielen Eigenschaften der verschiedenen Flutter-Widgets kennt. Nicht jedes einzelne Widget hat völlig unterschiedliche Eigenschaften, aber viele von ihnen funktionieren nach dem gleichen Prinzip.

Der Prozess von der Widget-Gestaltung und -Bearbeitung kann verbessert werden. Dabei können im «Flutter» nicht nur einzelne Farbpaletten oder Parameterlisten, sondern ganze «Themes» (Themen) für Texte, Knöpfe, Farben, aber auch ganze «lightModes» und «darkModes» definiert werden. Dies kann mit der «ThemeData» Klasse erreicht werden. Mit dieser Klasse kann das gesamte visuelle Thema (Aussehen) für die «MaterialApp» (Designsprache) konfiguriert werden, indem ein «Theme» Widget am Anfang des «subtree» (Teilbaums) eingefügt wird. [48]

Dieser Klasse ist mir bekannt und wurde zum Teil auch in der Appvorlage verwendet. Jedoch muss noch einiges über sie gelernt werden, damit sie richtig und optimal verwendet wird.

Auch die Verwaltung von Daten und Zustände der Widgets können verbessert werden. Dabei kann das «Provider» Paket verwendet werden. «Provider» stellt im Grunde eine Hülle um die «InheritedWidgets» (geerbte Widgets) dar, welche die Verwendung und Verwaltung erleichtert. Es bietet eine Zustandsverwaltungstechnik, die für die Verwaltung von Daten rund um die App verwendet wird. Ich bin erst spät im Programmierprozess über dieses Paket aufmerksam geworden, deshalb habe ich sie in meiner Arbeit nicht angewendet. Diese kann in späteren Zeitpunkten oder Projekten gebraucht werden. [49]

Die Vorbereitung und Erarbeitung sehr grosser Mengen von neuen Informationen erschwerten den Einstieg. Schon bei der Installation von «Flutter» waren mehrere Versuche erforderlich, da es immer wieder zu Fehlermeldungen gekommen ist. Es wurden zwei Tage benötigt, um die Programmierumgebung einzurichten. Besonders die Einrichtung von «Android Studio», die Installationen erforderlicher «SDKs» und die Verwaltung der Umgebungsvariablen auf dem Laptop verursachten grosse Schwierigkeiten.

Der Prozess vom Erlernen der «Dart» Programmiersprache und das «Flutter» Framework war inkonsequent. Es wurden mehrere Tutorials angeschaut und meistens nur bis zur Hälfte erarbeitet. Dabei wurde immer wieder an anderen Projekten gearbeitet und das Wichtige vergessen, obwohl die anderen Projekte zum Erlernen und für die Anwendung des eigentlichen Projektes gedacht waren. Eine bessere Vorgehensweise wäre, bestimmte Tutorials auszuwählen und deren Aufgaben vollständig abzuschliessen, bevor mit anderen Projekten begonnen wird.

Während der Entwicklung mussten auch technische Hürden überwunden werden. Nach einem Windows-Update traten plötzlich Probleme auf, die die Verwendung des Emulators verhinderten. Dieses Problem wurde durch den Anschluss eines Android-Smartphones gelöst, das als Emulator diente. In einem anderen Fall gab es Probleme mit der Mausverbindung, die mir zuerst nicht bewusst waren. Da sich die Maus sehr langsam bewegte, schien es, als liege es am Laptop. Um dieses Problem zu lösen, tüftelte ich herum, deinstallierte Apps, führte Virentests durch, bis ich nach zwei Tagen die Maus über den Adapter und nicht mehr über "Bluetooth" anschloss und das Problem somit löste

Mit der Programmierung der Appvorlage wurde teilweise bereits vor dem Festlegen eines Designs begonnen. Dieses Vorgehen, ohne ein visuelles Modell vor Augen und einen klaren Plan im Kopf zu haben, ist nicht effizient. Es führt womöglich sogar zur Verzögerung des Fortschritts. Dies wurde erst nach der Erstellung einer einigermaßen vollständigen Designvorlage sehr deutlich.

Einer der grossen Fehler in der Entwicklung war, dass zu wenige Pausen geplant wurden. Es wurde manchmal fünf bis sechs Stunden ohne Unterbrechung am Bildschirm gesessen. Dazu wird auch am Gymnasium Thun digital unterrichtet. So wurde zum Teil elf bis zwölf Stunden am Computer gearbeitet, was sehr ungesund ist. Das hängt auch mit meinem Zeitmanagement zusammen, denn ein anderer Fehler war, die Arbeit bis zu den letzten Wochen vor dem Abgabetermin zu verschieben. Aus Angst vor dem riesigem Aufwand, der Recherche und der Entwicklung mussten für die Erstellung dieser Arbeit Nächte durchgearbeitet werden. Die Programmierung der meisten Codes für die Appvorlage wurden in etwa drei Wochen angefertigt. Die Erstellung des Mockups sowie das Schreiben der Arbeit wurden jeweils in einer Woche gemacht. Ein besseres Zeitmanagement ist also einer der grössten Faktoren, die das Resultat dieses Projekts günstig beeinflussen würden. Dies ist ein wichtiger Punkt für die Planung zukünftiger Projekte und Arbeiten. Bei der Zeitmanagement besteht definitiv einen Verbesserungspotenzial. Die einzelnen Schritte sollten besser eingeteilt werden. Am besten wäre es, die Designplanung im Voraus vollständig abgeschlossen zu haben und erst dann mit dem Programmieren anfangen. Für das Schreiben des Werkberichtes sollte auf jeden Fall viel Zeit eingeplant werden und nicht alles bis auf den letzten Drücker hinauszögern. Einen

Monat Zeit während den Ferien einzuplanen, reicht auf keinen Fall für die gesamte Entwicklung einer solchen App und dem zusätzlichen Schreiben der Arbeit.

Die Motivation und der Anschub für die Weiterführung der Entwicklung trotz enormen Druckes und Stress sind durch die Teilnahme an einer Informatik Studienwoche entstanden. In dieser Studienwoche wurden mit anderen Schülern sowie Informatik-Enthusiasten Ideen ausgetauscht und dabei viel Neues erlernt, was zur Erarbeitung dieses Projektes beigetragen hatte. Es wäre sicherlich klug gewesen, mehr Menschen in meinem Umfeld um Hilfe zu bitten, aber es muss auch berücksichtigt werden, dass ich nur wenige Personen in meinem Umfeld kenne, welche sich mit dem Thema der Arbeit auskennen.

Trotz des grossen Aufwands und den vielen Problemen bin ich mit dem Ergebnis der Arbeit zufrieden. Verbesserungspotential existiert, sowohl bei der Programmierung als auch bei der Entwicklung.

Wenn man die Komplexität des gesetzten Zieles betrachtet und die Ausgangslage mit den geringen Programmierkenntnissen und den fehlenden Vorkenntnissen im Mobile App Design und dessen Entwicklung, ist meiner Meinung nach ein gutes Produkt entstanden. Um eine App zu entwickeln, werden normalerweise mindestens fünf Personen bei einer Firma daran arbeiten müssen. Bei grösseren Produkten und Firmen sind es viel mehr.

Die App ist zwar nicht vollständig, aber viele der gesetzten Ziele wurden erreicht.

Mehrere benutzerdefinierte und anpassbare Widgets wurden erstellt, jedoch ist ihrer Anzahl noch klein und kann mit der Zeit und der Entwicklung verschieden «E-Commerce» oder allgemein Appvarianten und Vorlagen, zunehmen.

Mit der Verwendung von Parametern und dem Erstellen von neuen benutzerdefinierten Elementen sowie die Wiederverwendbarkeit der Widgets kann mit einfachen Änderungen die gesamte Designvorlage bearbeitet und geändert werden. Durch einfache Schritte können die Eigenschaften der Widgets, wie ihre Farbe, Schriftart, die Kantenform, umgewandelt und nach Wunsch gestaltet werden. Dies alles führt zu einem sehr flexiblen, simplen und nahezu grenzenlosen Design.

Dieses Produkt agiert in der jetzigen Verfassung den Transfer zwischen einer Idee und einer App, einfach und schnell. Wenn es vollständig ist, hat es das Potenzial, sehr hochwertig zu sein.

6 Quellenverzeichnis

6.1 Internetquellenverzeichnis

- [1] Tenzer, F. (5. Oktober 2021). Statista. (Anzahl der Smartphone-Nutzer weltweit von 2016 bis 2020 und Prognose bis 2024) Abgerufen am 19. Oktober 2021 von <https://de.statista.com/statistik/daten/studie/309656/umfrage/prognose-zur-anzahl-der-smartphone-nutzer-weltweit/>
- [2] Wikipedia. (1. August 2021). Software. Abgerufen am 17. Oktober 2021 von [https://de.wikipedia.org/wiki/Software#:~:text=Software%20%5B%CB%88s%C9%92f\(t\)w%C9%9B%CB%90,Programme%20und%20die%20zugeh%C3%B6rigen%20Daten.](https://de.wikipedia.org/wiki/Software#:~:text=Software%20%5B%CB%88s%C9%92f(t)w%C9%9B%CB%90,Programme%20und%20die%20zugeh%C3%B6rigen%20Daten.)
- [3] Wirtschaftslexikon24. (2020). Systemsoftware. Abgerufen am 17. Oktober 2021 von <http://www.wirtschaftslexikon24.com/d/systemsoftware/systemsoftware.htm>
- [4] ITWissen. (2021). Systemsoftware. Abgerufen am 17. Oktober 2021 von <https://www.itwissen.info/Systemsoftware-system-software.html>
- [5] TenMedia. (2021). Softwareentwicklung. Abgerufen am 17. Oktober 2021 von <https://www.tenmedia.de/de/glossar/softwareentwicklung>
- [6] Klicksafe. (o. D.). Eine App - Was ist das? Abgerufen am 17. Oktober 2021 von <https://www.klicksafe.de/apps/eine-app-was-ist-das/>
- [7] t2informatik. (o. D.). Was ist Wireframing? Abgerufen am 17. Oktober 2021 von <https://t2informatik.de/wissen-kompakt/wireframing/>
- [8] Wikipedia. (27. Februar 2020). Abgerufen am 17. Oktober 2021 von <https://de.wikipedia.org/wiki/Drahtgittermodell>
- [9] Wikipedia. (25. Februar 2021). Vorführmodell. Abgerufen am 19. Oktober 2021 von <https://de.wikipedia.org/wiki/Vorf%C3%BChrmodell>
- [10] Ryte. (o. D.). Ryte Wiki. Mockup. Abgerufen am 17. Oktober 2021 von <https://de.ryte.com/wiki/Mockup>
- [11] Wikipedia. (23. Januar 2021). Entwicklungsstadium (Software). Abgerufen am 17. Oktober 2021 von [https://de.wikipedia.org/wiki/Entwicklungsstadium_\(Software\)](https://de.wikipedia.org/wiki/Entwicklungsstadium_(Software))
- [12] t2informatik. (o. D.). Was ist eine Beta-Version? Abgerufen am 17. Oktober 2021 von <https://t2informatik.de/wissen-kompakt/beta-version/>
- [13] reinstil. (2019). Die Phasen einer App-Entwicklung. Abgerufen am 17. Oktober 2021 von <https://www.digitalagentur-mainz.de/blog/allgemein/die-phasen-einer-app-entwicklung/>
- [14] Schiessel, T. (2020). Mission Mobile. 1, 2 oder 3? Die verschiedenen Varianten der Mobile Apps. Abgerufen am 17. Oktober 2021 von <https://mission-mobile.de/app-entwicklung/app-design/1-2-oder-3-die-verschiedenen-varianten-der-mobile-apps/>

- [15] brightsolutions. (o. D.). App entwickeln: Native App vs Hybride App, Web App & PWA. Abgerufen am 17. Oktober 2021 von <https://www.brightsolutions.de/blog/native-vs-hybride-vs-web-app/>
- [16] Wikipedia. (22. August 2021). User Experience. Abgerufen am 17. Oktober 2021 von https://de.wikipedia.org/wiki/User_Experience
- [17] RADIKANT. (o. D.). UI & UX Design. Abgerufen am 17. Oktober 2021 von <https://www.radikant.com/wissen/unterschied-ux-ui-design/>
- [18] SoftGuide. (o. D.). Benutzeroberfläche. Abgerufen am 17. Oktober 2021 von <https://www.softguide.de/software-tipps/benutzeroberflaeche>
- [19] Kraus, C., & Augsten, S. (2020). Dev Insider. Was ist eine Cross-Platform App? Abgerufen am 17. Oktober 2021 von <https://www.dev-insider.de/was-ist-eine-cross-platform-app-a-898699/>
- [20] Dart. (o. D.). Abgerufen am 17. Oktober 2021 von <https://dart.dev/>
- [21] Flutter. (o. D.). Abgerufen am 17. Oktober 2021 von <https://flutter.dev/>
- [22] Dart. (2021). Dart overview. Abgerufen am 17. Oktober 2021 von <https://dart.dev/overview>
- [23] Dart. (2021). A tour of the Dart language. Abgerufen am 17. Oktober 2021 von <https://dart.dev/guides/language/language-tour>
- [24] Flutter. (2021). Flutter documentation. Abgerufen am 17. Oktober 2021 von <https://flutter.dev/docs>
- [25] Flutter. (2021). StatelessWidget class. Abgerufen am 17. Oktober 2021 von <https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html?web=1&wdLOR=c96A08CC3-EC2F-4110-9102-029A0A7BB7DF>
- [26] Flutter. (2021). StatefulWidget class. Abgerufen am 17. Oktober 2021 von <https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html?web=1&wdLOR=c89A5ACFF-1B0B-4BD3-B0B9-74085EA60AD5>
- [27] Flutter. (23. Juni 2021). Windows install. Abgerufen am 17. Oktober 2021 von <https://flutter.dev/docs/get-started/install/windows>
- [28] Tiwari, B. (7. Februar 2021). YouTube. Flashlight App In Flutter | Flash Light In Flutter | How to make a Flash Light App in Flutter. Abgerufen am 17. Oktober 2021 von <https://www.youtube.com/watch?v=6pYL1GAUQv0>
- [29] Tiwari, B. (6. Februar 2021). GitHub. FlashLight-App-in-Flutter. Abgerufen am 17. Oktober 2021 von <https://github.com/GeekyBharat99/FlashLight-App-in-Flutter>
- [30] Nurik, R. (2021). Android Asset Studio. Launcher icon generator. Abgerufen am 17. Oktober 2021 von [https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html#foreground.type=clipart&foreground.clipart=android&foreground.space.trim=1&foreground.space.pad=0.25&foreColor=rgba\(96%2C%20125%2C%20139%2C%200\)&backColor=rgb\(68%2C%20138%2C%20255\)&crop=0](https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html#foreground.type=clipart&foreground.clipart=android&foreground.space.trim=1&foreground.space.pad=0.25&foreColor=rgba(96%2C%20125%2C%20139%2C%200)&backColor=rgb(68%2C%20138%2C%20255)&crop=0)

- [31] Flutter. (o. D.). Build and release an Android app. Abgerufen am 17. Oktober 2021 von <https://flutter.dev/docs/deployment/android>
- [32] Dudley, R. (o. D.). git - Der einfache Einstieg. Abgerufen am 17. Oktober 2021 von <https://rogerdudler.github.io/git-guide/index.de.html>
- [33] CloudBees. (2021). Git Pull: How It Works With Detailed Examples. Abgerufen am 17. Oktober 2021 von <https://www.cloudbees.com/blog/git-pull-how-it-works-with-detailed-examples>
- [34] Dart. (2021). Dart documentation. Abgerufen am 17. Oktober 2021 von <https://dart.dev/guides>
- [35] Dart. (2021). Tutorials. Abgerufen am 17. Oktober 2021 von <https://dart.dev/tutorials>
- [36] Ionos. (2020). DART-Tutorial – erste Schritte ganz praktisch. Abgerufen am 17. Oktober 2021 von <https://www.ionos.de/digitalguide/websites/web-entwicklung/dart-programmierung-tutorial/>
- [37] Gool, C. V. (2020). YouTube. Dart Programming | Learn the Basics Step | Dart Introduction 2020. Abgerufen am 17. Oktober 2021 von https://www.youtube.com/watch?v=cKxM0PPXGQw&list=PLzZo-bv_p44SC4-9hX3uSPzLKJ5nLQy3c
- [38] Flutter. (2018). Flutter basics. Abgerufen am 17. Oktober 2021 von <https://docs.flutter.dev/get-started/learn-more#flutter-basics>
- [39] Pelling, S. (2019). YouTube. Flutter Tutorial for Beginners. Abgerufen am 17. Oktober 2021 von <https://www.youtube.com/playlist?list=PL4cUxeGkcC9jLYyp2Aoh6hcWuxFDX6PBJ>
- [40] STARTUP TEENS. (2020). YouTube. Wie du deine eigene App programmierst. Abgerufen am 17. Oktober 2021 von <https://www.youtube.com/playlist?list=PLIYzsTnFhywySa9HZYC8wJvpV3c-Fu-Zs>
- [41] Kennedy, E. (2014). t3n. Die 7 goldenen Regeln des UI-Designs. Abgerufen am 17. Oktober 2021 von <https://t3n.de/news/7-goldene-regeln-ui-design-582053/>
- [42] dribbble. (o. D.). Abgerufen am 17. Oktober 2021 von <https://dribbble.com/>
- [43] Google Design. (o. D.). Fuel your best work with our collection of case studies, profiles, and practical guides. Abgerufen am 17. Oktober 2021 von <https://design.google/library/>
- [44] Adobe. (2021). Tutorial: How to wireframe in Adobe XD. Abgerufen am 17. Oktober 2021 von <https://www.adobe.com/products/xd/learn/design/productivity/how-to-wireframe-xd.html>
- [45] Framer. (o. D.). Design stunning interactive products with your team. Abgerufen am 17. Oktober 2021 von <https://www.framer.com/fp/>
- [46] material. (o. D.). MATERIAL DESIGN. Components. Abgerufen am 17. Oktober 2021 von <https://material.io/components?platform=flutter>

- [47] Wikipedia. (31. März 2021). Material Design. Abgerufen am 17. Oktober 2021 von https://de.wikipedia.org/wiki/Material_Design
- [48] Flutter. (2021). ThemeData class. Abgerufen am 17. Oktober 2021 von <https://api.flutter.dev/flutter/material/ThemeData-class.html>
- [49] pub.dev. (o. D.). provider. Abgerufen am 17. Oktober 2021 von <https://pub.dev/packages/provider>

6.1.1 Fussnoten

- Google Developers. (o. D.). *Android Studio*. Abgerufen am 17. Oktober 2021 von https://developer.android.com/studio?hl=de&gclid=CjwKCAiA7dKMBhBCEiwAO_crFM0nhWEPgoTSbAiest8sM0miQMQLyATB_BohlHyVKGcStSU-mKPBxxoCp7kQAvD_BwE&gclsrc=aw.ds
- Google Developers. (o. D.). *Android Studio. AVD Manager*. Abgerufen am 17. Oktober 2021 von <https://developer.android.com/studio/run/managing-avds>
- microTool. (o. D.). *Was ist ein Burn Down Chart?* Abgerufen am 17. Oktober 2021 von <https://www.microtool.de/wissen-online/was-ist-ein-burn-down-chart/>
- Ryte. (o. D.). Ryte Wiki. Widgets. Abgerufen am 17. Oktober 2021 von <https://de.ryte.com/wiki/Widgets>
- Ryte. (o. D.). Ryte Wiki. Themes. Abgerufen am 17. Oktober 2021 von <https://de.ryte.com/wiki/Themes>
- selfhtml. (22. Juli 2021). *HTML*. Abgerufen am 17. Oktober 2021 von <https://wiki.selfhtml.org/wiki/HTML>
- Wheeler, S., & olprod. (11. Oktober 2021). Microsoft. Was ist PowerShell? Abgerufen am 17. Oktober 2021 von <https://docs.microsoft.com/de-de/powershell/scripting/overview?view=powershell-7.1>
- Wikipedia. (21. Januar 2020). Software Development Kit. Abgerufen am 17. Oktober 2021 von https://de.wikipedia.org/wiki/Software_Development_Kit
- Wikipedia. (7. Juni 2021). Git. Abgerufen am 2021. Oktober 2021 von <https://de.wikipedia.org/wiki/Git>
- Wikipedia. (2. September 2021). No-Code-Plattform. Abgerufen am 17. Oktober 2021 von <https://de.wikipedia.org/wiki/No-Code-Plattform>
- Wikipedia. (21. Juni 2021). Plug-in. Abgerufen am 17. Oktober 2021 von <https://de.wikipedia.org/wiki/Plug-in#:~:text=Ein%20Plug%2Din%20%5B%CB%88pl%CA%8Cg%C9%AAan%5D,ver%C3%A4ndert.>
- Wikipedia. (23. Januar 2021). Entwicklungsstadium (Software). Abgerufen am 17. Oktober 2021 von [https://de.wikipedia.org/wiki/Entwicklungsstadium_\(Software\)](https://de.wikipedia.org/wiki/Entwicklungsstadium_(Software))

Wikipedia. (21. August 2021). *Xcode*. Abgerufen am 17. Oktober 2021 von <https://de.wikipedia.org/wiki/Xcode>

Wikipedia. (31. August 2021). *Syntax*. Abgerufen am 17. Oktober 2021 von <https://de.wikipedia.org/wiki/Syntax>

Wikipedia. (5. August 2021). *Uniform Resource Locator*. Abgerufen am 17. Oktober 2021 von https://de.wikipedia.org/wiki/Uniform_Resource_Locator

Wikipedia. (28. Septemeber 2021). *Programmierschnittstelle*. Abgerufen am 17. Oktober 2021 von <https://de.wikipedia.org/wiki/Programmierschnittstelle>

Wikipedia. (19. Juni 2021). *Global Positioning System*. Abgerufen am 17. Oktober 2021 von https://de.wikipedia.org/wiki/Global_Positioning_System

6.2 Abbildungsverzeichnis

Titelbild: Microsoft Word (o. D.), Piktogramm. [17.Oktober 2021]

Abb. 1: Designprozess einer App: Wireframe, Mockup und Prototyp

Aha. (o. D.). What is the difference: Wireframe vs. Mockup vs. Prototype? Abgerufen am 17. Oktober 2021 von <https://www.aha.io/roadmapping/guide/product-management/wireframe-mockup-prototype>

Abb. 2: «Hello World» Programm in Dart

Dart. (20. August 2021). Language samples. Abgerufen am 17. Oktober 2021 von <https://dart.dev/samples>

Abb. 3: Beispiel eine «for-Schleife» in Dart

DartPad. (o. D.). Hello World. Abgerufen am 17. Oktober 2021 von https://dartpad.dev/?null_safety=true&id=c0f7c578204d61e08ec0fbc4d63456cd

Abb. 4: «Stateless Widget» Beispiel, «Hello World» Programm

Abb. 5: «Hello World», Emulator Ansicht

Abb. 6: «Stateful Widget» Beispiel, «Checkbox» Widget

Abb. 7: «Checkbox» Widget, nicht ausgewählt, Emulator Ansicht

Abb. 8: «Checkbox» Widget, abgehakt, Emulator Ansicht

Abb. 9: Beispiel eines Layouts, Anordnung von Widgets

Flutter. (24. August 2021). Layouts in Flutter. Abgerufen am 17. Oktober 2021 von <https://flutter.dev/docs/development/ui/layout>

Abb. 10: Bewertungsreihe im Widgets Layouts von Abb.9

Flutter. (24. August 2021). Layouts in Flutter. Abgerufen am 17. Oktober 2021 von <https://flutter.dev/docs/development/ui/layout>

Abb. 11: «widget tree», der Bewertungsreihe auf Abb. 10

Flutter. (24. August 2021). Layouts in Flutter. Abgerufen am 17. Oktober 2021 von <https://flutter.dev/docs/development/ui/layout>

Abb. 12: «Flutter» Mindestanforderungen für ein «Windows» Betriebssystem, Datum: 17.10.2021

Flutter. (23. Juni 2021). Windows install. Abgerufen am 17. Oktober 2021 von <https://flutter.dev/docs/get-started/install/windows>

Abb. 13: Auswahlmöglichkeiten bei der Erstellung eines Flutter Projekts

Abb. 14: Ordnerstruktur eines Flutter Projekts

Abb. 15: Code der fertigen Taschenlampen App (Teil 1)

Tiwari, B. (6. Februar 2021). GitHub. FlashLight-App-in-Flutter. Abgerufen am 17. Oktober 2021 von <https://github.com/GeekyBharat99/FlashLight-App-in-Flutter>

Abb. 16: Code der fertigen Taschenlampen App (Teil 2)

Tiwari, B. (6. Februar 2021). GitHub. FlashLight-App-in-Flutter. Abgerufen am 17. Oktober 2021 von <https://github.com/GeekyBharat99/FlashLight-App-in-Flutter>

Abb. 17: «pubspec.yaml» Datei der fertigen Taschenlampen App

Tiwari, B. (6. Februar 2021). GitHub. FlashLight-App-in-Flutter. Abgerufen am 17. Oktober 2021 von <https://github.com/GeekyBharat99/FlashLight-App-in-Flutter>

Abb. 18: Ansicht der ersten Version der Taschenlampe App, links: Lampe deaktiviert, rechts: Lampe aktiviert

Abb. 19: Icon der Taschenlampe App

Abb. 20: «AppBar» Ansicht der zweiten Taschenlampenversion

Abb. 21: Ansicht der zweiten Version der Taschenlampe App, links nach rechts: inaktiv, aktiv, Text eingeblendet.

Abb. 22: Wireframe, links: «Splashscreen», rechts: «Homepage»

Abb. 23: Mockup, «CheckoutOrderCompleted»

Abb. 24: Ordnerstruktur der Appvorlage (...📁 = Ordner)

Abb. 25: Rechts: Beispiel eines Seitenlayouts mit einem Titel in «AppBar» und ein Text im Körper

Abb. 26: Beispiel einer schlechter Code-Implementierung des Layouts aus Abb. 25

Abb. 27: Beispiel einer gute Code-Implementierung des Layouts aus Abb. 25

Abb. 28: Beispiel eines «Container» Widgets ohne Übergabe von Parameter und Eigenschaften in Flutter

Abb. 29: Beispiel eines «Container» Widget mit Übergabe von Farbeigenschaft in Flutter

Abb. 30: Beispiel eines «Container» Widget mit Übergabe von Farbeigenschaft und Grössenparameter in Flutter

Abb. 31: Beispiel eines «Container» Widget mit Übergabe von Farbeigenschaft und Grössenparameter aus Datenmodellen in Flutter

Abb. 32: Erstellung einer benutzerdefinierten Listen Scroll-Ansicht Widget (Teil 1)

Abb. 33: Erstellung einer benutzerdefinierten Listen Scroll-Ansicht Widget (Teil 2)

Abb. 34: Erstellung einer benutzerdefinierten Listen Scroll-Ansicht Widget (Teil 3)

Abb. 35: Ansicht des Appvorlagen-Prototypen, links: «Homepage», Mitte und rechts: «Produkt-Information-Seite»

Abb. 36: Wireframe Endversion, links-rechts: «Homepage», «Profile + Settings», «Authentication»

Abb. 37: Mockup Endversion, von links oben nach rechts unten: «Homepage», «Product-Screen», «Cart», «Profile», «Settings», «Settings, Help», «Checkout, Information», «Checkout, Payment», «Checkout, OrderCompleted»

Abb. 38: Mockup Endversion, links: «Library», rechts: «Discover»

Abb. 39: Appvorlage-Prototyp Endversion, Seiten: links-rechts: «Homepage, Scroll» «CategoryScreen», «FilterDrawer»

Abb. 40: Appvorlage Prototyp Endversion, links-rechts: «Welcome», «Authentication», «CheckoutProcess»

Abb. 41: Appvorlage Prototyp Endversion, Seiten: links-rechts: «Settings», «FilterDrawer», «Profile»

Abb. 42: Appvorlage Prototyp Endversion, Seiten: links-rechts: «DiscoverMap», «FilterDrawer», «DiscoverProducts»

6.3 Tabellenverzeichnis

Tabelle 1: Beispiel für schlechte und gute Umsetzung von «UI Design» Grundsätze

Simon, G. (9. Januar 2019). YouTube. UI Design Crash Course for Beginners. Abgerufen am 17. Oktober 2021 von https://www.youtube.com/watch?v=_Hp_dI0DzY4

Tabelle 2: «Stateful Widget» und «Stateless Widget» im Vergleich

Mistry, M. (2021). kodytechnolab. Stateless and Stateful widget comparison. Abgerufen am 17. Oktober 2021 von <https://kodytechnolab.com/blog/stateless-vs-stateful-widget/>

Tabelle 3: Flutter und React Native im Vergleich

Skuz, B., Mroczkowska, A., & Włodarczyk, D. (2021). Droids on Roids. Flutter vs. React Native – What to Choose in 2021? Abgerufen am 17. Oktober 2021 von <https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2021>

Tabelle 4: App Publizierung bei iOS und Android Plattform im Vergleich

Tabelle 5: GitHub, GitLab und Bitbucket Plattformen im Vergleich

GeeksforGeeks. (8. September 2021). Difference Between GitLab and GitHub. Abgerufen am 17. Oktober 2021 von <https://www.geeksforgeeks.org/difference-between-gitlab-and-github/>

Tabelle 6: Liste der Elemente nach Hauptseiten der App geordnet

7 Eidesstattliche Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig und unter Angabe aller benötigten Quellen verfasst zu haben.

Datum: 17.10.2021

A handwritten signature in black ink, appearing to be 'D. Grahik', written in a cursive style with a horizontal line underneath.