Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Leonard Thiele

Cross-Plattform-Entwicklung einer mobilen Anwendung mit Flutter – Konzeption und prototypische Umsetzung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Leonard Thiele

Cross-Plattform-Entwicklung einer mobilen Anwendung mit Flutter – Konzeption und prototypische Umsetzung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Zukunft
Zweitgutachter: Prof. Dr. Sarstedt

Eingereicht am: 7. Dezember 2018

Leonard Thiele

Thema der Arbeit
Cross-Plattform-Entwicklung einer mobilen Anwendung mit Flutter – Konzeption und prototypische Umsetzung

Stichworte
Cross-Plattform-Entwicklung, Flutter, Dart, Android, iOS

Kurzzusammenfassung
Die Entwicklung mobiler Apps für Android und iOS stellt eine technisch komplexe und kostspielige Aufgabe dar. Das neue Framework Flutter der Firma Google tritt mit dem Versprechen an, die Entwicklung für beide Plattformen mit nur einer Codebasis ohne die üblicherweise damit verbundenen Kompromisse bei Aussehen oder Performanz der resultierenden Apps möglich zu machen. Diese Arbeit widmet sich der Frage, inwieweit sich Flutter zur Entwicklung professioneller Cross-Plattform-Apps eignet. Dazu wird für das Online-Magazin FINK.HAMBURG eine App beispielhaft mit Flutter umgesetzt. Das entwickelte System, der Entwicklungsprozess und das Framework werden anschließend evaluiert.

Leonard Thiele

Title of the paper
Cross platform development of a mobile application with Flutter – Concept and prototypical realisation

Keywords
Cross Platform Development, Flutter, Dart, Android, iOS

Abstract
Developing mobile apps for Android and iOS is a technically complex and expensive endeavor. The new framework Flutter by Google Inc. is trying to enable development for both platforms using a single codebase without the usually necessary compromises with the look and feel or performance of the resulting apps. This document deals with the question to what extent Flutter can be used to develop professional cross-platform apps. For this purpose an app for the online magazine FINK.HAMBURG is implemented using Flutter. The developed system, the process of creating it and the framework are evaluated afterwards.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

In der Einführung soll es darum gehen, einen Einstieg in das Thema zu geben und die Motivation der Arbeit zu erläutern. Diese Arbeit beschäftigt sich mit der Cross-Plattform-Entwicklung einer mobilen Anwendung mit Hilfe von Flutter, einem Framework für derartige Projekte. Cross-Plattform bedeutet in diesem Kontext, dass eine in einer bestimmten Programmiersprache geschriebene Anwendung auf mehrere Plattformen portiert und ausgeführt werden kann.

1.1 Motivation

Die Entwicklung professioneller mobiler Apps ist nach wie vor aufwendig und teuer. Steht ein Unternehmen heute vor der Aufgabe, eine mobile Anwendung zu entwickeln, müssen de facto zwei Apps entwickelt werden. Das liegt daran, dass sich im Betriebssystemmarkt für mobile Geräte inzwischen zwei große Lager herauskristallisiert haben. So muss jeder Dienst, der alle Zielgruppen erreichen will, eine App für Android und eine für iOS entwickeln. Damit ist ein hoher Kosten- und Organisations- sowie Wartungsaufwand verbunden. In den letzten zehn Jahren der mobilen Anwendungsentwicklung gab es bereits viele Ansätze, die Entwicklung für beide Plattformen zusammenzuführen. Eine Auswahl dieser Frameworks wird im zweiten Kapitel vorgestellt. Dennoch ist es immer noch oft so, dass native Apps eine bessere Erfahrung für den Benutzer bieten. Das kann sich in einem mit den Bedienelementen der jeweiligen Plattform konsistenteren Design der App äußern. Auch die Performanz und die Darstellung von 3D-Grafik ist bei nativ entwickelten Apps oftmals besser.

Das neue Framework Flutter [1] von Google Inc. stellt einen neuen interessanten Kandidaten für die Realisierung von Cross-Plattform-Projekten dar. Es hat einige bemerkenswerte Ansätze zu bieten und daher das Potential, zu einer Alternative zu nativer und bereits bestehender Cross-Plattform-App-Entwicklung zu werden.

1.2 Zielsetzung

In dieser Arbeit soll eine qualifizierte Aussage zur Eignung von Flutter für Cross-Plattform-Entwicklung getroffen werden. Zu diesem Zweck wird eine mobile Anwendung entwickelt, Erfahrung mit dem Framework gesammelt und das genutzte Werkzeug evaluiert. Im Rahmen dieser Arbeit wird die mobile Anwendung unter Android entwickelt und getestet.

1.3 Aufbau der Arbeit

Im zweiten Kapitel findet sich zunächst eine Einführung zu FINK.HAMBURG, dem Auftraggeber der mobilen Applikation. Hier wird die fachliche Umgebung und die Motivation für die Entwicklung einer eigenen App vorgestellt. Danach wird auf einige der bestehenden und am Markt relevanten Lösungen für die Entwicklung mobiler Cross-Plattform-Apps eingegangen. Hier wird ein Überblick vermittelt, welche Stärken und Schwächen die jeweiligen Frameworks haben. Am Ende des zweiten Kapitels erfolgt dann der Vergleich mit Flutter. Im dritten Kapitel geht es um die Anforderungen an das zu entwickelnde System und an die Entwicklung mit Flutter. Dazu wird zunächst das Anwendungsszenario erläutert. Nach der Schilderung des Analyseprozesses werden die Anforderungen aufgestellt und die daraus folgenden User Stories abgeleitet. Im vierten Kapitel wird das Vorgehen bei der Umsetzung des Projekts vorgestellt. Zur Realisierung des Projekts wird in Etappen gearbeitet. Kapitel fünf beschäftigt sich mit der ersten Iteration. Es wird behandelt, wie der erste Prototyp des Systems geplant und konzipiert wurde. Dabei werden unter anderem die technischen Entscheidungen dargelegt und begründet. Am Schluss des fünften Kapitels wird die tatsächliche Umsetzung beschrieben. Das sechste Kapitel ist wie das fünfte aufgebaut und schildert die zweite Iteration. Nach den beiden Iterationen findet in Kapitel sieben die Evaluation statt. Hier wird die Flutters Tauglichkeit für Cross-Plattform-Entwicklung bewertet. Dies geschieht zum einen konkret anhand des realisierten Projekts. Zum anderen wird auf einer abstrakten, allgemeinen Ebene über die typischen Anforderungen an ein Cross-Plattform-Framework geurteilt. Zum Schluss der Evaluation wird Flutter einer Nutzwertanalyse unterzogen. Hier wird Flutter für verschiedene Anwendungsszenarien bewertet. Zum Schluss wird im achten Kapitel ein Fazit aus dem Vorhaben gezogen. Das entwickelte System wird reflektiert und es wird ein Ausblick auf mögliche Weiterentwicklungen gegeben. Am Schluss des Fazits wird Flutters mögliche Zukunft thematisiert.

2 Grundlagen

In diesem Kapitel wird zunächst der Auftraggeber des Systems genauer vorgestellt. Danach folgt eine Übersicht aktuell gebräuchlicher Frameworks für die Cross-Plattform-Entwicklung mobiler Apps. Es werden der technische Unterbau, die Vorzüge und die Probleme der Lösungen thematisiert. Am Ende der Übersicht wird Flutter vorgestellt und mit den anderen Varianten verglichen.

2.1 FINK.HAMBURG

FINK.HAMBURG ist ein Online-Magazin für Lokalnachrichten aus dem Großraum Hamburg. Publiziert wird es von Studierenden der Hochschule für Angewandte Wissenschaften Hamburg, die den Master-Studiengang *Digitale Kommunikation* studieren. Diese arbeiten mehrere Stunden pro Woche parallel zum regulären Vorlesungsbetrieb an den Beiträgen. Vorbild für dieses Verfahren sind sogenannte *teaching hospitals* aus den USA [2]. Dort können die Studierenden unter realistischen Bedingungen ihr gelerntes journalistisches Handwerkszeug anwenden und trainieren. Der Newsroom, also der Arbeitsraum für die Artikel des Mediums, befindet sich auf dem Kunst- und Mediencampus der HAW Hamburg in der Finkenau. Die Beiträge sind unter der Webadresse https://fink.hamburg zu finden. Dort werden mehrmals pro Woche neue Beiträge veröffentlicht. Manche sind klassische News, andere sind Hintergrundberichte. Es gibt reine Textartikel ohne interaktive Inhalte und welche mit Quiz. Außerdem produziert FINK.HAMBURG Videos als alleinstehende Inhalte oder auch als Begleitung zu Artikeln. Die Beiträge sind in Ressorts aufgeteilt. Die Ressorts sind *Mensch & Umwelt, Technik, Kultur, Politik* sowie *Wirtschaft & Soziales*.

So wie viele Medienschaffende heutzutage haben auch die Mitglieder der Redaktion von FINK.HAMBURG den Stellenwert sozialer Medien und digitaler Vertriebskanäle im Allgemeinen erkannt. Die auf der Website veröffentlichten Artikel werden bisher nur via Twitter und Facebook verbreitet. Eine weitere Möglichkeit zur Verteilung der veröffentlichen Inhalte stellen Push-Benachrichtigungen in einer eigenen App dar. Diese hat den Vorteil, dass sie

näher an den Nutzenden ist als eine normale Website. Sie erhöht die öffentliche Sichtbarkeit und Nutzerbindung an das Medium. Außerdem kann sie neue Erkenntnisse im Umgang mit journalistischem Handwerkszeug liefern. Die Summe dieser Vorteile führte dazu, dass FINK.HAMBURG Interesse an der Entwicklung bekundet hat.

2.2 Aktuelle Cross-Plattform-Frameworks

Für den Kontext dieser Arbeit werden die größten und ausgereiftesten Möglichkeiten zur Cross-Plattform-Entwicklung mobiler Anwendungen verglichen und dann in Bezug zu Flutter gesetzt. Diese Frameworks kommen oftmals von großen Firmen und nicht aus dem akademischen Bereich. Daher finden sich viele Quellen, die zwar umfassend informieren, aber von wirtschaftlichen Interessen beeinflusst sein können. Der folgende Vergleich ist also mit einem gewissen wissenschaftlichen Abstand zu lesen.

2.2.1 Xamarin

Xamarin [3] wurde 2011 gegründet und gehört seit 2016 zu Microsoft. Das Projekt ist open source und steht unter der MIT-Lizenz [4]. Zur Entwicklung bietet sich als Entwicklungsumgebung *Visual Studio* an, welches von Microsoft für größere Teams kommerziell vertrieben wird. Mit der *Xamarin Platform* lassen sich in C# geschriebene Anwendungen auf alle großen mobilen Betriebssysteme portieren. Dies wird möglich, da ein gewisser Teil des *.NET Frameworks* von Microsoft für Android und iOS implementiert ist. Diese Komponenten heißen *Xamarin.iOS* und *Xamarin.Android*. Durch diese Wrapper ist beispielsweise die Nutzung eines Fingerabdrucksensors im C#-Code möglich.

Für die Programmierung grafischer Benutzeroberflächen gibt es im Kontext des Frameworks *XAML*, eine von der jeweiligen Zielplattform abstrahierte Beschreibungssprache. Mit *Xamarin.Forms* ist es möglich, aus dem XAML-Code native Bedienelemente zu erzeugen. Die Standardkomponenten werden also generisch beschrieben und dann von Xamarin in iOS- respektive Android-Widgets umgewandelt. Zusätzlich sind eigene Ergänzungen möglich, da Xamarin native Projektdateien generiert. Diese können dann mit plattformspezifischem Code verfeinert werden. Teilweise sind kleine Anpassungen des automatisch erzeugten Codes sogar unvermeidbar. Das macht bereits einen Nachteil von Xamarin klar: Es kann zwar mit nur einem Team gemeinsam an der Entwicklung der App für Android und iOS gearbeitet werden, trotzdem muss bei den Entwicklern plattformspezifisches Know-how vorhanden sein.

Ein weiterer Faktor sind Betriebssystem-Updates von Android und iOS. Diese ziehen oft Anpassungen an systeminternen oder die Hardware betreffenden Schnittstellen nach sich. Das bedeutet, dass auch alle Wrapper und Adapter aktuell gehalten werden müssen, die Xamarin nutzt. Dadurch hängt die Unterstützung neuester Features wie beispielsweise Apples *FaceID* in Xamarin gegenüber den nativen SDKs in der Regel etwas nach.

2.2.2 Ionic

Ionic [5] wird seit 2012 von der Firma Drifty entwickelt und ist ein Open-Source-Webframework unter der MIT-Lizenz [4]. Zusätzlich zum eigentlichen Framework vertreibt Drifty einige kommerzielle Services wie zum Beispiel ein grafisches Werkzeug zur Erstellung von Ionic-Benutzeroberflächen. Mit Ionic erstellte Apps sind im Kern Websites, die auf dem Gerät installiert sind und im Gegensatz zu herkömmlichen Websites Zugriff auf die nativen Komponenten des Geräts haben. Das hat den großen Vorteil, dass für die Entwicklung von Apps mit Ionic auch Webentwickler eingesetzt werden können, die wenig bis keine Erfahrung mit der Entwicklung nativer Apps haben.

Ionic zugrunde liegen zwei weitere große Frameworks, Googles *Angular* [6] für Unterstützung bei der Erstellung von Geschäfts- und Oberflächenlogik der „Website“ sowie *Apache Cordova* [7] für den Zugriff auf native Komponenten wie den Geosensor des ausführenden Geräts. Zum Einsatz kommen HTML5, CSS, Sass und TypeScript. Letzteres bietet gegenüber JavaScript viele Vorteile in der Entwicklung. Unter anderem durch eine statische Typisierung werden viele Fehler im Code schon vor der Laufzeit der App und damit früher erkennbar.

Die Anbindung der App an native Funktionen erfolgt über Cordova-Plugins, die das gleiche Problem wie Xamarin offenbaren. Im Falle eines Systemupgrades müssen die Entwickler der entsprechenden Plugins schnell nachziehen, da es möglicherweise Änderungen an den relevanten Schnittstellen gibt oder neue hinzukommen. Außerdem kann es passieren, dass für eine bestimmte gewünschte Funktion wie die Integration von Kartenmaterial in die App kein Plugin verfügbar ist.

Ein weiteres Problem ist auch bei Ionic die Performanz der resultierenden App. Wird aus dem Webcode eine native Funktion aufgerufen, kommt es zu einem Kontextwechsel. Der TypeScript-Code ruft für die jeweilige Plattform spezifischen Code auf, der wiederum das Ergebnis zurückliefert. Dieser Schritt kostet Zeit und ist bei anspruchsvollen Apps oft vonnöten. Das macht sich zum Beispiel beim initialen Start einer mit Ionic erstellten App bemerkbar. Diese braucht einen Moment für das Laden der Webkomponenten. Auch Entwicklung von

Apps mit 3D-Inhalten wird so erschwert, da diese im Allgemeinen keine Kompromisse bei der Performanz machen können. So sind native Apps bei einem direkten Vergleich oftmals schneller als die mit Ionic erstellten Pendants [8].

2.2.3 React Native

React Native [9] wurde 2015 von Facebook vorgestellt und ist open source unter der MIT-Lizenz [4]. Es bringt Facebooks Webframework *React* [10] auf mobile Betriebssysteme. React Native hat konzeptuell große Ähnlichkeiten mit Ionic. Webentwickler mit Erfahrung in React werden mit React Native auch ohne vorherigen Kontakt mit nativer Entwicklung schnell produktiv. Die erstellten Apps werden statt TypeScript mit JavaScript geschrieben. Im Gegensatz zu Ionic verwendet React Native bei der Zeichnung der Benutzeroberfläche native Komponenten. Diese sind spürbar schneller als die Webelemente, die bei Ionic zum Einsatz kommen. Zur Anbindung plattformspezifischer Features nutzt auch React Native eine sogenannte „Brücke“ von der JavaScript- in die native Welt. Diese kostet zwar Performanz, wird aber dank der nativen Widgets für die grafische Oberfläche nicht so oft benötigt wie beispielsweise bei Ionic. Die Oberfläche erfordert wegen der nativen Einbindung aber mehr Anpassungen von plattformspezifischem Code. Ionic umgeht dieses Problem, indem direkt eigene Widgets verwendet werden, die sich nicht an den nativen orientieren.

2.2.4 Flutter

Alle bisher vorgestellten Frameworks zur Cross-Plattform-Entwicklung versuchen durch eine gemeinsame Codebasis die Abstraktion von der jeweiligen Plattform zu realisieren. Die Nutzung einer gemeinsamen Sprache wie C# oder JavaScript hat dabei aber den Nachteil, dass sie auf Android und iOS nicht nativ unterstützt wird. Daher wird der Code der vorgestellten Lösungen entweder *just in time* kompiliert oder zur Laufzeit interpretiert. Das führt zwangsläufig zu Einbußen bei der Laufzeitperformanz gegenüber *ahead of time* kompiliertem, nativem Code. So werden native Android-Apps, die in Java oder Kotlin programmiert werden, vor der Laufzeit zu Java-Bytecode kompiliert. Dieser Bytecode läuft dann auf den mobilen Geräten auf der *Java Virtual Machine*, die über viele Jahre hinweg optimiert wurde.

Flutter [1] wurde 2017 von Google ins Leben gerufen, ist open source und BSD-lizenziert [11]. Hier wird versucht, viele der typischen Probleme vorher genannter Frameworks bereits beim Design anzugehen. Zum Einsatz kommt hier Dart, eine statisch typisierte Skriptsprache, die

2011 vorgestellt wurde. Dart wird von Google weiterentwickelt und kann auch in der Webentwicklung eingesetzt werden, da es die Transpilation nach JavaScript unterstützt. Während der Entwicklung wird auf dem Testgerät oder Emulator eine Dart-VM ausgeführt, die den zu bearbeitenden Code just in time kompiliert. Dadurch kann dieser bei Änderungen schnell nachgeladen werden. Dieses sogenannte *Hot Reload* verkürzt die Entwicklungszyklen dramatisch und ermöglicht, dass der Entwickler die gemachten Änderungen schnell überprüfen kann. Geht die App in den produktiven Einsatz, wird der Code dann ahead of time zu nativem ARM-Code kompiliert. Flutters Engine ist in C++ implementiert und wird auf Android mittels des *Android NDK* [12] und unter iOS mit Hilfe von *LLVM* (Low Level Virtual Machine [13]) zu nativem Code kompiliert. Flutter bietet komplett eigene Widgets, deren Rendering von Googles Skia Engine [14] übernommen wird. Dadurch sind Flutter-Apps komplett unabhängig vom Renderstack der jeweiligen Plattform und laufen schneller als vergleichbare just in time kompilierte Systeme.

Flutter folgt dem Prinzip *Composition over inheritance*. Das bedeutet, dass nahezu alle Oberflächenelemente und Funktionen wie Wischgesten als Widget realisiert werden. Die Entwicklungsarbeit abseits der reinen Geschäftslogik funktioniert also maßgeblich über das geschickte Kombinieren und Schachteln verschiedener Typen von Widgets.

2.2.5 Tabellarischer Überblick

	Xamarin	**Ionic**	**React Native**	**Flutter**
Sprachen	C#	HTML, CSS, TypeScript	JavaScript, CSS	Dart
GUI-Elemente	nativ	Webkomponenten	nativ	Flutter Engine/Skia
Performanz	nahe nativ	lange Startzeit	nahe nativ	teils schneller als nativ
Anpassungen[1]	wenig	wenig	viel	wenig

Tabelle 2.1: Übersicht über aktuelle Cross-Plattform-Frameworks [8]

[1] *Vor Fertigstellung der App muss der native Code angepasst werden, um überall korrekt zu funktionieren.*

3 Anforderungsanalyse

3.1 Anwendungsszenario

FINK.HAMBURG besitzt zwar eine responsive Website, aber keine eigene App. Für den modernen Journalismus sind mobile Geräte eine gute Möglichkeit, unkompliziert eine große Zahl von Lesern zu erreichen. Besonders für Eilmeldungen ist es sehr wichtig, den Nutzer direkt per Push-Benachrichtigung auf einen neuen Artikel zu stoßen und so große Aufmerksamkeit zu generieren. Die Push-Benachrichtigungen sind auch auf dem Sperrbildschirm eines mobilen Geräts präsent und sorgen so für maximale Sichtbarkeit der beworbenen Inhalte.

Die Artikel von FINK.HAMBURG liegen auf einem WordPress-Server, der auch für das Ausspielen der Website verantwortlich ist. Intern werden die veröffentlichten Artikel nicht als Plaintext/Klartext abgelegt, sondern im HTML-Format. Die Inhalte des WordPress-Servers lassen sich über eine bereits automatisch verfügbare REST-API direkt von diesem abrufen.

3.2 Analyseprozess

Die Anforderungen an die App stammen vornehmlich aus zwei Quellen: Den Wünschen und Vorstellungen der Redaktion von FINK.HAMBURG und den Anforderungen zur Evaluation des Systems im Rahmen dieser Arbeit.

3.2.1 Analyse der FINK.HAMBURG-App

Zur Analyse und Aufnahme der Anforderungen an die App wurden zunächst Interviews mit den an FINK.HAMBURG beteiligten Personen geführt. Zuallererst wurde mit Prof. Dr. Christian Stöcker, dem Studiengangsleiter des Masters *Digitale Kommunikation*, gesprochen. Mit ihm wurde der Rahmen des Projekts abgesteckt. Es wurde unter anderem vereinbart, über das von den Studierenden verwendete Kommunikationswerkzeug *Just Social* [15] Kontakt zu halten. Gesprochen wurde auch über die Anforderungen an die App. Dazu wurden offene

Fragen gestellt, wie beispielsweise: „Was empfinden Sie an anderen Nachrichtenapps als besonders gelungen?“ Prof. Stöcker war vor allem die Unterstützung von Push-Benachrichtigungen wichtig. Diese sind schließlich das entscheidende Feature, was eine eigene App von der ohnehin vorhandenen mobilen Website abhebt. Danach gab Philipp Kessling, der wissenschaftliche Mitarbeiter von Prof. Stöcker, eine kleine Einführung zur bereits vorhandenen Schnittstelle zum Backend von FINK.HAMBURG.

Zur Erhebung weiterer Anforderungen an die App wurde eine Liste mit Feature-Ideen von den Redaktionsmitgliedern gewünscht. Diese waren zum Teil perspektivische Ideen zur zukünftigen Umsetzung wie etwa ein mögliches Live-Feed-Feature. Der zweite, größere Teile war eine Aufstellung von Eigenschaften anderer Nachrichten-Apps, die sich bewährt haben und bei den Studierenden gut ankommen wie unter anderem eine Gestensteuerung zur Navigation zwischen den Artikeln.

3.2.2 Analyse einer typischen Flutter-App

Die Entwicklung von mobilen Anwendungen wird stetig durch neue Werkzeuge und Frameworks vereinfacht. Gleichzeitig werden aber auch die Anforderungen immer höher, da die digitale Infrastruktur zum Betrieb einer modernen Dienstleistung immer komplexer wird. Im Laufe der vergangenen zehn Jahre haben sich die Werkzeuge und Vorgehensmodelle zur Entwicklung mobiler Apps rasant entwickelt. Auch die mit den inzwischen verfügbaren Libraries und Frameworks erreichbaren Ergebnisse sind inzwischen deutlich ausgereifter. Flutter sollte die üblichen Anforderungen an moderne App-Entwicklung erfüllen, diese werden später in der Arbeit noch konkretisiert.

3.3 Anforderungen

Innerhalb des oben genannten Szenarios haben sich die folgenden Anforderungen zur Realisierung ergeben. Die hochgestellten Zahlen dienen zur Orientierung in der Tabelle 3.1 auf Seite 11.

3.3.1 Funktionale Anforderungen an die App

Die App soll alle reinen Textartikel von FINK.HAMBURG darstellen können[1]. Dazu soll es mehrere verfügbare Ressorts geben, aus denen der Nutzer auswählen kann[2]. Artikel sollen

durch eine Suche gefunden werden können[3]. Es soll in der App eine eigene Kategorie *Videos* geben, die auf die YouTube-Inhalte von FINK.HAMBURG verlinkt[4]. Außerdem soll sie auf Wunsch der Redaktion auf neue Artikel per Push-Benachrichtigung hinweisen können[5]. Der Nutzer der App soll auswählen können, von welchen Ressorts Push-Benachrichtigungen eingehen sollen[6]. Ein Artikel soll als Favorit speicherbar sein, um ihn später schnell wieder zu finden und erneut abrufen zu können[7]. Bei jeder Artikelansicht soll es einen Share-Button geben, der den Link zum Artikel direkt an eine andere App weitergibt. So wird es etwa möglich, Artikel direkt via Messenger-App an Interessierte weiterzuleiten[8]. In der Artikelansicht soll durch eine Wischgeste zur Seite der nächste Artikel des gleichen Ressorts angezeigt werden[9].

3.3.2 Nichtfunktionale Anforderungen an die App

Die App sollte auf Android und iOS voll funktionsfähig sein[10]. Das System soll in Zukunft mit einer wachsender Zahl von Artikeln gut skalieren können. Dafür ist es wichtig, dass die App auch zu Spitzenzeiten (*peak times*) einen gewünschten Artikel innerhalb von zwei Sekunden anzeigt und das Backend die erhöhte Last bewältigt[11]. Außerdem sollte das System erweiter- und veränderbar sein, damit eventuell nötige Änderungen in der Zukunft kostengünstig machbar sind. So sollte ein neues Ressort, also ein neuer Themenbereich des Mediums, innerhalb von vier Arbeitsstunden in die App einzubauen sein[12].

Gerade im mobilen Bereich ist die Performanz der dargestellten Inhalte Schlüssel zur Aufmerksamkeit des Nutzers. Die Animationen bei Veränderungen der Oberfläche sollen daher nicht mehr als jeweils eine Sekunde in Anspruch nehmen [13]. Die App sollte keinen höheren Speicherbedarf haben als andere durchschnittliche Nachrichten-Apps, das betrifft sowohl den Flash-Speicher als auch den RAM des Geräts[14]. Das Aussehen und Verhalten der App sollte konsistent sein. Es sollte ein einheitliches Farbschema innerhalb der App und zur Website eingehalten werden[15].

3.3.3 Tabelle der Anforderungen an die App

Die eben beschriebenen Anforderungen sind in folgender Tabelle für eine bessere Übersicht zusammengefasst.

1	Reine Textartikel darstellen
2	Auswahl von Artikeln nach Ressorts
3	Suche von Artikeln
4	Verknüpfung zu den Video-Inhalten auf YouTube
5	Push-Benachrichtigungen für neue Artikel
6	Auswahl, zu welchem Ressort Benachrichtigungen empfangen werden sollen
7	Artikel als Favoriten vormerken
8	Teilen-Funktion in der Artikelansicht
9	Wischgesten zur Navigation zwischen Artikeln
10	Voller Funktionsumfang auf Android und iOS
11	Abruf und Anzeige von Artikeln innerhalb von zwei Sekunden
12	Einbau neuer Ressorts dauert nicht nicht länger als vier Arbeitsstunden
13	Dauer der Animationen nicht länger als zwei Sekunden
14	Speicherbedarf nicht größer als bei vergleichbaren Nachrichten-Apps
15	Konsistentes Aussehen und Verhalten

Tabelle 3.1: Tabelle der Anforderungen an die App

3.3.4 Anforderungen an die Entwicklung mit Flutter

Zunächst einmal sollte die Vorlaufzeit zum Erlernen der Grundlagen des Frameworks kurz sein, im Idealfall unter drei Wochen. Eine bekannte oder eine bekannten Sprachen ähnliche Programmiersprache kann die Entwicklung und den Einstieg für den Entwickler vereinfachen. Desweiteren sind eine IDE mit Komfortfunktionen wie eine gute Autovervollständigung hilfreich. Der Zeitraum zwischen dem Neustart der App nach einer Änderung im Code und dem Bereitstehen der Änderungen zur Prüfung durch den Entwickler ist sehr wichtig.

Weiterhin sollte die Integration von Diensten dritter Parteien in die App ohne größeren Aufwand machbar sein. Das schließt vor allem die Anbindung von Netzwerkdiensten, aber auch die Einbindung von Bibliotheken ein. Flutter sollte den Abruf von Daten von einem Server so unkompliziert wie möglich gestalten ohne dabei gegenüber vergleichbaren Lösungen wie *Retrofit* [16] an Funktionalität einzubüßen.

Das Integrieren von Werbung ist ein wichtiger wirtschaftlicher Faktor bei der Entwicklung mobiler Applikationen. Vor allem bei Cross-Plattform-Apps sollte eine zentrale Anforderung wie das Einbauen von Werbung keine plattformspezifischen Anpassungen nötig machen. Zuletzt sind die Komponenten zur Erstellung der Oberfläche wichtig. In Bezugnahme auf Anforderung 15 hat der Kunde den Wunsch geäußert, die App solle „sexy“ sein. Diese Aussage muss natürlich noch operationalisiert werden. Flutter sollte ausreichende Gestaltungsmöglichkeiten bieten, um attraktive Oberflächen zu programmieren.

3.4 User Stories

Aus den vorher genannten Anforderungen ergeben sich folgende User Stories:

1. Ich, als Leser/-in, kann alle Artikel anzeigen.
    - Ausgangspunkt für die App
    - Mögliche Testfälle:
        - Das Gerät hat keine Internetverbindung
        - Der Server von FINK.HAMBURG antwortet nicht
        - Der Server von FINK.HAMBURG antwortet sehr langsam
2. Ich, als Leser/-in, kann ein Ressort auswählen, aus dem dann Artikel angezeigt werden.
    - Kann über eine Seitenleiste realisiert werden
    - Mögliche Testfälle:
        - Kein Ressort auswählen
        - Ein anderes Ressort auswählen, während noch Artikel des alten geladen werden
3. Ich, als Leser/-in, kann nach Artikeln suchen.
    - Suche nach Titel oder Inhalt eines Artikels denkbar
    - Mögliche Testfälle:
        - Nach einem leeren Text suchen
        - Nach einem Text mit einer Million Zeichen suchen
        - Nach einem Text mit einem Zeichen suchen, das nicht Unicode-konform ist
4. Ich, als Leser/-in, habe in der App Zugriff auf die Video-Inhalte auf YouTube.
    - Kann ein eigener Bereich der App sein und/oder beim gegebenenfalls passenden Artikel verlinkt werden
5. Ich, als Leser/-in, kann mir Artikel als Favoriten vormerken und später abrufen.
    - Bei der jeweiligen Artikelansicht wird ein Stern oder ähnliches angezeigt
    - Ein Tippen auf den Button speichert den Artikel als Favoriten

- Vorgemerkte Artikel werden in einem eigenen Bereich angezeigt
- Mögliche Testfälle:
  - Artikel als Favorit markieren und gleich wieder entfernen
  - Mehr als tausend Artikel favorisieren und dann abrufen

6. Ich, als Leser/-in, kann einen Artikel über die Teilen-Funktion verbreiten.
   - Das Teilen wird allgemein gehalten, die Weitergabe an die entsprechende App übernimmt das jeweilige Betriebssystem
   - Mögliche Testfälle:
     - Teilen via Messenger-App
     - Teilen als reiner Text (Link wird kopiert)
7. Ich, als Leser/-in, kann zwischen den Artikeln eines Ressorts hin und her wischen.
   - Mögliche Testfälle:
     - Mit zwei Fingern wischen
8. Ich, als Leser/-in, bekomme Push-Benachrichtigungen für neue Artikel.
   - Mögliche Testfälle:
     - Altes Gerät als Empfänger benutzen
9. Ich, als Leser/-in, kann wählen, zu welchem Ressort ich Benachrichtigungen bekomme.
   - Wird in den Einstellungen der App verankert
10. Ich, als Autor/-in, kann Nutzern der App eine Push-Benachrichtigung zukommen lassen.
    - Mögliche Testfälle:
      - Eine leere Push-Benachrichtigung abschicken

4 Vorgehensmodell

In diesem Kapitel wird erläutert und begründet, wie bei Analyse, Entwurf und Umsetzung des Systems vorgegangen wurde.

4.1 Auswahlkriterien

Da nur ein Entwickler mit der Arbeit betraut ist, sollte das Vorgehensmodell nicht allzu viel Management erfordern, sonst bleibt keine Kapazität für die eigentliche Entwurfs- und Implementationsarbeit. Der Auftraggeber FINK.HAMBURG hat bis auf das Feature der Push-Benachrichtigungen keine kritischen Anforderungen an das System. Der Spielraum für das Konzept des Systems ist also groß und die Umsetzung kann relativ frei und flexibel erfolgen. Ein Vorteil davon ist, dass die mangelnde Erfahrung in der Entwicklung mit dem Flutter-Framework berücksichtigt werden kann. Diese stellt ein Risiko dar, das sich schlecht einschätzen lässt. Daher sollte das Vorgehensmodell eine schnelle Reaktion auf ungeplante Verzögerungen oder Hindernisse bei der Implementation ermöglichen.

4.2 Agiler Ansatz

Für die Umsetzung des Projekts wird ein agiler Ansatz gewählt. Dafür bietet sich Kanban an [17], welches im Gegensatz zu anderen Vorgehensweisen wie Scrum [18] keine Rollen vorschreibt. Die wären bei einem Ein-Mann-Projekt ohnehin nur unzureichend abgedeckt und trügen somit nicht zum Projekterfolg bei. Zum Verfolgen der Arbeit wird ein Kanban Board verwendet, auf dem alle Arbeitspakete abgetragen sind. Anders als bei Scrum mit seinen regelmäßigen Sprints ist die Arbeit bei Kanban ein kontinuierlicher Fluss, der Änderungen an der Planung sofort integrieren kann. Die Pakete werden gemäß ihres Status von den Spalten „To Do“ über „In Arbeit“ zu „Fertig“ geschoben. Dieses Vorgehen kann durch ein sogenanntes *WIP-Limit* erweitert werden. Hierbei wird die Zahl der Karten, die gleichzeitig in der Spalte

„To Do“ stehen, begrenzt. Das stellt sicher, dass nicht zu viel Arbeit gleichzeitig begonnen wird und es entsteht keine größere Menge an unvollendeten Aufgaben. Außerdem werden so den Fluss blockierende Arbeitspakete schnell sichtbar. Auf diese Blocker kann dann reagiert werden. So wäre denkbar, der Erledigung der Aufgabe mehr Ressourcen zuzuweisen oder den gewählten Ansatz grundsätzlich zu überdenken.

Zusätzlich zu Kanban als Rahmenwerk kommt in diesem Projekt die Technik des *Rapid Prototyping* zum Einsatz. Nach der Analyse des Szenarios und der Prozesse wird ein erster Prototyp erstellt, der die Grundfunktionalität des Systems abdeckt und sukzessiv erweitert wird [19]. Durch das schnelle Feedback zu einem frühen Zeitpunkt im Projekt ist die Überprüfung der gesetzten Ziele möglich, bevor unnötige Arbeit in nicht benötigte Funktionen gesteckt wird. Das inkrementelle Vorgehen erleichtert außerdem den Umgang mit dem neuen Framework, da Fehler einkalkuliert werden.

Im vorliegenden Projekt wird Kanban mithilfe von Trello [20] umgesetzt. Trello ist für kleine Projekte kostenlos und lässt sich über eine Website sowie eine App nutzen. Es ist leicht einzurichten und sehr gut geeignet zur Realisierung von Kanban-Boards. Für dieses Projekt wurde auf die „Fertig“-Spalte verzichtet. Dadurch wird Platz gespart, denn die Spalte wird ohnehin nach getaner Arbeit selten genutzt. Abgearbeitete Karten landen in Trellos Archivfunktion, wo sie später erneut abrufbar sind. Hier kann bei Bedarf die erledigte Arbeit nachgelesen oder reflektiert werden.

Zusätzlich zu den von Kanban vorgeschriebenen Spalten gibt es eine mit dem Titel „Ideen“. Hier werden Visionen für eine künftige Weiterentwicklung der App gesammelt, die noch nicht entwickelt werden sollen. Die „Ideen“-Spalte fungiert in diesem Fall als eine Art Backlog, welches aus den genannten Vorgehensmodellen bekannt ist.

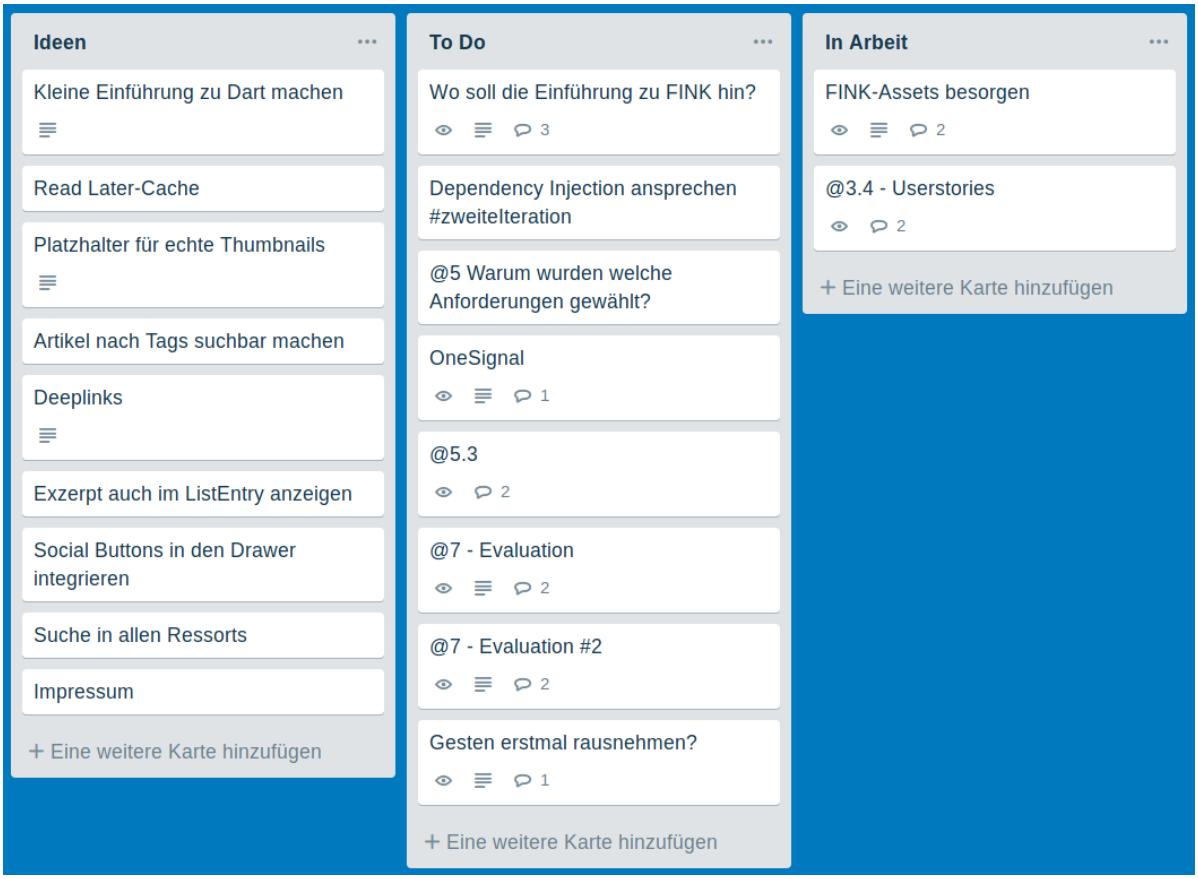

Abbildung 4.1: Ausschnitt aus dem Kanban Board des Projekts

5 Erste Iteration

In der ersten Iteration soll es darum gehen, eine erste Grundfassung des Systems zu planen und umzusetzen. Dazu wird zunächst erläutert, warum bestimmte Funktionen des Systems einbezogen wurden und andere nicht. Außerdem wird das Konzept der Applikation vorgestellt und soweit nötig auch technisch erläutert.

5.1 Planung

Das Ziel der ersten Iteration ist ein lauffähiger Prototyp, der die Kernanforderungen an die App erfüllt und einen Einblick in die Entwicklung mit Flutter möglich macht. Die Kernanforderungen an die App sind die Push-Benachrichtigungen als Hinweis auf neue Artikel und die Anzeige der Artikel von FINK.HAMBURG. Die Push-Benachrichtigungen sind ein Feature, das die Entwicklung einer App für FINK.HAMBURG gegenüber der bereits vorhandenen Website besonders attraktiv macht. Das Feature macht zwei Arbeitsschritte notwendig: Die Integration der Empfängerseite in der App und das Bereitstellen der Senderseite, etwa über einen Webservice. Der Service zum Versenden einer neuen Benachrichtigung von außen verdient genauere Betrachtung, da er später von der Redaktion von FINK.HAMBURG genutzt werden soll. Daher muss er auch für technisch weniger kundige Nutzer eine angemessene Bedienung ermöglichen. Nach der ersten Iteration soll es also ein Treffen mit der Redaktion geben, um die Qualität und Benutzbarkeit der vorläufigen Lösung zu prüfen.

Die Anzeige von Artikeln ist der wichtigste Anwendungsfall der App und stellt eine Selbstverständlichkeit dar. Vor allem für das Sammeln von Erfahrung mit der Programmierung von grafischen Oberflächen ist dieses Feature gut geeignet und wird daher frühzeitig evaluiert.

5.2 Konzept

Im Folgenden soll erläutert werden, wie die technische Lösung der im dritten Kapitel beschriebenen Anforderungen im ersten Prototypen aussehen soll. Dieser Prototyp soll mit dem Auftraggeber evaluiert werden. Sollten große Änderungen an den festgelegten Zielen und deren Umsetzung nötig sein, werden diese in der zweiten Iteration angegangen.

5.2.1 Anwendungsfälle

Der erste Anwendungsfall ist die Ansicht sämtlicher verfügbarer Artikel. Dazu wird beim Start der App eine kurze Liste von Kacheln geladen. Diese Kacheln enthalten ein Vorschaubild und den Titel des Artikels. Erreicht der Nutzer das Ende der Liste, wird sie dynamisch erweitert und es entsteht eine endlose Aufstellung aller Artikel von FINK.HAMBURG. Tippt der Nutzer auf eine der Kacheln, wird der entsprechende Artikel zum Lesen geladen.

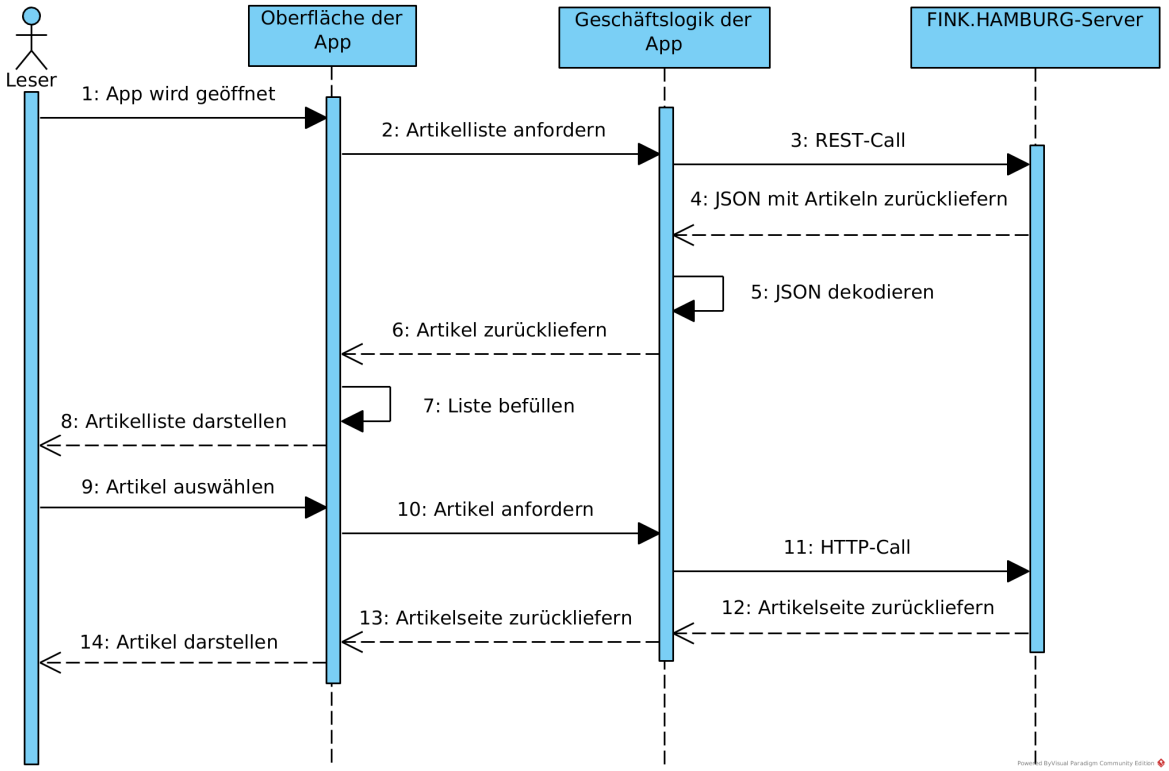

Abbildung 5.1: Sequenzdiagramm des ersten Anwendungsfalls

Der zweite wichtige Anwendungsfall ist die Push-Benachrichtigung über einen neuen Artikel. Dazu stößt ein Mitglied der Redaktion auf einer Weboberfläche den Versand der Benachrichtigung an. Diese wird dann an die Nutzer zugestellt und führt zurück in die App.

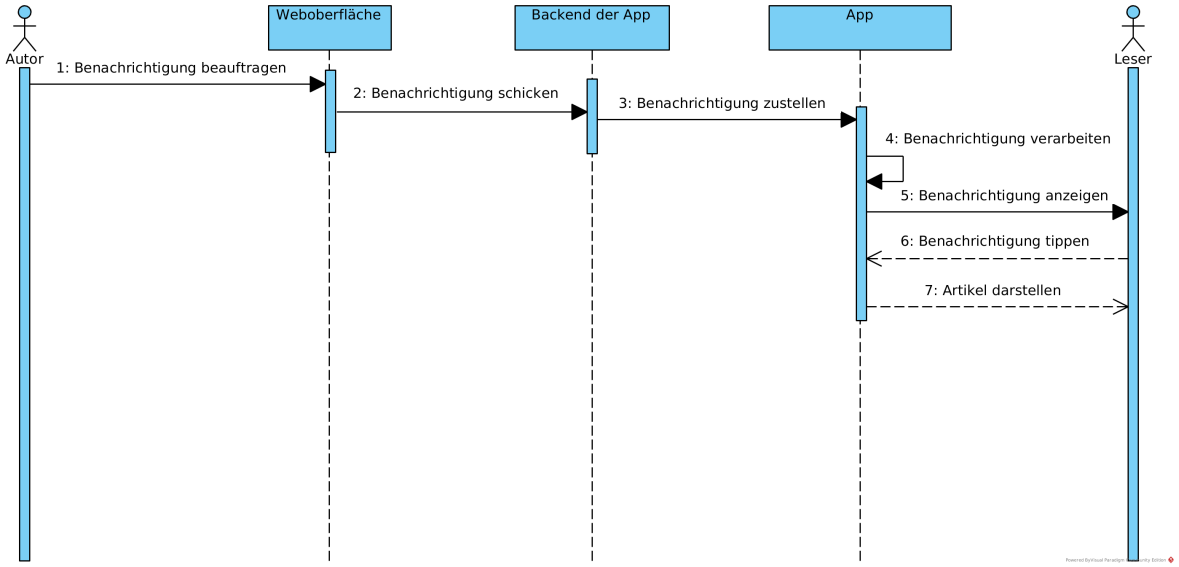

Abbildung 5.2: Sequenzdiagramm des zweiten Anwendungsfalls

5.2.2 Fachliches Datenmodell

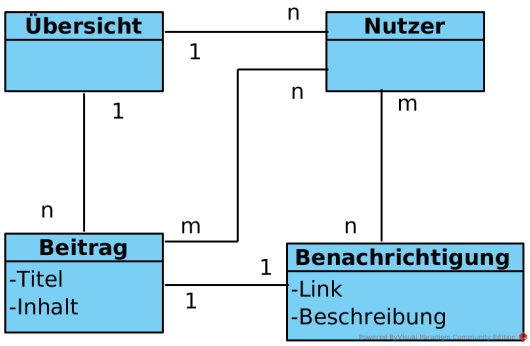

Abbildung 5.3: Fachliches Klassendiagramm des Systems, erste Iteration

Die abzubildende Fachlichkeit des Systems ist überschaubar. Jeder Nutzer bekommt eine Übersicht zu sehen. Der Nutzer kann Benachrichtigungen bekommen. Diese Benachrichtigungen erreichen wiederum mehrere Nutzer. Eine Benachrichtigung enthält einen Link zum

beworbenen Beitrag und einen dafür werbenden Beschreibungstext. Die Übersicht besitzt eine variable Zahl an Beiträgen. Ein Beitrag besteht aus seinem Titel und dem Inhalt. Er kann ein Quiz, Videos und/oder Texte beinhalten. Jeder Nutzer konsumiert eine variable Zahl von Beiträgen. Die Beiträge können von einer beliebigen Zahl von Nutzern betrachtet werden.

5.2.3 Technische Entscheidungen

Entwurfsmuster

Die App wird nach dem *BLoC-Pattern* entworfen, welches auch intern von Google zur Entwicklung von Flutter-Apps verwendet wird [21]. BLoC steht für *Business Logic Components* und folgt etablierten Entwurfsmustern wie *Model View Controller (MVC)* oder *Model View Presenter (MVP)*. Sie alle haben das Ziel, die Kopplung zwischen der für die Geschäftslogik und der Darstellung der Anwendung zuständigen Teile zu verringern. Das macht die einzelnen Komponenten leichter zu testen und vereinfacht die Entwicklung an verschiedenen Stellen der App mit mehreren Entwicklern.

Alle diese Entwurfsmustern haben gemein, dass sie die Anwendung in drei Bereiche unterteilen. Die oberste Schicht ist für die Darstellung der Benutzeroberfläche (View/Widget) verantwortlich und kümmert sich um die visuelle Aufbereitung der von der Geschäftslogik bereit gestellten Daten. Auf der untersten Schicht findet sich die Repräsentation der domänenspezifischen Daten (Model). Dazwischen, in der mittleren Schicht, liegt die Geschäftslogik (Controller/Presenter/BLoC). Sie ist zum Beispiel für den Abruf von Daten aus dem Internet verantwortlich und setzt die Benutzereingaben in Aktionen um. Hier unterscheiden sich die drei Pattern am deutlichsten. Ein Controller kann mehreren Views zugeordnet sein und ist je nach Auslegung des Patterns auch für das Erstellen der Views verantwortlich. Ein Presenter hingegen fungiert in der Regel nur als Mittelsmann zwischen der obersten und untersten Schicht und beinhaltet keinen Code zur Darstellung.

Das BLoC-Pattern erweitert diese Regeln: Eine BLoC funktioniert prinzipiell wie ein Presenter, die Kommunikation der verschieden Komponenten ist aber nur über sogenannte *Streams* erlaubt. Jeder Stream verarbeitet *Events*, die von einer Komponente an beliebig viele andere geschickt werden. Dieses Konzept kommt ursprünglich aus der reaktiven Programmierung [22] und ist zum Beispiel in der Frameworkfamilie *ReactiveX* [23] umgesetzt. Es bietet den Vorteil, dass Nutzereingaben oder Serverantworten alle ihre eigenen Streams bekommen und die Komponenten sehr feingranular vernetzt werden können. Reaktive Programmierung ist ursprünglich aus dem Zweck heraus entstanden, asynchrone Programmabläufe besser

kontrollieren zu können. Sender und Empfänger sind dabei oft zustandslos. Was mit gesendeten Events passiert, ist für den Sender nicht von Relevanz. Die Empfänger reagieren auf eingehende Events, ohne konkrete Kenntnisse von den vorgelagerten Prozessen zu haben. Beide Seiten haben nur eine Referenz auf den Stream, nicht aber auf die jeweilige Gegenstelle. Dadurch sind die Komponenten der App schwächer gekoppelt und es ist eine hohe Kohäsion möglich.

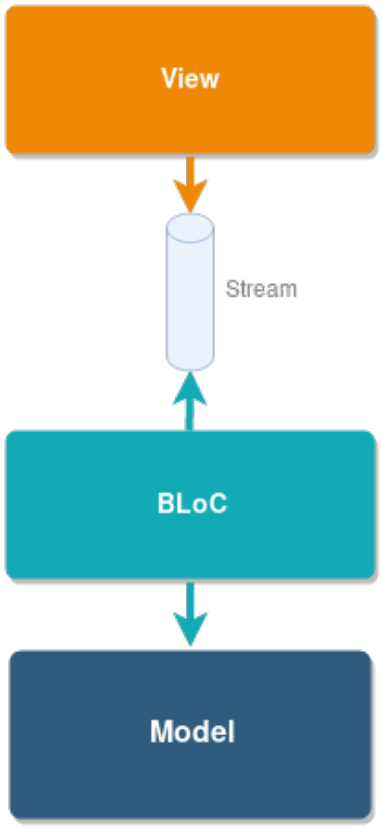

Abbildung 5.4: Das BLoC-Pattern

Anzeige der Beiträge

Die Darstellung der Beiträge in der App gestaltet sich schwierig. Die Artikel liegen im HTML-Format vor, das um WordPress-Plugins erweitert wird. Flutter unterstützt zum Zeitpunkt des Projekts kein natives Rendern von HTML und auch keine WordPress-Plugins. Zum Anzeigen der Beiträge in einem nativen Widget müsste der Text, der von der WordPress-API kommt, also zunächst in ein Flutter-kompatibles Format gebracht werden. Das könnte in einer Art Adapter-Server geschehen, wäre aber wohl Thema für eine eigene Bachelorarbeit.

Dieser erhebliche Aufwand führt zu einer Zwischenlösung: Die vom Nutzer ausgewählten Beiträge werden nicht nativ als Flutter-Widgets, sondern von einer Webview angezeigt. Diese nutzt den Browser des Geräts, um die Website von FINK.HAMBURG anzuzeigen. Für die Webview bietet Flutter zum jetzigen Zeitpunkt noch keine Umsetzung an. Daher wird ein von der Community entwickeltes Package verwendet.

Push-Benachrichtigungen

Zum Erreichen dieser Kernanforderung bietet Flutter von sich aus nur die Integration von Googles Firebase-Dienst an. Firebase tritt als Backend-as-a-service (*BaaS*) an. Eine mobile App braucht in der Regel einen gewissen Satz an Funktionen, die von einem Backend bereitgestellt werden. Firebase bietet Entwicklern die Möglichkeit, Teile dieser klassischen Backend-Funktionen in die Cloud auszulagern [24]. Dazu zählen unter anderem Datenbanken, Machine Learning und auch Push-Benachrichtigungen. Letztere sind unter der Bezeichnung *Firebase Cloud Messaging (FCM)* im Paket enthalten. Firebase hat den großen Vorteil, dass es sich sehr leicht mit Apps unter Android und iOS integrieren lässt. Die Verbindung zur Zustellung funktioniert sehr systemnah, da Android und bei Bedarf auch iOS eine kontinuierliche Verbindung mit den Firebase-Servern unterhalten können. Allerdings ist das eigentlich benötigte Feature der Benachrichtigungen nur ein kleiner Teil des großen Gesamtpakets. Um das Projekt nicht zu überfrachten, wird Firebase daher nicht integriert. Stattdessen wurde der Dienst der Firma *OneSignal* für die App gewählt. Diese bietet zum jetzigen Zeitpunkt das einzige weitere Cloud-getriebene Flutter-Package an. OneSignal bietet eine kostenfreie Umgebung für Push-Benachrichtigungen und ein Client-SDK, welches in die App eingebunden wird. Über eine dedizierte Weboberfläche kann dann auf einen neuen Artikel hingewiesen werden. OneSignal kümmert sich um die Zustellung der Benachrichtigung und bei einem Tipp auf ebendiese landet der Nutzer direkt auf dem beworbenen Artikel.

Testkonzept

Ein wesentlicher Bestandteil von Kanban in der Softwareentwicklung ist die Automatisierung von Tests. Bei der iterativen Entwicklung ist regelmäßiges Testen des Systems essentiell. Durch die Automatisierung ist das ohne größeren Aufwand machbar und die entwickelten Komponenten können kontinuierlich mit den Anforderungen abgeglichen werden. Bei bestehenden Systemen ist es zuweilen schwierig, die Architektur im Nachhinein auf gute Testbarkeit auszurichten. Durch die Wahl des BLoC-Patterns bereits beim Entwurf der Soft-

ware sind die einzelnen Komponenten des Systems schwach gekoppelt. Das vereinfacht das Testen einzelner Teile des Systems enorm [25].

Für das Testkonzept kristallisieren sich drei zu testende Bereiche heraus. Den simpelsten Bereich stellen die *Unit Tests* zur Überprüfung einer einzelnen Methode oder Klasse dar. Zum Bereitstellen benötigter Abhängigkeiten gibt es für die Entwicklung mit Flutter das Paket *Mockito*, welches das benötigte Verhalten imitiert, ohne die oft komplexe Logik der zu erfüllenden Abhängigkeit tatsächlich auszuführen. Zum Testen der BLoCs (Geschäftslogik) via Unit Tests muss die App nicht auf einem physikalischen oder emulierten Gerät laufen, was die Laufzeit der Tests deutlich verkürzt.

Der zweite zu testende Bereich umfasst die Widgets und Logik der Oberfläche (*Komponententest*). Sollen die Widgets getestet werden, können die anzuzeigenden Daten einfach generiert („gemockt“) werden und müssen nicht vom Internet abgerufen werden. Diese vorgetäuschten Daten werden dann über einen Stream an die Benutzeroberfläche gesendet und das Verhalten des Widgets kann sehr zielgerichtet überprüft werden. Die Logik zum Abruf der API wird nicht ausgeführt, was den Test noch kleiner und dadurch besser überschaubar und damit nachvollziehbar macht. Durch die geringe Kopplung lassen sich die Funktionen auch ohne Benutzereingaben vorhersehbar ausführen und testen.

Der dritte Testbereich umfasst die App als Ganzes, es wird auch vom *Integrationstest* gesprochen. Dieser wird in der Regel auf einem echten Gerät oder voll funktionsfähigen Emulator ausgeführt und soll das ordnungsgemäße Zusammenspiel aller Komponenten der App testen. Auch die Performance der App lässt sich so testen. Mit zeitgesteuerten automatisierten Eingaben wird gewährleistet, dass die App stets mit der erwarteten Geschwindigkeit läuft.

5.3 Umsetzung

Grundsätzlich lässt Flutter sehr viel Freiheit bei der Umsetzung einer Architektur. Einige Eigenschaften der Umsetzung ergeben sich allerdings aus dem Framework. So werden private Attribute und Methoden in Flutter pauschal mit einem Unterstrich begonnen. Außerdem gibt es zu jedem Widget, welches einen veränderbaren Zustand hat *(StatefulWidget)* eine Widget- und eine State-Klasse, die oftmals in der gleichen Datei vorliegen. Das findet sich im folgenden Diagramm wieder. Die beiden Seiten *OverviewPage* und *ArticlePage* bestehen jeweils aus der Widget-Klasse und ihrem abhängigen State.

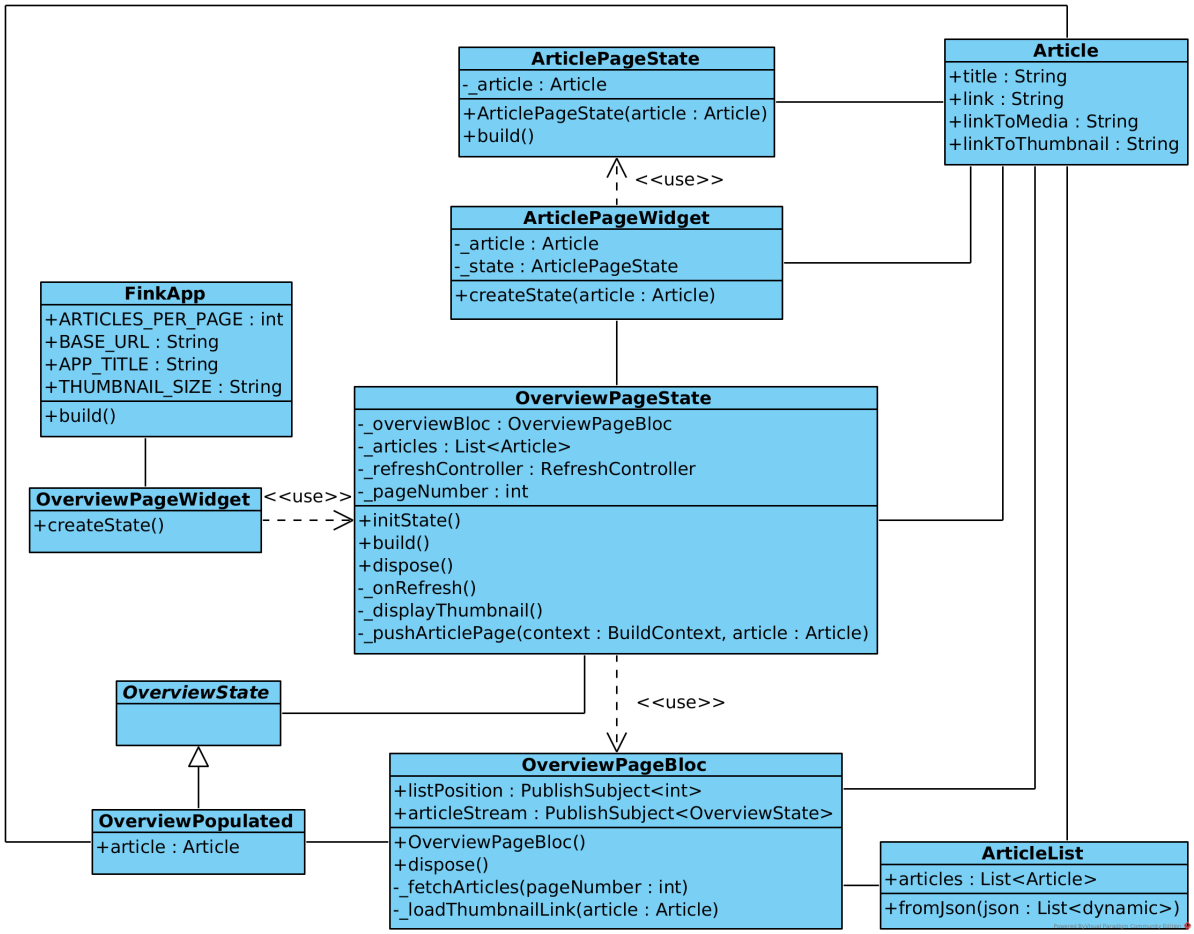

Abbildung 5.5: Technisches Klassendiagramm der ersten Iteration

In der Mitte des Diagramms steht die Model-Klasse Artikel. Sie wird in fast allen Widgets und States verwendet und besteht aus einem Titel, der URL für den Abruf, einem Link zu sämtlichen zugehörigen Medieninhalten und einem Link zum Vorschaubild des Artikels. Letzteres wird in der Übersicht für jeden Artikel angezeigt. In der Hauptklasse FinkApp sind die globalen Konstanten gespeichert, die steuern, mit welchem Server sich die App verbindet, wie groß die Vorschaubilder für die Artikelübersicht sein sollen etc. Außerdem besitzt sie die für Flutter-Widgets typische build-Methode. Hier wird der Empfang von Push-Benachrichtigungen aktiviert und das OverviewPageWidget auf die Oberfläche gezeichnet. Dieses Widget erzeugt bei seiner Erstellung den OverviewPageState. Dieser startet im initState() die OverviewPageBloc und setzt die Streams zur Kommunikation zwischen den Komponenten auf. Die Streams sind vom Datentyp PublishSubject. Dieser Typ repräsentiert einen Stream mit mehreren Empfängern. Bei zwischenzeitigem Schließen der App wird die

dispose-Funktion des Widgets aufgerufen. Diese schließt die Streams in der OverviewPageBloc. Außerdem hält der OverviewPageState eine Liste der anzuzeigenden Artikel in der Hauptansicht und einen RefreshController, der ausgelöst wird, wenn der Nutzer die Übersichtsseite neu lädt. Die geladenen Artikel werden im _onRefresh() zurückgesetzt und neu angelegt.

Klickt der Nutzer auf dem OverviewWidget auf einen Artikel, wird in der Methode _pushArticlePage das ArticlePageWidget gerendert. Dieses enthält eine Webview (nicht dargestellt) und zeigt den gewählten Artikel an. Die OverviewPageBloc hat zwei Streams, über die das Nachladen neuer Artikel in die Übersichtsliste gesteuert wird. Über die listPosition wird die BLoC vom OverviewPageState informiert, welcher Teil der Liste von Artikeln geladen werden soll, das passiert dann in _fetchArticles().

Über den articleStream werden neue OverviewStates an den OverviewPageState gegeben. OverviewState ist eine abstrakte Klasse, die mögliche Verbindungszustände mit dem FINK.HAMBURG-WordPress repräsentiert. Wird von der OverviewPageBloc eine erfolgreiche HTTP-Anfrage an das Backend gemacht, werden die Ergebnisse als konkrete Implementation OverviewPopulated in den articleStream gegeben. In späteren Entwicklungsschritten ist der OverviewState noch durch einen Fehlerzustand bei fehlgeschlagenen HTTP-Anfragen erweiterbar. In der Funktion _fetchArticles() der OverviewPageBloc ist der HTTP-Call realisiert. Das Backend liefert einen JSON-kodierten String zurück, der eine Liste von Artikeln enthält. Dieser wird mittels der Model-Klasse ArticleList zu einem Objekt decodiert. Diese besitzt die Factory-Methode fromJson(), die den JSON-String als Eingabe erwartet. Die ArticleList wird dann in der OverviewPageBloc verwertet und die Artikel werden einzeln als OverviewPopulated verpackt durch den articleStream geschickt. Es ist sehr gut sichtbar, wie die Abhängigkeiten der Komponenten stets nur von den Widgets zu den States zu den BLoCs zeigen. Es gab nur geringfügige Abweichungen vom Entwurf. Das Entwurfsmuster konnte dank guter Planung und Recherche erfolgreich umgesetzt werden. Einzig der Rücksprung in die App aus einer Push-Benachrichtigung bleibt problematisch. Statt direkt zum beworbenen Artikel kommt der Nutzer zunächst einmal nur zur Übersicht der Applikation. Der direkte Aufruf des Artikels erfordert mehr technisches Know-how und Aufwand und wird auf eine spätere Iteration verschoben.

6 Zweite Iteration

In der zweiten Iteration sollen weitere Anforderungen erfüllt werden. Dazu wurde, wie im agilen Prozess üblich, mit dem Auftraggeber Rücksprache gehalten. Zur Umsetzung der neuen Anforderungen ist eine Überarbeitung des Konzepts unter Berücksichtigung der während der ersten Iteration gewonnenen Erkenntnisse nötig.

6.1 Planung

Nach der Implementation des ersten Prototypen wurde ein Termin mit dem wissenschaftlichen Mitarbeiter von FINK.HAMBURG gemacht. Dieser äußerte im Interview den Wunsch, die Artikel suchbar zu machen. Außerdem wäre eine Unterteilung der Übersicht in die verschiedenen Ressorts praktisch.

Weitere Anforderungen werden für die zweite Iteration erst einmal zurück gestellt. Dadurch werden nicht zu viele Änderungen gleichzeitig vorgenommen. So bleibt das Risiko klein, in eine unerwünschte Richtung zu entwickeln. Beispielsweise die Umsetzung von horizontalen Wischgesten zum Wechsel zwischen Artikeln macht eine größere Anpassung der Oberfläche erforderlich, die eine eigene Iteration verdient. Weitere übrig gebliebene Anforderungen sollen in späteren Iterationen angestrebt werden, die dann nicht mehr Teil dieser Arbeit sind.

6.2 Überarbeitung des Konzepts

In diesem Abschnitt sollen die neuen Anwendungsfälle und das erweiterte fachliche Datenmodell beschrieben werden.

6.2.1 Anwendungsfälle

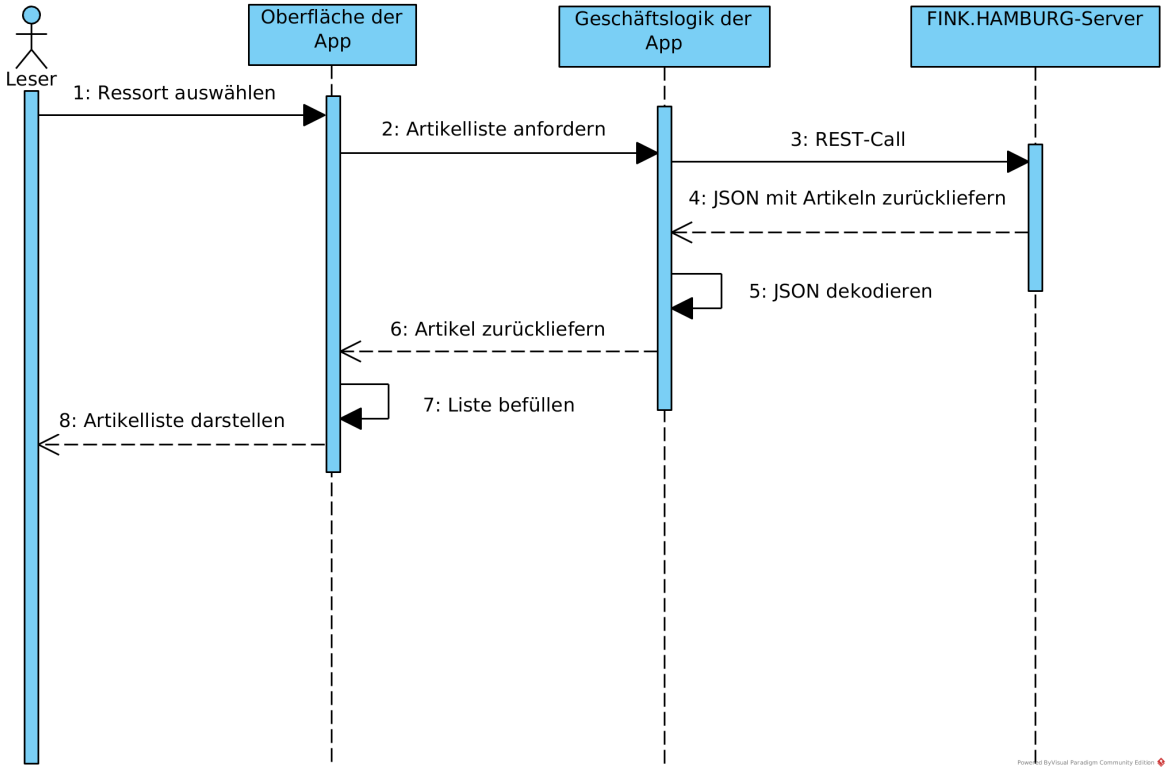

Abbildung 6.1: Sequenzdiagramm des dritten Anwendungsfalls

Der dritte Anwendungsfall ist die Auswahl eines spezifischen Ressorts zur Anzeige von Artikeln. Der Nutzer hat über ein Menü an der Seite *(Drawer)* die Möglichkeit, ein spezifisches Ressort zu wählen. Daraufhin wird der Abruf der Artikel auf das gewünschte Ressort beschränkt und die Artikel werden wie in der allgemeinen Übersicht in der Liste angezeigt.

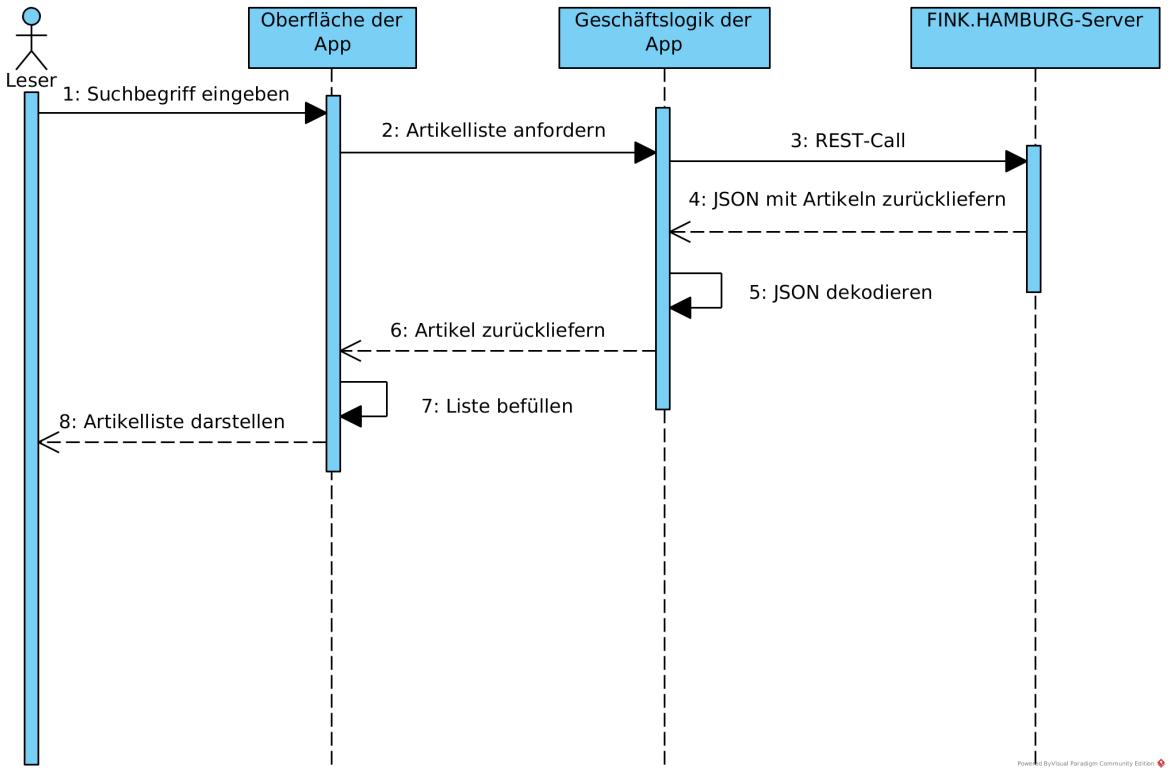

Abbildung 6.2: Sequenzdiagramm des vierten Anwendungsfalls

Der vierte Anwendungsfall ist die Suche nach Artikeln. Nach einem Tipp auf ein Lupensymbol in der oberen Leiste der App wird ein Textfeld sichtbar, welches der Nutzer zur Textsuche nutzen kann. Daraufhin wird in der Übersichtsliste eine Liste relevanter Artikel angezeigt.

6.2.2 Fachliches Datenmodell

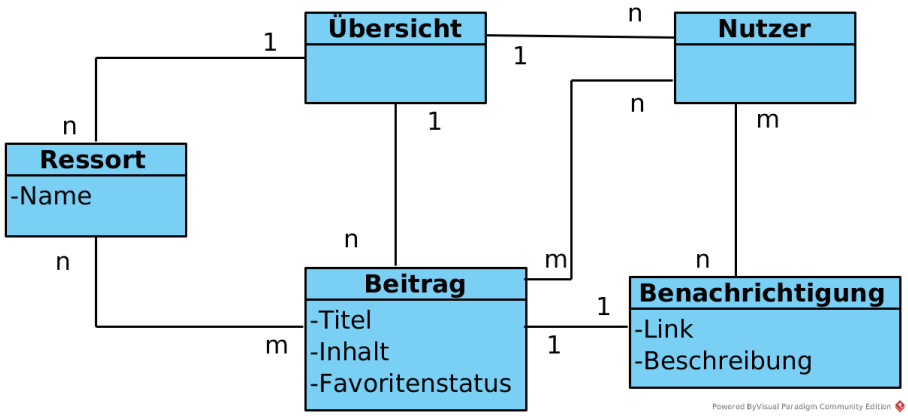

Abbildung 6.3: Fachliches Klassendiagramm des Systems, zweite Iteration

Die Fachlichkeit des Systems wird leicht erweitert. Ein Beitrag kann nun als Favorit markiert werden. Außerdem ist er einem oder mehreren Ressorts zugeordnet. In der Übersicht werden die Ressorts von FINK.HAMBURG angezeigt, aus denen der Nutzer auswählen kann.

6.3 Umsetzung

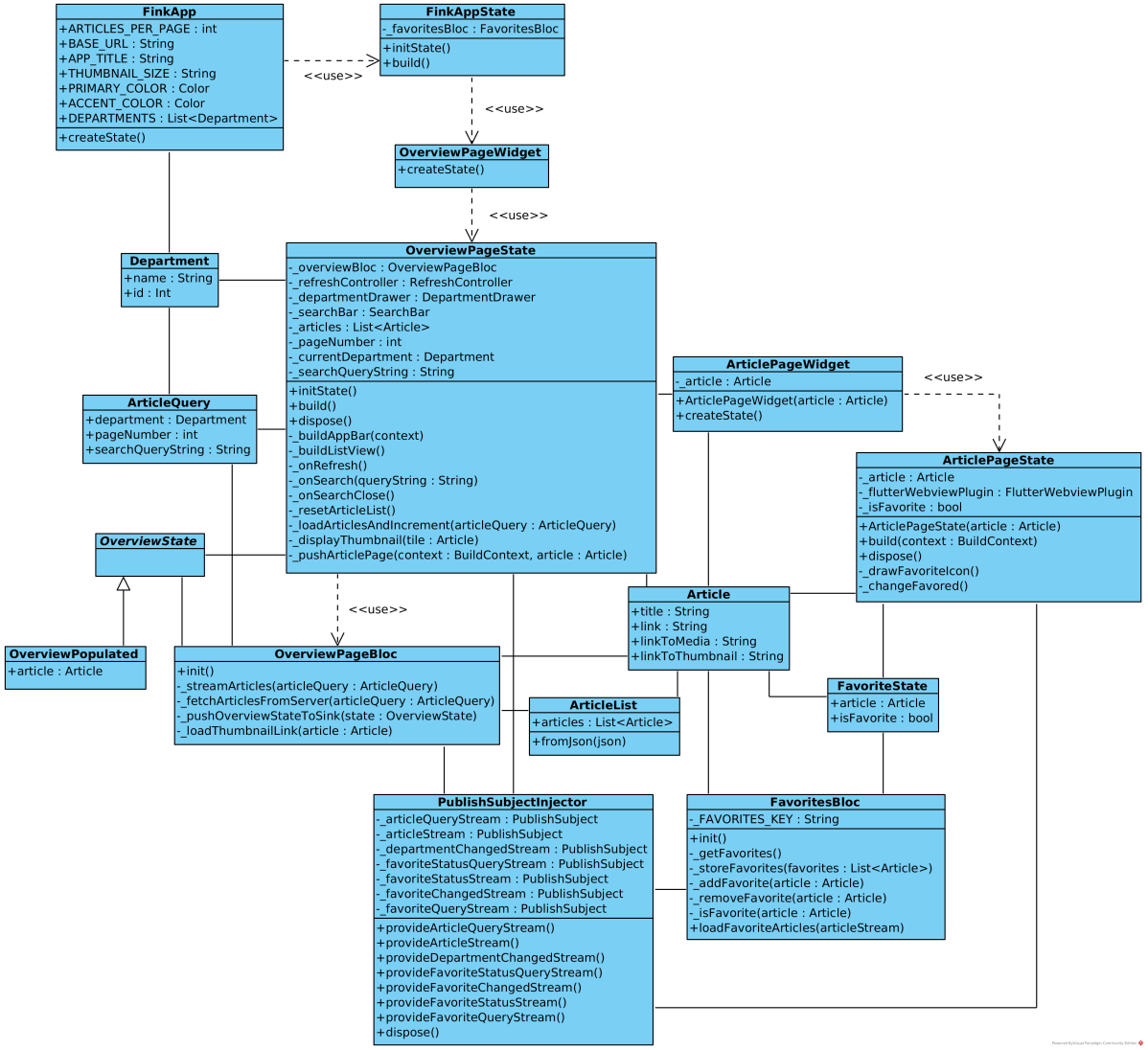

Abbildung 6.4: Technisches Klassendiagramm der zweiten Iteration

Die Architektur wurde gegenüber der ersten Iteration erheblich weiterentwickelt. Die Einführung der Favoritenfunktion macht eine weitere BLoC erforderlich, die FavoritesBLoC. Diese fungiert als Zugang zur Persistenz des Systems und kümmert sich um das Ablegen und Abrufen von favorisierten Artikeln. Außerdem ist mit der Einführung der Suchfunktion der Abruf von Artikeln nun komplexer. Daher gibt es die Klasse ArticleQuery neu, die einen Artikelabruf modelliert. Dieser enthält das abzurufende Ressort, die aktuelle Seitennummer und einen

eventuellen Filtertext, der in der Suche eingegeben wurde. Für die Kommunikation zwischen den Komponenten sind nun sieben Streams verantwortlich, die im PublishSubjectInjector zu sehen sind. Von dort können sie nach Bedarf in den entsprechenden Klassen abgerufen werden. Diese Vorgehensweise ist als *Dependency Injection* bekannt. Das macht ein eventuell später nötiges Refactoring einfacher, da die Streams nicht in den BLoCs initialisiert werden, sondern von außen kommen.

In Abbildung 6.5 sind drei exemplarische Screenshots der App dargestellt. Von links nach rechts finden sich hier: Die Übersichtsseite der App (OverviewPage), der Drawer sowie eine Artikelseite (ArticlePage).

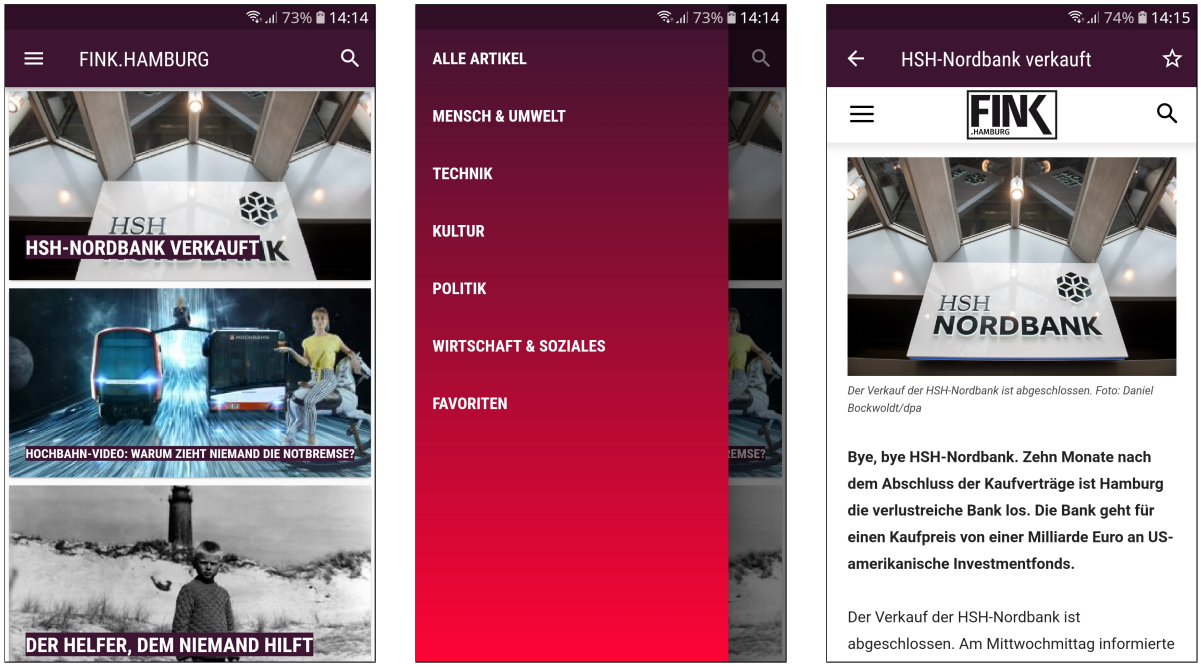

Abbildung 6.5: Screenshots der FINK.HAMBURG-App

7 Evaluation

Die Evaluation besteht aus zwei großen Teilen. Zu Anfang soll der konkrete, experimentelle Teil des Projekts evaluiert werden. Dazu wird Bezug auf die Anforderungen aus Abschnitt 3.3.3 auf Seite 11 genommen. Dazu wurde mit der Redaktion von FINK.HAMBURG über das Ergebnis der zweiten Iteration reflektiert.

Der zweite Part ist der abstrakte, analytische Teil. Hier wird ein Urteil gefällt, ob Flutter für die Entwicklung professioneller mobiler Apps mit moderaten Anforderungen geeignet ist.

7.1 Feedback zum entwickelten System

Nach der zweiten Iteration wurde ein Termin mit der Redaktion von FINK.HAMBURG gemacht. Dort findet regelmäßig als Teil des Arbeitsablaufs eine Retrospektive statt. Bei diesem Termin reflektieren die Studierenden über den erreichten Erfolg der Artikel und Vertriebskanäle. So wird zum Beispiel analysiert, wie groß die Leserschaft war, die über Facebook-respektive Twitter-Posts gekommen ist. An diese Retrospektive wurde das Feedback über die FINK.HAMBURG-App angehängt. Die App wurde frontal am Beamer vorgestellt. Mit der erfolgreichen Umsetzung der funktionalen Anforderungen 1,2,3,5 und 7 (vgl. Tabelle 3.3.3) ist eine App im Sinne der Auftraggeber entstanden. Nach der Vorstellung der App konnten die Studierenden Fragen und Wünsche äußern. Diese drehten sich vor allem um die Gestaltung der App und nicht um die Technik dahinter. So wurde unter anderem gewünscht, die Schriftart der Website zu verwenden. Ein weiterer Punkt war die Gestaltung der Artikelübersicht. Hier gab es verschiedene Vorschläge zur Darstellung der einzelnen Listeneinträge. Es wurde deutlich, dass selbst in einer verhältnismäßig kleinen Runde die Vorstellung von einer gelungenen Bedienoberfläche weit auseinander geht. Über Gedankenexperimente und Vergleiche wurden die Studierenden angeregt, über die eingebrachten Vorschläge zu diskutieren. Letztendlich ist eine Liste an Änderungs- und Weiterentwicklungswünschen entstanden, aus der sich weitere Iterationen ableiten lassen.

7.2 Bewertung von Flutter

Im folgenden Abschnitt wird Flutter abstrakt vom tatsächlich realisierten System bewertet. Es folgen Einschätzungen zu den verschiedenen technischen und fachlichen Entscheidungen, die dem Framework zu Grunde liegen.

7.2.1 Engine und Sprache

Flutters Wahl, Dart als Sprache zu verwenden, erweist sich als gut geeignet für das Anwendungsszenario. Das Ausführen der App im Debug-Modus in der dann auf dem Gerät installierten Dart-VM ist schnell genug, um einen repräsentativen Eindruck von der App zu bekommen. Die ahead of time kompilierte Release-Version der App ist unter Android merklich schneller und macht deutlich, dass Dart eine gute Wahl für diesen spezifischen Anwendungsfall ist. Der Ansatz, den Code je nach Szenario ahead oder just in time zu kompilieren, geht in der Praxis auf. Für Entwickler mit Vorerfahrung in Java ist die Sprache tatsächlich sehr leicht zu benutzen. Die Flutter Engine läuft schnell und liefert ein performantes Ergebnis, dass sich mit nativen Apps vergleichen lässt.

Im Folgenden wird auf die Speicherbelegung von Flutter-Apps eingegangen. Alle Messwerte stammen von der während der zweiten Iteration entwickelten App, die auf einem Samsung Galaxy S7 unter Android 8 ausgeführt wurde. Zur Analyse der Laufzeitperformanz wurde *Flutter Performance* verwendet, ein Menüpunkt des Flutter-Plugins für Android Studio. Die App läuft mit flüssigen 60 Bildern pro Sekunde. Beim Start belegt sie ungefähr 16MB an Arbeitsspeicher. Diese Zahl steigt mit dem Scrollen durch die Übersichtsliste, für jeden geladenen Artikel um etwa 3MB. Wird ein Artikel aufgerufen und die Webview gestartet, belegt die App durchschnittlich 30MB.

Interessant ist vor allem auch die Belegung des Systemspeichers. Diese findet sich in Tabelle 7.1. Hier sind die zwei mit Flutter erzeugten Varianten der FINK.HAMBURG-App abgetragen. Außerdem finden sich zum Vergleich die nativen Apps von Zeit Online in der Version 1.9.2 und Spiegel Online in der Version 3.2.6.

Die just in time kompilierte Debug-Version der Flutter-App ist durch die mitgelieferte Dart-VM deutlich größer. Bei der Release-Variante trägt zwar auch die Flutter-Engine zur Größe der App bei, allerdings ist diese ahead of time kompilierte Edition deutlich kleiner als die Debug-Version. Der zusätzliche Speicheraufwand durch Flutters Engine ist also vernachlässigbar, die Release-Fassung ist sogar kleiner als die native App von Spiegel Online.

Flutter (Debug)	Flutter (Release)	Zeit Online	Spiegel Online
49,92MB	21,07MB	13,19MB	32,27MB

Tabelle 7.1: Speichervergleich verschiedener News-Apps

7.2.2 Designentscheidungen

Bezogen auf mögliche Architekturen für Flutter-Apps ist der Entwickler sehr frei. Das hat Vor- und Nachteile. Einerseits ist es gut, dass Flutter wenige Einschränkungen bei der Organisation des Codes macht. So kann flexibel auf technische oder fachliche Anforderungen reagiert werden, die an die Architektur gestellt werden. Andererseits gibt es für Einsteiger eine Vielzahl an Wegen, wie eine Architektur organisiert sein kann, was zu Verunsicherung führen kann.

Zur Entwicklung grafischer Oberflächen setzt Flutter auf den Leitsatz „Everything is a widget“ („Alles ist ein Widget“). Das stellt sich als zweischneidiges Schwert dar. Einerseits ist der Weg, einen Bereich der App um eine Funktion zu erweitern, fast immer klar. Durch die Idee des *composition over inheritance (Komposition statt Vererbung)* wird jede noch so beeindruckende Kombination von Features in kleine, für den Entwickler gut handhabbare Einzelteile zerlegt. Andererseits führt gerade dieser Weg dazu, dass bei komplexen Layouts und Funktionen ein sogenannter *widget tree* entsteht, der schnell unübersichtlich wird. Viele Klammern können beim Bewegen oder Verändern von Codezeilen Verwirrung beim Entwickler stiften. Soll eine geschachtelte Ebene entfernt oder hinzugefügt werden, muss auf sehr viele Klammern geachtet werden.

Ein Beispiel für einen widget tree mit vielen Schachtelungen sieht in Flutter so aus:

```
GestureDetector(
 onTap: () => _pushArticlePage(context, tileArticle),
  child: Card(
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
     SizedBox(
      height: 184.0,
      child: Stack(
       children: <Widget>[
        Positioned.fill(
         child: _displayThumbnail(tileArticle)),
        Positioned(
         bottom: 16.0,
         left: 16.0,
         right: 16.0,
         child: FittedBox(
          fit: BoxFit.scaleDown,
          alignment: Alignment.centerLeft,
          child: Container(
           decoration: BoxDecoration(
           color: FinkApp.ACCENT_COLOR),
           child: Text(tileArticle.title,
            style: TextStyle(
             fontWeight: FontWeight.bold,
             color: Colors.white))),
        ),
        ),
       ],
      ),
     )
    ])));
```

7.2.3 Integration in den Arbeitsfluss

Das Problem unübersichtlicher *widget trees* wird durch Flutters exzellentes Tooling abgemildert. Flutter bietet sowohl für Android Studio wie auch Visual Studio Code Plugins an, die das Entwickeln angenehmer machen. Die Integration in Android Studio funktioniert solide und schnell. Erfahrene Android-Entwickler brauchen keine lange Einarbeitungszeit, viele Features wie der Hot Reload sind sinnvoll und übersichtlich integriert. Der Hot Reload stellt einen echten Gewinn für die Entwicklungsarbeit dar, allerdings mit einer Einschränkung. Während der Entwicklung der Oberfläche ist er gerade im Vergleich zu nativer Android-Entwicklung deutlich praktischer und schneller. Unter Android kann ein Gradle-Build zum Anzeigen der Codeänderungen durchaus eine Minute dauern. Durch den Hot Reload lässt sich eine Änderung im Flutter-Code tatsächlich in unter zwei Sekunden nachvollziehen. Bei der Entwicklung von komplexer Geschäftslogik reicht der Hot Reload allerdings manchmal nicht aus, um die Änderungen vollumfänglich sichtbar zu machen. Flutters sogenannter *Hot Restart*, der die komplette App neu startet, ist aber immer noch schneller als ein Gradle-Build unter Android.

7.2.4 Community und Ökosystem

Flutters Dokumentation ist sehr gut. Der Ansatz, das Framework bereits früh während des Entwicklungsprozesses open source zu machen, scheint vielversprechend. Es gibt bereits eine schnell wachsende Community rund um Flutter. Das zeigt sich gut am Diagramm 7.1. Dort ist der prozentuale Anteil der Fragen, die sich um Flutter drehen, an allen auf https://stackoverflow.com gestellten Fragen abgetragen. Entscheidend ist hierbei nicht der konkrete Anteil von 0,45% sondern vor allem der Trend. Es lässt sich gut erkennen, dass Flutter nach Erscheinen der ersten Entwicklerversionen eine kleine Plateau-Phase in seiner Bekanntheit hatte. Seit dem Release der offiziellen Beta wächst das Interesse am Framework kontinuierlich.

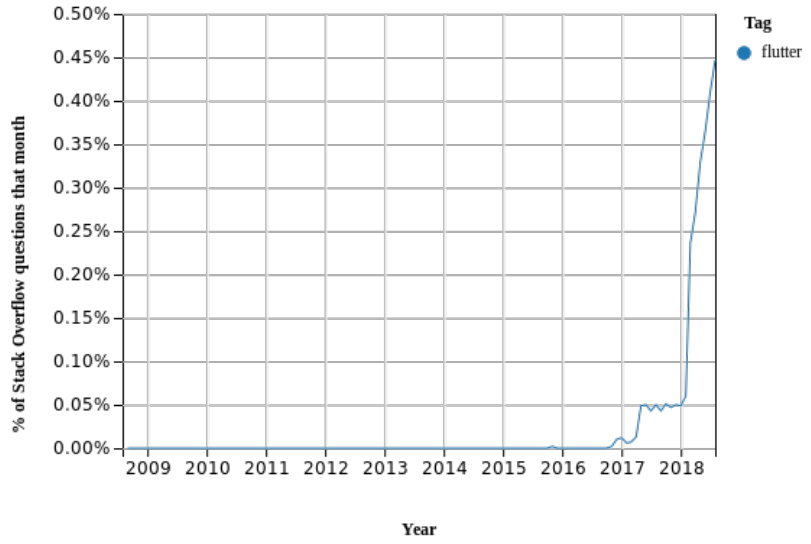

Abbildung 7.1: Entwicklung der Fragen auf Stackoverflow zum Thema Flutter

Mit der wachsenden Zahl an Entwicklern ist auch mit einer Verbesserung der Situation rund um Community-Bibliotheken zu rechnen. Für viele Probleme sind bereits Bibliotheken vorhanden, diese sind im Moment oftmals aber noch nicht als stabil gekennzeichnet und nicht in großer Zahl verfügbar. Das in der zweiten Iteration genutzte Webview-Package beispielsweise verlangsamt die ansonsten sehr schnelle App spürbar.

7.2.5 Nutzwertanalyse

Aus der Evaluation ergibt sich eine grobe Nutzwertanalyse für die Entwicklung mobiler Apps. Es werden drei Varianten bewertet und verglichen: Native App-Entwicklung, Flutter und React Native. Hierbei ist zu beachten, dass React Native stellvertretend für alle Web-basierten Cross-Plattform-Frameworks aufgeführt ist.

Diese Nutzwertanalyse basiert maßgeblich auf subjektiven Erfahrungen und den aus Abschnitt 2.2 (Seite 4) gewonnenen Erkenntnissen. Auch hier ist also ein gewisser wissenschaftlicher Abstand angeraten.

Zur Analyse werden zwei Szenarien entworfen, deren Gewichtungen in den beiden rechten Spalten dargestellt sind. Im ersten Szenario geht es um die Entwicklung einer hochqualitativen, langfristig betriebenen Applikation. Dieses Szenario findet sich zum Beispiel bei der Umsetzung einer App mit Endkunden, welche die App häufig benutzen sollen. Das zweite Szenario beschreibt ein rein auf die Kosten ausgerichtetes Projekt. Dieser Fall liegt in der Regel vor, wenn eine firmeninterne App mit einem sehr spezifischen Zweck entwickelt wird.

	nativ	**React Native**	**Flutter**	**Szenario 1**	**Szenario 2**
Aufwand	1	4	6	x2.0	x3.0
Performanz	9	6	9	x2.0	x1.0
Look & Feel	10	7	9	x2.0	x1.0
Community	10	8	6	x0.5	x0.5
Summe Szenario 1	45	38	51		
Summe Szenario 2	27	29	39		

Tabelle 7.2: Nutzwertanalyse aktueller App-Entwicklungsmethoden

Native Entwicklung von mobilen Apps ist mit Abstand am aufwändigsten, liefert aber auch die performantesten und oftmals schönsten Ergebnisse. Die Community ist über mehr als zehn Jahre gewachsen und es ist leicht, Hilfe zu bekommen.

React Native ist mit deutlich weniger Aufwand verbunden, das geht allerdings stellenweise auf Kosten der Performanz und des Designs. Die Community ist seit nun schon mehr als drei Jahre alt und es gibt für viele bekannte Probleme erprobte Lösungen.

Flutter schafft es, bei geringem Aufwand eine große Flexibilität bei der Gestaltung von Apps zu ermöglichen. Die Performanz ist teilweise sogar besser als die von nativen Apps. Einzig die Community ist bei Flutter noch nicht so stark ausgebildet wie bei den anderen Kandidaten. Dieser Faktor ist aber aufgrund von Flutters hoher Produktivität vernachlässigbar, was sich auch in der Gewichtung zeigt. In beiden Szenarien liegt Flutter vorne. Die gesteigerte Produktivität und damit der geringere Aufwand bei der Entwicklung kombiniert mit nahezu nativer Performanz macht hier den entscheidenden Unterschied.

8 Fazit und Ausblick

Die App für FINK.HAMBURG ist schnell, erfüllt die Wünsche der Redaktion und wurde von nur einem Entwickler über einen Zeitraum von wenigen Monaten entwickelt. Das Vorgehensmodell hat sich bewährt und die Zusammenarbeit mit den Studierenden des Departments Medien war fruchtbar.

In weiteren Iterationen ist die Erweiterung der App für FINK.HAMBURG denkbar. Die Unterstützung der Video-Inhalte macht eine Integration der YouTube-API erforderlich. Eine Gestensteuerung zur Navigation zwischen den einzelnen Artikeln wäre praktisch, würde aber wohl größere Änderungen der Widgets erforderlich machen. Die Webview zum Anzeigen der Artikel ist im Moment der einzige verlangsamende Faktor in der App. Hier bleibt abzuwarten, ob sich die Situation in Zukunft verbessert. Durch eine Weiterentwicklung von Drittanbietern oder offiziellen Support durch das Flutter-Team wäre eine schnelle Unterstützung von HTML in Flutter-Apps möglich. Die Unterstützung von Push-Benachrichtigungen erfordert weitere Aufmerksamkeit. Mittels sogenannter *Deeplinks* wird dann hoffentlich auch ein Sprung von der Benachrichtigung direkt in den angezeigten Artikel und nicht nur in die Übersicht möglich. Flutter bietet außerdem einfache Unterstützung für selbst entwickelte Animationen. Durch die Integration von mehr Animationen in angemessenem Rahmen wäre eine weitere Verbesserung der Benutzererfahrung möglich.

Trotz der anfangs etwas steilen Lernkurve ist das Arbeiten mit Flutter nach einer ersten Eingewöhnung sehr angenehm. Die größte Herausforderung stellte wohl die Erarbeitung der Architektur dar. Das lag aber vor allem daran, dass Flutter eine große Flexibilität beim Aufbau von Architekturen bietet. Es bleibt abzuwarten, ob sich in der noch jungen Community ein favorisierter Architekturstil durchsetzt. Insgesamt lässt sich feststellen, dass Flutter viele Probleme der vorangegangenen Technologien schon bei der Konzeption angeht und erfolgreich bewältigt.

Mit dem Release von Flutter 1.0 wurde *Hummingbird* angekündigt [26]. Hummingbird befindet sich noch in einem frühen Stadium der Entwicklung. Es soll das Ausführen von mit Flutter erstellten Anwendungen im Browser möglich machen. Ermöglicht wird dies

durch Darts Fähigkeit, zu JavaScript transpiliert zu werden. Auch das Einbetten in einem Desktop-Kontext ist inzwischen möglich. Flutter-Apps können bereits jetzt experimentell auf Windows, MacOS und dem Raspberry Pi ausgeführt werden. Abgesehen von der Cross-Plattform-Entwicklung für alle genannten Systeme gibt es Hinweise, dass Flutter auch zur Entwicklung von Anwendungen für Fuchsia benutzt werden kann [27]. Fuchsia ist ein bei Google in der Entwicklung befindliches Betriebssystem für Geräte des *Internet Of Things* und Smartphones. So könnte Google mit Flutter den Übergang von Android zu Fuchsia anstreben. Es ist zu erwarten, dass Flutter mit seinem universellen Ansatz einen Platz in der Anwendungsentwicklung finden wird.

Literatur

[1] Google Inc. *flutter.io*. 5. Dez. 2018. URL: https://flutter.io/.

[2] Eric Newton. "The "teaching hospital"—-a goal for journalism education". In: *Knight Foundation* 22 (2013).

[3] Microsoft Corporation. *Xamarin Documentation*. 5. Dez. 2018. URL: https://docs.microsoft.com/de-de/xamarin/.

[4] o.V. *The MIT License*. 5. Dez. 2018. URL: https://opensource.org/licenses/MIT.

[5] Drifty Co. *Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular*. 5. Dez. 2018. URL: https://ionicframework.com/.

[6] Google Inc. *Angular*. 5. Dez. 2018. URL: https://angular.io/.

[7] The Apache Software Foundation. *Apache Cordova*. 5. Dez. 2018. URL: https://cordova.apache.org/.

[8] Altexsoft. *Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison*. 5. Dez. 2018. URL: https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-comparison/.

[9] Facebook Inc. *React Native - A framework for building native Apps using React*. 5. Dez. 2018. URL: https://facebook.github.io/react-native/.

[10] Facebook Inc. *React - A JavaScript library for building user interfaces*. 5. Dez. 2018. URL: https://reactjs.org/.

[11] o.V. *The 3-Clause BSD License*. 5. Dez. 2018. URL: https://opensource.org/licenses/BSD-3-Clause.

[12] Google inc. *Concepts | Android NDK | Android Developers*. 5. Dez. 2018. URL: https://developer.android.com/ndk/guides/concepts.

[13] llvm-admin team. *The LLVM Compiler Infrastructure Project*. 5. Dez. 2018. URL: https://llvm.org/.

[14] Google inc. *Skia Graphics Library*. 5. Dez. 2018. URL: https://skia.org/.

[15] Just Software AG. *Just Social | Dein Digital Workspace*. 5. Dez. 2018. URL: https://www.just.social/de/.

[16] Square Inc. *Retrofit*. 5. Dez. 2018. URL: https://square.github.io/retrofit/.

[17] Thomas Epping. *Kanban für die Softwareentwicklung*. Springer Berlin Heidelberg, 2011. Kap. 4.

[18] Ralf Wirdemann und Johannes Mainusch. *Scrum mit User Stories*. 3. Aufl. Carl Hanser Verlag München, 2017, S. 36–45.

[19] Jochen Ludewig und Horst Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. 3. Aufl. dpunkt.verlag Heidelberg, 2013, S. 163–168.

[20] Atlassian Corporation Plc. *Trello*. 5. Dez. 2018. URL: https://trello.com/.

[21] Google Developers. *Build reactive mobile apps with Flutter (Google I/O '18)*. 5. Dez. 2018. URL: https://youtu.be/RS36gBEp8OI.

[22] Jonas Bonér u. a. *The Reactive Manifesto*. 5. Dez. 2018. URL: https://www.reactivemanifesto.org/.

[23] o.V. *ReactiveX*. 5. Dez. 2018. URL: http://reactivex.io/.

[24] Google Inc. *Firebase*. 5. Dez. 2018. URL: https://firebase.google.com/.

[25] Brian Egan. *Keep it Simple, State: Architecture for Flutter Apps (DartConf 2018)*. 5. Dez. 2018. URL: https://youtu.be/zKXz3pUkw9A.

[26] Google Developers. *One More Thing (Flutter Live)*. 5. Dez. 2018. URL: https://youtu.be/5SZZfpkVhwk.

[27] Google Inc. *Fuchsia - Flutter Module Development*. 5. Dez. 2018. URL: https://fuchsia.googlesource.com/docs/+/HEAD/development/languages/dart/mods.md.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 7. Dezember 2018 Leonard Thiele