

1 Technologie Auswahl

Die folgenden drei Kapitel behandeln die Auswahl der Frontend-Technologie für die Umsetzung der Formularanwendung. Dazu werden im ersten Schritt die dafür infrage kommenden Technologien identifiziert. Anschließend wird der Trend der Popularität dieser Technologien miteinander verglichen. Die daraus resultierenden Kandidaten sollen dann detaillierter untersucht werden. In Hinblick auf die Anforderungen an die Formularanwendung soll dabei die angemessenste Frontend-Technologie ausgewählt werden.

1.1 Trendanalyse

Zwei Quellen wurden für die Analyse der Technologie-Trends ausgewählt: die Ergebnisse der jährlichen *Stack Overflow*-Umfragen und das Suchinteresse von Google Trends.

1.1.1 *Stack Overflow*-Umfrage

Die Internetplattform *Stack Overflow* richtet sich an Softwareentwickler und bietet ihren Nutzern die Möglichkeiten, Fragen zu stellen, Antworten einzustellen und Antworten anderer Nutzer auf- und abzuwerten.

Besonders für Fehlermeldungen, die häufig während der Softwareentwicklung auftreten, findet man auf dieser Plattform rasch die Erklärung und den Lösungsvorschlag gleich mit. So lässt sich auch die Herkunft des Domainnamens herleiten:

We named it Stack Overflow, after a common type of bug that causes software to crash – plus, the domain name stackoverflow.com happened to be available.
— Joel Spolsky, Mitgründer von *Stack Overflow*¹

Aufgrund des Erfolgsrezepts von *Stack Overflow* ist die Plattform kaum einem Softwareentwickler unbekannt. Dementsprechend nehmen auch jährlich tausende Entwickler an den von *Stack Overflow* herausgegebenen Umfragen teil. Seit 2013 beinhalten die Umfragen

¹Spolsky, *How Hard Could It Be?: The Unproven Path*.

auch die Angabe der aktuell genutzten und in Zukunft gewünschten Frontend-Technologien. *Stack Overflow* erstellt aus diesen gesammelten Daten Auswertungen und Übersichten. Die zugrundeliegenden Daten werden ebenfalls veröffentlicht.²

Um den Trend der Beliebtheit der Frontend-Technologien aufzuzeigen, wurde ein Jupyter Notebook erstellt. Es transformiert die Daten in ein einheitliches Format, da die Umfrageergebnisse von Jahr zu Jahr in einer unterschiedlichen Struktur abgelegt wurden. Anschließend erstellt es Diagramme, die im Folgenden analysiert werden.

1.1.2 Google Trends

Suchanfragen, die über die Suchmaschine Google abgesetzt werden, lassen sich über den Dienst Google Trends als Trenddiagramm visualisieren. Die Ergebnisse werden normalisiert, um das relative Suchinteresse abzubilden und die Ergebnisse auf einer Skala von 0 bis 100 darstellen zu können.³

Google Trends ist keine wissenschaftliche Umfrage und sollte nicht mit Umfragedaten verwechselt werden. Es spiegelt lediglich das Suchinteresse an bestimmten Themen wider.⁴

Genau aus diesem Grund wird Google Trends im Folgenden lediglich zum Abgleich der Ergebnisse der *Stack Overflow* Umfrage eingesetzt.

1.1.3 Frameworks mit geringer Relevanz

NativeScript, *Sencha* (bzw. *Sencha Touch*) und *Appcelerator* spielen in den Umfrageergebnissen eine untergeordnete Rolle. Dies ist in den aufsummierten Stimmen von 2013 bis 2020 für alle in der Umfrage auftauchenden Frontend-Technologien zu sehen (Abb. 1.1). Auch das Suchinteresse auf Google ist für diese Frameworks äußerst gering. In Abbildung 1.2 wird das relative Suchinteresse von *NativeScript*, *Sencha*, *Appcelerator*, *Adobe PhoneGap* und *Apache Cordova* miteinander verglichen.

Abbildung 1.1: Summe der Stimmen der *Stack Overflow*-Umfrage von 2013 bis 2020, Quelle: Eigene Abbildung, Notebook: [Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb](#), Daten-Quelle: <https://insights.stackoverflow.com/survey>

Abbildung 1.2: Suchinteresse der Frameworks mit geringer Relevanz, Quelle: Eigene Abbildung, Notebook: [Charts/GoogleTrends/GoogleTrends.ipynb](#), Daten-Quelle: Google Trends

²Vgl. Stack Exchange, Inc., *Stack Overflow Insights / Stack Overflow Annual Developer Survey*.

³Vgl. Google LLC, *Google Trends-Hilfe / Häufig gestellte Fragen zu Google Trends-Daten*.

⁴Google LLC, *Google Trends-Hilfe / Häufig gestellte Fragen zu Google Trends-Daten*.

Verwandte Technologien zu Apache Cordova

Das *Ionic*-Framework taucht in den Ergebnissen der *Stack Overflow*-Umfragen nicht auf. Ein Grund dafür könnte sein, dass es auf *Apache Cordova* aufbaut⁵, welches bereits in den Ergebnissen vorkommt. *Adobe PhoneGap* taucht zwar in den Ergebnissen von 2013 mit 1043 Stimmen auf (Abb. 1.3), verliert jedoch in den Folgejahren mit weniger als 10 Stimmen abrupt an Relevanz.

Abbildung 1.3: Stimmen für *Cordova* und *PhoneGap*, Quelle: Eigene Abbildung, Notebook: [Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb](#), Daten-Quelle: <https://insights.stackoverflow.com/survey>

Das stimmt nicht mit dem Suchinteresse auf Google überein, da *Adobe PhoneGap* dort erst ab 2014 anfängt, langsam an Relevanz zu verlieren (Abb. 1.2). 2013 existierte *PhoneGap* noch als extra Mehrfachauswahlfeld in den Daten, während es ab 2014 nur noch in dem Feld für die sonstigen Freitext Angaben auftaucht.⁶ Auch *Adobe PhoneGap* baut auf *Apache Cordova* auf.⁷ Für diese Auswertung spielen diese verwandten Technologien eine untergeordnete Rolle, da sie auch in den Google Trends weit hinter *Apache Cordova* zurückbleiben.

Am Beispiel von *Adobe PhoneGap* wird deutlich, wie wichtig es ist, auf eine Technologie zu setzen, die weit verbreitet ist. Im schlimmsten Fall wird die Technologie sogar vom Betreiber aufgrund zu geringer Nutzung komplett eingestellt, wie es bei *PhoneGap* bereits geschehen ist. Adobe gab am 11. August 2020 bekannt, dass die Entwicklung an *PhoneGap* eingestellt wird und empfiehlt die Migration hin zu *Apache Cordova*.⁸

1.1.4 Frameworks mit sinkender Relevanz

Die Technologien *Xamarin* und *Cordova* zeigen bereits einen abfallenden Trend, wie in Abbildung 1.4 ersichtlich ist. Im Fall von *Xamarin* gibt es immerhin mehr Entwickler, die sich wünschen, mit dem Framework zu arbeiten, als Entwickler, die tatsächlich mit *Xamarin* arbeiten. *Cordova* scheint in diesem Hinblick dagegen eher unbeliebt: Es gibt mehr Entwickler, die mit *Cordova* arbeiten, als tatsächlich damit arbeiten wollen.

Abbildung 1.4: Stimmen für *Xamarin* und *Cordova*, Quelle: Eigene Abbildung, Notebook: [Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb](#), Daten-Quelle: <https://insights.stackoverflow.com/survey>

⁵Vgl. Lynch, *The Last Word on Cordova and PhoneGap*.

⁶Vgl. Stack Exchange, Inc., *Stack Overflow Insights | Stack Overflow Annual Developer Survey*.

⁷Vgl. Adobe Inc., *PhoneGap Docs | FAQ*.

⁸Vgl. Adobe Inc., *Update for Customers Using PhoneGap and PhoneGap Build*.

In Abbildung 1.5 ist noch einmal zu sehen, dass Google Trends die Erkenntnisse aus der *Stack Overflow*-Umfrage reflektiert; und es wird auch sichtbar, welche beiden Technologien möglicherweise der Grund für den Rückgang von *Xamarin* und *Cordova* sind.

Abbildung 1.5: Suchinteresse sinkende und steigende Relevanz, Quelle: Eigene Abbildung, Notebook: [Charts/GoogleTrends/GoogleTrends.ipynb](#), Daten-Quelle: Google Trends

1.1.5 Frameworks mit steigender Relevanz

Besser ist es, auf Technologien zu setzen, die noch einen steigenden Trend der Verbreitung und Beliebtheit zeigen. In Abbildung 1.6 wird sichtbar, dass es sich dabei um *Flutter* und – immerhin im Hinblick auf die Verbreitung – auch um *React Native* handelt. Ungünstigerweise wird *React Native* in der *Stack Overflow*-Umfrage erst seit 2018 als tatsächliches Framework abgefragt. Vorher erschien lediglich das Framework React, welches sich nicht für den Vergleich der *Cross-Plattform-Frameworks* eignet, da es sich um ein reines Webframework handelt. Doch auch die Ergebnisse von Google Trends zeigen einen ähnlichen Verlauf für die Jahre 2019 und 2020 (Abb. 1.5).

Abbildung 1.6: Stimmen gewünschter Frameworks: *React Native*, *Flutter*, *Xamarin* und *Cordova*, Quelle: Eigene Abbildung, Notebook: [Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb](#), Daten-Quelle: <https://insights.stackoverflow.com/survey>

Im Vergleich des Jahres 2019 mit 2020 wird sichtbar, dass die Zahl der Entwickler, die sich wünschen, mit *React Native* zu arbeiten, gesunken ist. Dennoch ist die Anzahl der Entwickler, die mit *React Native* arbeiten möchten, noch weit höher, als die der Entwickler, die tatsächlich mit *React Native* arbeiten.

Es ist möglich, dass der abfallende Trend daran liegt, dass die Zahl der Entwickler, die mit *Flutter* arbeiten möchten, im selben Jahr gestiegen ist. *React Native* hat im Vergleich zu *Flutter* jedoch noch immer mehr aktive Entwickler und die Tendenz ist steigend. Doch die Anzahl der aktiven *Flutter*-Entwickler zeigt einen noch stärker steigenden Trend. So könnte es sein, dass die Zahl der *Flutter*-Entwickler die der *React Native*-Entwickler in einem der nächsten Jahre überholt. Im Suchinteresse hat sich diese Entwicklung bereits vollzogen (Abb. 1.5). Auch in der Anzahl der Entwickler, welche *Flutter* einsetzen, ist ein steiler Aufwärtstrend zu beobachten (Abb. 1.7). Damit überholt *Flutter* die etablierten Frameworks *Xamarin* und *Cordova*. Gleichzeitig muss betrachtet werden, dass die Anzahl der Nutzer, die *React Native* als verwendete Technologie angegeben haben, noch immer höher ist und – wenn auch stagnierend – weiter steigt.

Abbildung 1.7: Stimmen verwendeter Frameworks: *React Native*, *Flutter*, *Xamarin* und *Cordova*, Quelle: Eigene Abbildung, Notebook: [Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb](#), Daten-Quelle: <https://insights.stackoverflow.com/survey>

Nichtsdestotrotz scheinen beide Technologien als Kandidaten für einen detaillierteren Vergleich für dieses Projekt in Frage zu kommen. Im nächsten Kapitel soll evaluiert werden, welches Framework für die Entwicklung der Formularanwendung angemessener ist.

1.2 Vergleich von *React Native* und *Flutter*

Es soll eine Formularanwendung mit komplexer Validierung im Rahmen dieser These erstellt werden. Es ist durchaus sinnvoll, die beiden Technologien anhand von Beispielanwendungen, welche Formulare und die Validierung dieser beinhalten, zu vergleichen. Deshalb soll nachfolgend jeweils eine solche Beispielanwendung der jeweiligen Technologie gefunden werden. Die Anwendungen werden sich stark voneinander unterscheiden, weshalb sie im nächsten Schritt vereinfacht und aneinander angeglichen werden. Anschließend wird ersichtlich werden, nach welchen Kriterien sich die Technologien im Hinblick auf die Entwicklung der Formularanwendung vergleichen lassen.

1.2.1 Vergleich zweier minimaler Beispiele für Formulare und Validierung

Formulare in *React Native*

React Native stellt nur eine vergleichsweise geringe Anzahl von eigenen Komponenten zur Verfügung und zu diesen gehören keine, welche die Validierung von Formularen ermöglichen. Doch die im *react.js* Raum sehr bekannten Bibliotheken *Formik*, *Redux Forms* und *React Hook Form* sind alle drei kompatibel mit *React Native*.^{9,10,11}

Für die Formularanwendung ist die Validierung komplexer Bedingungen nötig. Die Formular-Validierungs-Bibliotheken bieten in der Regel Funktionen an, welche überprüfen, ob ein Feld gefüllt ist oder der Inhalt einem speziellen Muster entspricht – wie etwa einem regulären Ausdruck. Doch solche mitgelieferten Validierungs-Funktionen reichen nicht aus, um die Komplexität der Bedingungen abzubilden. Stattdessen müssen benutzerdefinierte Funktionen zum Einsatz kommen.

Keiner der drei oben genannten Validierungs-Bibliotheken ist in dieser Hinsicht limitiert. Sie alle bieten die Möglichkeit, eine *JavaScript*-Funktion für die Validierung zu übergeben. Diese Funktion gibt einen Wahrheitswert zurück – wahr, wenn das Feld oder die Felder valide sind, falsch, falls nicht. In *React Hook Form* ist es die Funktion *register*, die ein Parameterobjekt namens *RegisterOptions* erhält. Der Eigenschaft *validate* dieses Objektes

⁹Vgl. *Formik Docs / React Native*.

¹⁰Vgl. *Does redux-form work with React Native?*

¹¹Vgl. *React Hook Form - Get Started*.

kann eine *JavaScript*-Funktion für die Validierung übergeben werden.¹² In *Redux Form* ist es die Initialisierungs-Funktion *reduxForm*, die ein Konfigurations-Objekt mit dem Namen *config* erhält, in welchem die Eigenschaft ebenfalls *validate* heißt.¹³ Auch in *Formic* ist der Bezeichner *validate*, und ist als Attribut in der *Formic*-Komponente zu finden.¹⁴

Es ist also absehbar, dass die Formularanwendung in *React Native* entwickelt werden kann. Die nötigen Funktionen werden von den Bibliotheken bereitgestellt. Einziger Nachteil hierbei ist, dass es sich um Drittanbieter-Bibliotheken handelt, welche im Verlauf der Zeit an Beliebtheit gewinnen und verlieren können. Möglicherweise geht die Beliebtheit einer der Bibliotheken mit der Zeit zurück, weshalb es weniger Kontributionen wie etwa neue Funktionalitäten oder Fehlerbehebungen, sowie Fragen, Antworten und Anleitungen zu diesen Bibliotheken geben wird, da die Entwickler sich für andere Bibliotheken entscheiden. Die Wahl der Bibliothek kann also schwerwiegende Folgen wie Mangel an Dokumentation oder Limitationen im Vergleich zu anderen Bibliotheken mit sich bringen. Eine Migration von der einen Bibliothek zu einer anderen könnte in Zukunft notwendig werden, wenn diese Limitationen während der Entwicklung auffallen. Aus dem Grund ist es in der Regel von Vorteil, wenn solche Funktionalitäten bereits im Kern der Frontend-Technologie integriert sind. Der Fall, dass die Kernkomponenten an Relevanz verlieren und empfohlen wird, auf externe Bibliotheken zuzugreifen, ist zwar nicht ausgeschlossen, geschieht aber im Wesentlichen seltener.

Für den Vergleich wurde eine Schritt-für-Schritt-Anleitung zum Erstellen eines Formulars mit *React Hook Form* ausgewählt.

Formulare in *Flutter*

Die *Flutter*-Dokumentation stellt in ihrer *cookbook*-Sektion ein Beispiel einer minimalistischen Formularanwendung mit Validierung bereit.¹⁵ Das Rezept ist Teil einer Serie von insgesamt fünf Anleitungen, welche Formulare in *Flutter* behandeln.¹⁶ Die Rezepte wurden genutzt, um eine Formularanwendung zu implementieren, welche dem Ergebnis der *React Hook Form*-Applikation in Funktionalität und Layout ähnlich ist.

Ergebnisse des Vergleiches

Abbildung 1.8 zeigt die *Flutter*-Anwendung (links) und die *React Native*-Anwendung (rechts).

¹²Vgl. *React Hook Form - API | register*.

¹³Vgl. *Redux Form - API | reduxForm*.

¹⁴Vgl. *Formik Docs API | <Formik />*.

¹⁵Vgl. Google LLC, *Build a form with validation*.

¹⁶Vgl. Google LLC, *Flutter Docs Cookbook | Forms*.

¹ <https://dev.to/elaziziyousouf/forms-in-react-native-the-right-way-4d46>

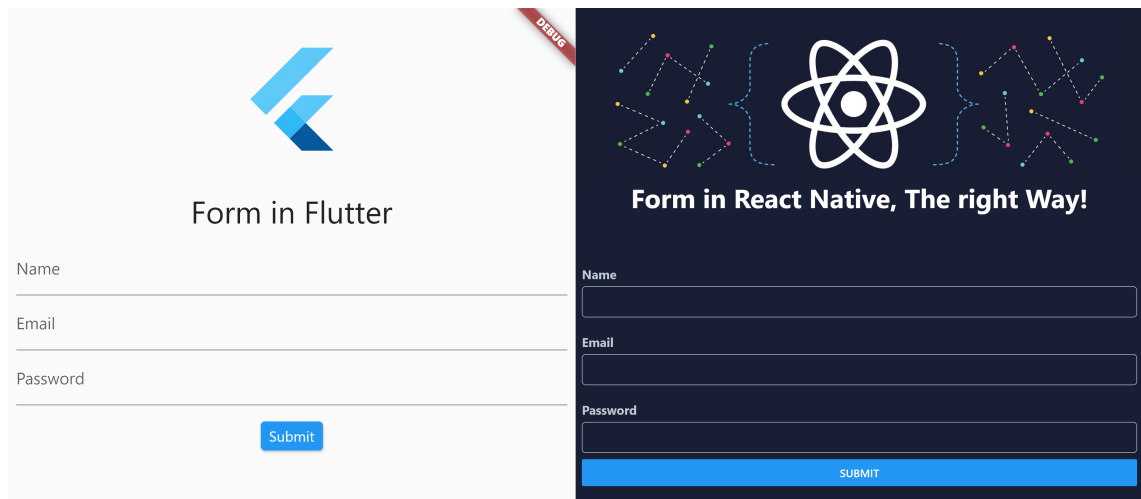


Abbildung 1.8: Gegenüberstellung der minimalistischen Flutter- und *React Native*-Formularapplikationen

Tabelle 1.1: Anzahl der Quelltextzeilen der minimalistischen *Flutter* und *React Native*-Formular-Applikationen.

Die Listings ?? und ?? des *Flutter*-Formulars sind in Anhang ?? auf den Seiten ?? und ?? zu finden. Anhang ?? beinhaltet die Listings ??, ??, ??, ??, ?? und ?? der *React Hook Form*-Anwendung auf den Seiten ?? bis ??.

Folgende Unterschiede waren bei der Untersuchung beider Applikationen auffällig:

- Die Anzahl der benötigten Quelltextzeilen unterscheidet sich stark.
- *Flutter*-Widgets bringen ohne zusätzliche Konfiguration bereits ein ansprechendes Aussehen mit. *React Native* Anwendungen müssen dafür mit zusätzlichen Stylesheets ausgestattet werden.

Tabelle 1.1 auf Seite 7 stellt die Anzahl der Quelltextzeilen der unterschiedlichen Anwendungen auf Dateiebene gegenüber.

Kommentare, leere Zeilen und *import*-Anweisungen wurden bei der Untersuchung ignoriert. Es wurde kein Versuch unternommen, den Quellcode der *React Native*-Applikation zu refaktorisieren, sodass auch dieser die minimal nötige Anzahl von Zeilen aufweist. Die allgemeine Anzahl der Quelltext-Zeilen ist für die Evaluierung der Technologie deshalb zweitrangig.

Vor dem Hintergrund, dass die Wissenschaftler und Wissenschaftlerinnen des Thünen-Instituts keine Anforderungen an das Aussehen der Oberfläche stellen, soll sich für die Technologie entschieden werden, welche den Aufwand für das Gestalten der Oberfläche gering hält. In diesem Fall ist das *Flutter*, da es in der Standardkonfiguration bereits ein ansprechendes Design mitbringt. Die *React Native*-Applikation benötigte dagegen mit 55


Zeilen in den *Stylesheets* bereits 73 % der Zeilenanzahl, die insgesamt für die Entwicklung des Formulars in *Flutter* nötig waren. 


«««< HEAD

1.2.2 Automatisiertes Testen


=====

1.2.3 Automatisiertes Testen

 > main

Um die Codequalität zu gewährleisten, sollen bei der Entwicklung Unit- und Integrations-tests zum Einsatz kommen. Im Folgenden sollen die Frameworks *Flutter* und *React Native* in Hinblick auf die Dokumentation zu diesen Themen untersucht werden. 

Automatisierte Tests in *React Native*



Die *React Native*-Dokumentation führt genau eine Seite mit einem Überblick über die unterschiedlichen Testarten. Dabei wird das Konzept von Unittests, *Mocking*, Integrationstests, Komponententests und *Snapshot*-Tests kurz erläutert, jedoch ohne ein Beispiel zu geben oder zu verlinken. Vier Quellcodeschnipsel sind auf der Seite zu finden: Ein Schnipsel zeigt den minimalen Aufbau eines Tests; zwei weitere Schnipsel veranschaulichen beispielhaft, wie Nutzerinteraktionen getestet werden können. Letzteres zeigt die textuelle Repräsentation der Ausgabe einer Komponente, die für einen *Snapshot*-Test verwendet wird. Weiterhin wird auf die *Jest*-API-Dokumentation verwiesen, sowie auf ein Beispiel für einen *Snapshot*-Test in der *Jest*-Dokumentation.^{II} 

Um die notwendigen Anleitungen für das Erstellen der jeweiligen Tests ausfindig zu machen, ist es notwendig, die Dokumentation von *React Native* zu verlassen.

Die Dokumentation von *Jest* enthält mehr Details zum Einsatz der Testbibliothek, welche für mehrere Frontend-Frameworks kompatibel ist, die auf *JavaScript* basieren^{III}. Somit muss zum Erstellen der Unittests immerhin nur dieses Framework studiert werden.

^{II} <https://jestjs.io/docs/snapshot-testing>

^{III} <https://jestjs.io/docs/getting-started>


Zum Entwickeln von Tests für *React Native*-Komponenten wird unter anderem auf die Bibliothek *React Native Testing Library* verwiesen. Anders als der Name vermuten lässt, handelt es sich nicht um eine von *React Native* bereitgestellte Bibliothek. Im Unterschied zur *React Testing Library*, von der sie  inspiriert ist, läuft sie – so wie *React Native* auch – nicht in einer Browser-Umgebung.¹⁷  verausgegeben wird die *React Native Testing Library* vom Drittanbieter *Callstack* – einem Partner im *React Native*-Ökosystem.¹⁸

Sie verwendet im Hintergrund den *React Test Renderer*^{IV}, welcher wiederum vom React Team angeboten wird und auch zum Testen von *react.js* Anwendungen geeignet ist. Der *React Test Renderer* wird ebenfalls empfohlen, um Komponententests zu kreieren, die keine *React Native* spezifischen Funktionalitäten nutzen.

Um Integrationstests zu entwickeln – welche die Applikation auf einem physischen Gerät oder auf einem Emulator testen – wird auf zwei weitere Drittanbieter-Bibliotheken verlinkt: *Appium*^V und *Detox*^{VI}. Es wird darauf hingewiesen, dass *Detox* speziell für die Entwicklung von *React Native*-Integrationstests entwickelt wurde. *Appium* wird lediglich als ein weiteres bekanntes Werkzeug erwähnt.

Es lässt sich damit zusammenfassen, dass der Aufwand der Einarbeitung für automatisiertes Testen in *React Native* vergleichsweise hoch ist. Die Dokumentation ist auf die Seiten der jeweiligen Anbieter verteilt. Der Entwickler muss sich den Überblick selbst verschaffen und zusätzlich die für das Framework *React Native* relevanten Inhalte identifizieren. Notwendig ist auch das Erlernen von mehreren APIs um alle Testarten abzudecken. Für einen Anfänger kommt erschwerend hinzu, dass eine Entscheidung für die eine oder andere Bibliothek notwendig wird. Um diese Entscheidung treffen zu können, ist eine Auseinandersetzung mit den Vor- und Nachteilen der Technologien im Vorfeld vom Entwickler zu leisten.

Automatisierte Tests in Flutter

Die *Flutter*-Dokumentation erklärt sehr umfangreich auf 11 Unterseiten die unterschiedlichen Testarten mit Quellcodebeispielen und verlinkt für jede Testart auf eine  oder mehrere detaillierte Schritt-für-Schritt-Anleitungen, wie ein solcher Test erstellt wird.

Eine Seite erklärt den Unterschied zwischen Unittests, *Widget*-Tests und Integrationstests^{VII}. Eine weitere Seite erklärt Integrationstests detaillierter^{VIII}.

¹⁷Vgl. Borenkraout, *Testing Library Docs / Native Testing Library Introduction*.

¹⁸Vgl. Facebook Inc., *The React Native Ecosystem*.

^{IV} <https://reactjs.org/docs/test-renderer.html>

^V <http://appium.io/>

^{VI} <https://github.com/wix/detox/>

^{VII} <https://flutter.dev/docs/testing>

^{VIII} <https://flutter.dev/docs/testing/integration-tests>

Ein sogenanntes *Codelab* führt durch die Erstellung einer minimalistischen App und der anschließenden Implementierung von zwei Unit-, fünf *Widget*- und zwei Integrationstests für diese App^{IX}.

Im sogenannten Kochbuch tauchen folgende Rezepte auf:

- 2 Rezepte für Unittests
 - eine grundlegende Anleitung zum Erstellen von Unittests ^X
 - Eine weitere Anleitung zum Nutzen von *Mocks* in Unittest mithilfe der Bibliothek *mockito* ^{XI}
- 3 Rezepte für *Widget*-Tests
 - Eine grundlegende Anleitung zum Erstellen von *Widget*-Tests ^{XII}
 - Ein Rezept mit detaillierteren Beispielen zum Finden von *Widgets* zur Laufzeit eines *Widget*-Tests ^{XIII}
 - Ein Rezept zum Testen vom Nutzerverhalten wie dem Tab, dem Drag und dem Eingeben von Text ^{XIV}
- 3 Rezepte für Integrationstests
 - Eine grundlegende Anleitung zum Erstellen eines Integrationstests ^{XV}
 - eine Anleitung zum Simulieren des Scrollens in der Anwendung während der Laufzeit eines Integrationstests ^{XVI}
 - eine Anleitung zum Performance Profiling ^{XVII}

1.3 Fazit und Begründung der Auswahl



Die beiden Frameworks *Flutter* und *React Native* wurden in diesem Kapitel auf die Möglichkeit zur Erstellung von Formularanwendungen geprüft. Beide Frameworks stellen die nötigen Komponenten dafür bereit. In *Flutter* ist der Vorteil zudem, dass die Komponenten in der Standardbibliothek enthalten sind. In *React Native* müssen Bibliotheken wie etwa *Formic*, *Redux Forms* oder *React Hook Form* dafür eingebunden werden.

^{IX} <https://codelabs.developers.google.com/codelabs/flutter-app-testing>

^X <https://flutter.dev/docs/cookbook/testing/unit/introduction>

^{XI} <https://flutter.dev/docs/cookbook/testing/unit/mocking>

^{XII} <https://flutter.dev/docs/cookbook/testing/widget/introduction>

^{XIII} <https://flutter.dev/docs/cookbook/testing/widget/finders>

^{XIV} <https://flutter.dev/docs/cookbook/testing/widget/tap-drag>

^{XV} <https://flutter.dev/docs/cookbook/testing/integration/introduction>

^{XVI} <https://flutter.dev/docs/cookbook/testing/integration/scrolling>

^{XVII} <https://flutter.dev/docs/cookbook/testing/integration/profiling>

Für einen Vergleich wurde eine minimalistische *Flutter*-Formularanwendung implementiert, welche einer Beispielanwendung für die Bibliothek *React Hook Form* ähnlich ist. Die *Flutter*-Anwendung hatte weniger Quellcode, entscheidender ist aber: Es war ein äußerst geringer Aufwand nötig, um eine ähnlich benutzerfreundliche Oberfläche zu erstellen. Auch in diesem Punkt ist *Flutter React Native* vorzuziehen.

Schließlich wurden beide Frameworks hinsichtlich der Erstellung von automatisierten Tests untersucht. Die Anleitungen für das Testen in *React Native* sind auf mehreren Webportalen verteilt und eine Evaluierung der Bibliotheken und Testtreiber durch den Entwickler ist nötig. Der Aufwand der Einarbeitung in das Testen in *Flutter* ist dagegen gering. Alle Werkzeuge werden vom *Dart*- und *Flutter*-Team bereitgestellt. Die Dokumentation ist umfangreich, folgt jedoch einem roten Faden. Eine Übersichtsseite fasst die Kerninformationen zusammen und verweist auf die jeweiligen Seiten für detailliertere Informationen und Übungen.

Als Ergebnis dieser drei Vergleiche wird dementsprechend *Flutter* als Technologie zur Umsetzung der Formularanwendung ausgewählt.

2 Konzeption

In diesem Kapitel soll die grafische Benutzeroberfläche konzipiert werden. Dazu gehören der Übersichtsbildschirm, die Eingabemaske und der Selektionsbildschirm.

2.1 Der Übersichtsbildschirm

Bei Programmstart wird der Benutzer mit dem Übersichtsbildschirm begrüßt (Abb. 2.1). Er listet die bisher eingegebenen Maßnahmen auf. Sie werden in zwei Rubriken gruppiert. Maßnahmen, in welchen bereits alle Eingabefelder gefüllt und valide sind, werden in der Gruppe *Abgeschlossen* eingeblendet. Maßnahmen, welche sich noch im Bearbeitungsmodus befinden, da ihnen Inhalte in den Eingabefeldern fehlen oder die Eingaben nicht valide sind, erscheinen in der Rubrik *in Bearbeitung*.

Klickt der Benutzer auf den Aktionsbutton unten rechts, so gelangt er auf den zweiten Bildschirm: die Eingabemaske.

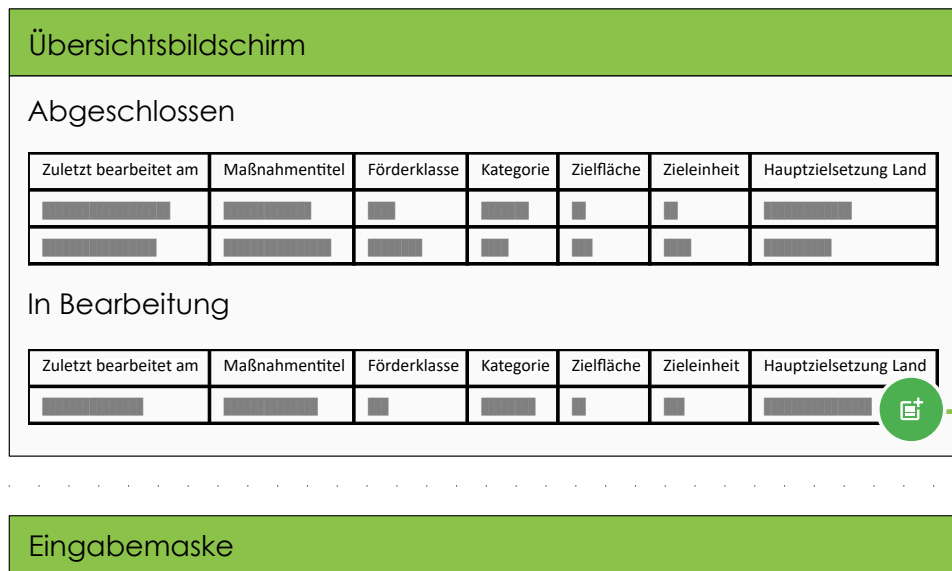


Abbildung 2.1: Konzeption des Übersichtsbildschirms

2.2 Die Eingabemaske

Die Eingabemaske (Abb. 2.2) listet die Eingabefelder für die Eigenschaften der Maßnahme. Bedeutsam sind hierbei vor allem die Einfach- und Mehrfachauswahlfelder. Sie werden als Karten dargestellt. Unterhalb des Titels der Eigenschaft, dessen Wert mit der Selektionskarte ausgewählt werden soll, erscheint auch die Anzeige des bisher ausgewählten Wertes – für Einfachauswahlfelder – bzw. der aktuell ausgewählten Werte – für Mehrfachauswahlfelder. Selektionskarten, welche invalide Werte enthalten, werden rot eingefärbt. Die Selektionskarten können über Überschriften in Gruppen zusammengefasst werden, um die Übersichtlichkeit zu erhöhen. Auch Zwischenüberschriften sollen möglich sein.

Die zwei Aktionsbuttons unten rechts ermöglichen das Speichern der Maßnahme. Der untere der beiden wird dazu verwendet, die Maßnahme vor dem Speichern zu validieren. Nur wenn die Validierung erfolgreich ist, wird die Maßnahme auch gespeichert und der Benutzer gelangt zurück zum Übersichtsbildschirm. Anderenfalls erhält er eine Fehlermeldung. Mit dem Button darüber wird die Maßnahme dagegen direkt im Entwurfsmodus abgespeichert, ohne eine Validierung durchzuführen. Auch nach Anklicken dieses Buttons gelangt der Nutzer zurück zum Übersichtsbildschirm. Klickt der Benutzer auf den Zurück-Button oben links im Bildschirm, so wird versucht, die Eingabemaske wieder zu verlassen, sofern dies möglich ist. Ist die Maßnahme im Entwurfsmodus, so gelangt der Benutzer mit diesem Button direkt zurück zum Übersichtsbildschirm, ist die Maßnahme dagegen im Modus *Abgeschlossen*, so wird zunächst eine Validierung ausgeführt.

Mit einem Klick auf die Selektionskarten wird der Benutzer auf den Selektionsbildschirm weitergeleitet, um Auswahloptionen für das angeklickte Auswahlfeld auszuwählen.



Abbildung 2.2: Konzeption der Eingabemaske

2.3 Der Selektionsbildschirm

Auf dem Selektionsbildschirm (Abb. 2.3) werden alle möglichen Auswahloptionen aufgelistet. Mit einem Klick darauf kann der Benutzer die Optionen aktivieren und deaktivieren.

Auswahloptionen, welche mit den in anderen Auswahlfeldern ausgewählten Werten nicht kompatibel sind, erscheinen am Ende der Liste. Sie sind mit einem Kreuz-Symbol gekennzeichnet. Optionen, welche zuvor angewählt waren und durch eine neue Selektion nun invalide geworden sind, erscheinen mit einem roten Hintergrund.

Klickt der Benutzer auf den Aktionsbutton unten rechts im Bildschirm oder auf den Zurück-Button oben links, so gelangt er zurück zur Eingabemaske.

3 Diskussion

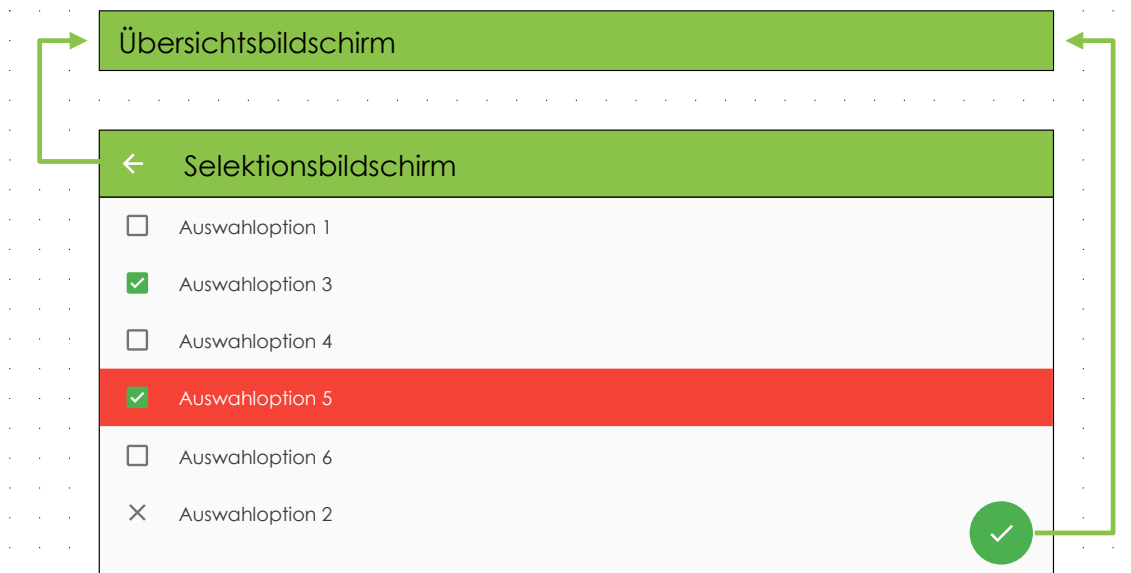


Abbildung 2.3: Konzeption des Selektionsbildschirms

4 Schlussfolgerung-und-Ausblick

Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Masterarbeit *Entwicklung einer Formularanwendung mit Kompatibilitätsvalidierung der Einfach- und Mehrfachauswahl-Eingabefelder* selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Ich versichere, dass die eingereichte schriftliche Fassung der auf dem beigefügten Medium gespeicherten Fassung entspricht.

Wernigerode, den 01.09.2021