DreamSwipe
Tinder für Filme

Leon Gieringer, Robin Meckler,Vincent Schreck

Studienarbeit

4. März 2021

Inhaltsverzeichnis

1 Einleitung **1**

2 Motivation **1**

3 Theoretische Grundlagen **1**

3.1 Anwendungsentwicklung für mobile Endgeräte 1

3.1.1 Plattformspezifische native Anwendungen 1

3.1.2 Adaptive Webanwendungen 2

3.1.3 Plattformübergreifende Anwendungen 2

3.1.4 Flutter 2

3.1.5 React Native 3

3.2 Language 3

3.3 IDE 3

3.4 Database 3

3.5 Firebase 3

3.6 Recommender System 3

3.6.1 Nutzerinformation 4

3.6.2 Content-based filtering 5

3.6.3 Collaborative Filtering 6

3.6.4 Ähnlichkeit von Objekten und Nutzern 6

4 Konzept? **7**

5 Funktionen/Komponenten **7**

5.1 Swipe/Aussuchen/Voting 7

5.2 Matches/Chat 7

5.3 Film-/Serienvorschläge 7

5.4 Gruppenorgien 7

5.5 Gespeicherte Filme/Filmliste 7

5.6 Zugänglichkeit/Behindertenfreundlichkeit 7

6 Benutzeroberflächen **7**

6.1 Home-Screen 7

6.2 Gruppen 7

6.3 Chat 7

6.4 Filmliste 7

7 CodeBeispiele **7**

8 Probleme **7**

9 Fazit **7**

1 Einleitung

2 Motivation

3 Theoretische Grundlagen

3.1 Anwendungsentwicklung für mobile Endgeräte

Mobile Geräte sind heutzutage ein sehr großer Teil unseres Tagesablaufs. Durchschnittlich verbringen wir 3:54 Stunden pro Tag an mobilen Geräten (hier bezogen aus Bürger der USA). Die meiste Zeit hiervon wird in Apps (ca. 90%). [1] Laut Cisco wird dieser Markt sich jedoch nicht nur auf Industrieländer beruhen, sondern bis 2023 sollen weltweit 71% der Bevölkerung mobile Konnektivität haben. [4] Diese Entwicklung forcierte viele Firmen immer mehr ihre Anwendungen auch *mobile ready* zu gestalten. Dies kann man bspw. deutlich bei der Anpassung vieler Webseiten an Mobile Seiten- und Größenverhältnisse oder auch dem Anbieten von *Apps*, welche bereits für Desktop o.ä. verfügbar waren, erkennen.

Daher ist es für die Wirtschaft und Entwicklung gleichermaßen wichtig sich ständig weiterzuentwickeln und sich nicht auf (Kosten-) ineffiziente Entwicklungsprozesse auszuruhen. Dabei bieten jährliche, wenn nicht sogar halbjährliche Design- und Performanceänderungen von den Geräten selbst oder der Betriebssysteme Herausforderungen an die mobilen Anwendungen und gleichzeitig an deren Programmierumgebung. Trotz einer riesigen Auswahl an *Apps* lassen sich diese allgemein in drei Kategorien eingliedern: Plattformspezifische Native Anwendungen, Adaptive Webanwendungen und Plattformübergreifende Native Anwendungen.

3.1.1 Plattformspezifische native Anwendungen

Plattformspezifische oder auch native Anwendungen sind Programme, welche auf eine gewisse Plattform abzielen und in einer der davon unterstützen Programmiersprachen geschrieben wurden. Da diese Art der (mobilen) Anwendung mit plattformspezifischen *Software Development Kits (SDK)* und *Frameworks* entwickelt wird, ist diese Anwendung an eine Plattform gebunden. Dies bringt zum einen natürlich Vorteile wie allgemein best mögliche Performance auf der jeweiligen Plattform und direkt vom Hersteller unterstützte Entwicklungsumgebungen/SDKs. Zudem lassen sich plattformspezifische Fähigkeiten oder Einstellungen nutzen - beispielsweise mehrere Kameras oder *Global Positioning System (GPS)*.

Gleichzeitig beschränkt man sich aber logischerweise auf eine Plattform und deckt mit einer Anwendung nur einen Teil des gesamten Marktes. Dies bringt im Vergleich zu den anderen Möglichkeiten einen deutlich erhöhten Entwicklungs- und Wartungsaufwand mit sich, da für andere Plattformen Programmcode nicht übernommen werden kann. Zusätzlich benötigen Entwickler spezifische Kompetenzen für beide Plattform und Entwicklungsumgebungen.

Zwei der am weitesten verbreiteten Plattformen sind Android von Google und iOS von Apple. Anwendungen für Android können in Kotlin oder Java als Programmiersprache beispielsweise in dem *integrated development environment (IDE)* von Google Android Studio entwickelt werden. Für iOS wird hingegen mit Objective-C und Swift als Programmiersprache primär in der IDE XCode entwickelt.

Beide bieten jeweils Plattform eigene Services an, beispielsweise das direkte Veröffentlichen in den jeweiligen Appstore [2]

[1] `https://www.emarketer.com/content/us-time-spent-with-mobile-2019`, zuletzt aufgerufen: 26.02.2021

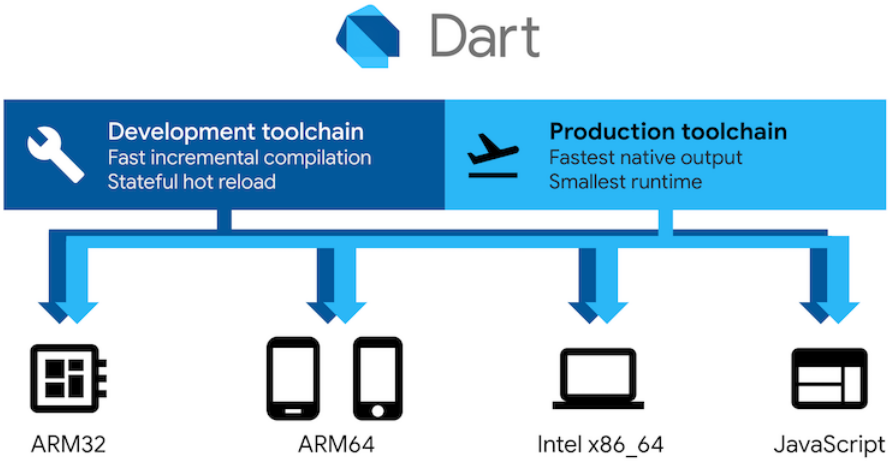

Abbildung 1: Kompatibilität der Dart Plattform [3]

3.1.2 Adaptive Webanwendungen

3.1.3 Plattformübergreifende Anwendungen

3.1.4 Flutter

Flutter ist eine open-source SDK entwickelt von Google und ist geschrieben in C, C++ und Dart. Flutter erlaubt es Anwendungen für Android, iOS, Web und Desktop basierend auf einem Code zu erstellen und ist zudem die primäre Methode für Google Fuchsia, Googles Betriebssystem.[2] Flutter verwendet Skia als 2D Grafikbibliothek, welche auch von Chrome, Firefox und Android verwendet wird. Zudem basiert Flutter auf der Dart Plattform welche das Compilieren auf 32-bit und 64-bit ARM Prozessoren, auf Intel x64 Prozessoren und in JavaScript ermöglicht (siehe Abbildung 1).

Während der Entwicklung werden Flutter Apps in einer Virtuellen Maschine (VM) gestartet, welche *stateful hot reload* ermöglicht - bei Änderungen muss die App also nicht komplett neu kompiliert werden. Wird die App nun veröffentlicht, wird sie in die Maschinencode der beschriebenen Plattformen übersetzt.

Architektur

> Flutter is designed as an extensible, layered system. It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable.[4]

Grundlegend ist das Framework in drei Prozesseinheiten gegliedert. Diese bestehen wiederum jeweils aus, für sie charakteristischen APIs und Bibliotheken:

- *Flutter embedder*: Der Einstiegspunkt in die jeweilige Plattform. Er koordiniert Zugriffe auf Services des Betriebssystems; er ist also zuständig für bspw. die Kommunikation mit dem

[2] Quelle: `https://fuchsia.dev/`
[3] Quelle: `https://github.com/flutter/flutter`
[4] `https://flutter.dev/docs/resources/architectural-overview`

Input Method Editor (IME) und den Lifecycle Events der App. Daher ist der Embedder in der, von der Plattform unterstützten Programmiersprache geschrieben: derzeit wird Java und C++ für Android, Objective-C/Objective-C++ für iOS und macOS, und C++ für Windows und Linux verwendet.

- *Flutter Engine*: Der Kern von Flutter, geschrieben hauptsächlich in C und C++, ist die *low-level* Implementierung der Flutter Kern Programmierschnittstelle (API). Daher ist sie zuständig für das graphische Darstellen (Rasterisierung) des Codes sobald ein neuer *Frame* angezeigt werden muss. Im Flutter Framework wird die *Engine* als dart:ui Bibliothek offengelegt - der zugrundeliegende C++ Code wird in Dart Klassen eingefügt.
- *Flutter Framework*: Das Framework, mit welchem der Entwickler schlussendlich meistens arbeiten wird. Es ist in Dart geschrieben und bietet sogenannte *Layer* für Animationen, Layout und Widgets. Widgets werden von Flutter als Einheit der Komposition von Benutzeroberflächen verwendet und sind als einzelne Bausteine zu verstehen, welche zusammengefügt ein Objekt oder sogar einen kompletten Bildschirm ergeben.

Bei der Entwicklung mit Flutter wird ein Baum von Widgets erzeugt, welcher als Bauplan der Applikation angesehen werden kann. Nach diesem Plan wird mithilfe von States der einzelnen Widgets schlussendlich das User Interface (UI) gerendert.[3]

Widgets

States

3.1.5 React Native

3.2 Language

Hier steht mein Language Text.

3.3 IDE

Hier steht mein IDE Text.

3.4 Database

Hier steht mein Database Text.

3.5 Firebase

Hier steht mein Firebase Text.

3.6 Recommender System

Auf der Webseite Youtube allein werden minütlich mehr als 500 Stunden Videomaterial hochgeladen. (https://blog.youtube/press/, 10.02.2021) Um bei einer solch unvorstellbaren Menge an Daten (allein auf einer Webseite) den Überblick als Endnutzer behalten zu können, ist ein personalisierten Filtersystems unausweichlich.

Solche Filtersysteme, auch Recommendation System genannt, nutzt bisher gesammelte Daten um Nutzern potentiell interessante Objekte jeweils individuell vorzuschlagen. Ein sogenannter

[5]Quelle: `https://github.com/flutter/flutter`

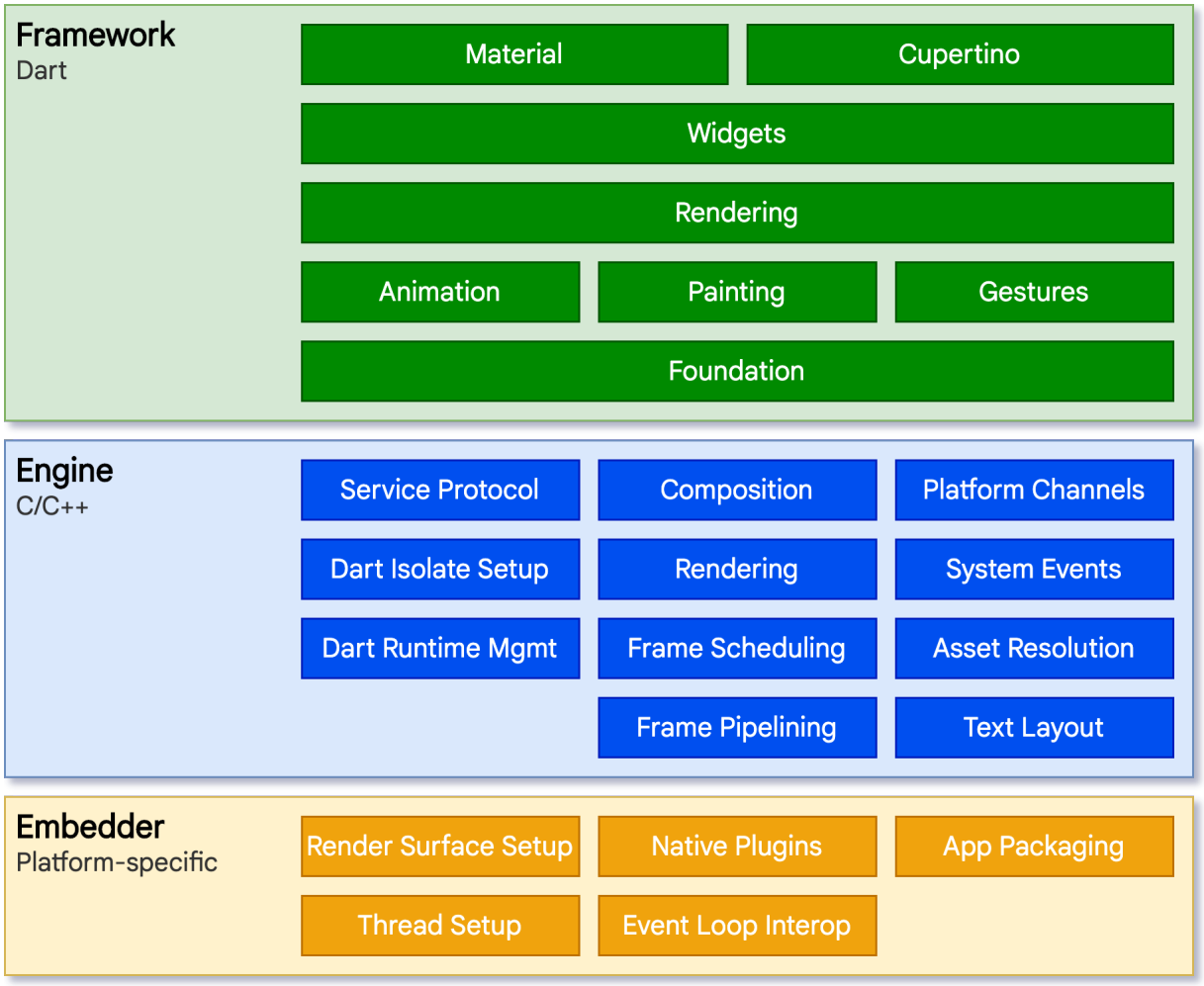

Abbildung 2: Bibliotheken und Ebenen der Flutter Plattform [5]

Candidate Generator ist hierbei ein Recommendation System, welches die Menge M als Eingabe erhält und für jeden Nutzer eine Menge N ausgibt. Hierbei umfasst M alle Objekte und gleichzeitig gilt $N \subset M$.

Die Bestimmung einer solchen Menge N beruht grundlegend auf zwei Informationsarten. Erstens die sogenannten Nutzer-Objekt Interaktionen, also beispielsweise Bewertungen oder auch Verhaltensmuster; Und zweitens die Attributwerte von jeweils Nutzer oder Item, also beispielsweise Vorlieben von Nutzern oder Eigenschaften von Items.[1] Systeme, welche zum Bewerten ersteres benutzen, werden *collaborative filtering* Modelle genannt. Andere, welche zweiteres verwenden, werden *content-based filtering* Modelle genannt. Wichtig hierbei ist jedoch, dass *content-based filtering* Modelle ebenfalls Nutzer-Objekt Interaktionen (v.a. Bewertungen) verwenden können, jedoch bezieht sich dieses Modell nur auf einzelne Nutzer - *collaborative filtering* basiert auf Verhaltensmustern von allen Nutzern bzw. allen Objekten.

Ein solches Recommendation System kann im einfachsten Fall wie in 3.6 als Matrix dargestellt werden.

3.6.1 Nutzerinformation

Damit ein *Recommender System* einem Nutzer Vorschläge bereitstellen kann, benötigt es Nutzerinformationen. Das Design des jeweiligen Systems hängt auch, wie oben beschrieben, von der

		Items					
		1	2	...	i	...	m
Users	1	2		1			3
	2	4			5		
	...			1			4
	u		4		5		1
		2				3	
	n		4		3		

Tabelle 1: Nutzer-Item Matrix mit Bewertungen. Jede Zelle $r_{u;i}$ steht hierbei für die Bewertung des Nutzers u an der Stelle i

Art der Information und von der Art der Beschaffung dieser ab.

Explizite Nutzerinformation Bei der expliziten Methode muss der Nutzer individuelle Informationen aktiv über sich preisgeben. Dies kann über konkrete Fragestellungen zu beispielsweise Geburtsdatum, Geschlecht oder Interessen geschehen. Diese Art der Information beschreiben einen Nutzer konkret.

Eine andere Art der Information sind Bewertungen von Objekten. Diese lassen sich beispielsweise Intervall basiert darstellen. Hierbei werden geordnete Zahlen in einem Intervall als Indikator genutzt, ob ein Objekt gut oder schlecht war - zum Beispiel eine Bewertung eines Produktes von 0 bis 5 Sternen bei Amazon. Diese Information beschreiben die Vorlieben eines Nutzers konkret.

Je größer diese Skala ist, desto differenzierter ist auch das Meinungsbild, da jeder Nutzer sich genau ausdrücken kann. Jedoch desto komplizierter und unübersichtlich wird auch das Bewertungsverfahren an sich, da man einen zu großen Entscheidungsraum für den Nutzer darbietet.

Implizite Nutzerinformation Um implizit Nutzerinformationen zu erfassen, muss ein System die Verhaltensmuster seiner Kunden als Daten abspeichern. Beispielsweise könnte das System von YouTube erfassen, ob Videos frühzeitig abgebrochen oder ganz angeschaut werden. Anklicken von Webseiten und die darauf verbrachte Zeit könnte ebenfalls als Bewertung gespeichert und zur Generierung von Vorschlägen genutzt werden.

3.6.2 Content-based filtering

Unter *content-based filtering* versteht man das Betrachten von Ähnlichkeiten zwischen Objekten anhand von Schlüsselwörtern (Eigenschaften) und daraus dann das Vorhersagen der Nutzer-Objekt Kombination für ein bestimmtes Objekt. Nimmt man an, Film 1 und Film 2 haben ähnliche Eigenschaften (gleiches Genre, gleiche Schauspieler, ...) und Nutzer A mag Film 1, so wird das System Film 2 vorschlagen.

Das System ist also unabhängig von anderen Nutzerdaten, da die Vorschläge nur auf Präferenzen eines einzelnen Nutzers basieren. Dies bietet im Hinblick auf eine App auch gute Skalierungsmöglichkeiten. Zudem kann auf Nischen-Präferenzen gut eingegangen werden, da nicht mit anderen Nutzerdaten verglichen wird, sondern nur ein Nutzer für sich betrachtet wird.

Gleichzeitig schlagen *content-based filtering* Systeme aber eher offensichtliche Objekte vor, da Nutzer oft unzureichend genaue "Beschreibungen", also Vorlieben mit sich bringen. Dadurch, dass nur basierend auf Schlüsselwörter neue Objekte vorgeschlagen und andere Nutzerwertungen nicht miteinbezogen werden, sind die Vorschläge sehr wahrscheinlich oftmals ähnlich bis gleich - man "verfängtßich quasi in eine Richtung.[1]

3.6.3 Collaborative Filtering

Unter *collaborative filtering* versteht man das Betrachten von Ähnlichkeiten im Verhalten von Nutzern anhand von Bewertungen und Präferenzen, bzw. anhand der Ähnlichkeiten von Objekten.

Generell unterscheidet man in zwei Typen:[1]

1. *Memory-based Methoden*: Es wird, wie oben beschrieben, aus gesammelten Daten Ähnlichkeit herausgearbeitet und Nutzer-Objekt Kombinationen durch eben diese vorhergesagt. Daher wird dieser Typ auch *neighborhood-based collaborative filtering* genannt. Man unterscheidet weiter in:

   (a) *User-based*: Ausgehend von einem Nutzer A werden andere Nutzer mit ähnlichen Nutzer-Objekt Kombinationen gesucht, um Vorhersagen für Bewertungen von A zu treffen. Ähnlichkeitsbeziehungen werden also über die Reihen der Bewertungsmatrix berechnet.

   (b) *Item-based*: Hierbei werden ähnliche Objekte gesucht und diese genutzt um die Bewertung eines Nutzers für ein Objekt vorherzusagen. Es werden somit Spalten für die Berechnung der Ähnlichkeitsbeziehungen verwendet.

2. *Model-based Methoden*: Machine Learning und Data Mining Methoden werden verwendet um Vorhersagen über Nutzer-Objekt Kombinationen zu treffen. Hierbei sind auch gute Vorhersagen bei niedriger Bewertungsdichte in der Matrix möglich.

Vereinfacht gesagt: Wenn Nutzer A ähnliche Bewertungen verteilt wie Nutzer B, und B den Film 1 positiv bewertet hat, wird das System Film 1 auch Nutzer A vorschlagen. Das selbe gilt auch umgekehrt (*Item-based*).

Diese Art leidet sehr unter dem *sparsity* Problem, also dass die Nutzer zu wenige Bewertungen von Objekten ausüben. Daher sind Vorhersagen über Ähnlichkeit von Nutzern aufgrund unzureichender Datensätze nicht sinnvoll möglich. Dieses Problem wird *Cold-Start Problem* genannt.

3.6.4 Ähnlichkeit von Objekten und Nutzern

Sowohl bei *collaborative filtering*, als auch bei *content-based filtering* wird jedes Objekt und jeder Nutzer als ein Vektor im Vektorraum-Modell $E = \mathbb{R}^d$ (englisch *embedding space*) erfasst. Sind Objekte beispielsweise ähnlich, haben sie eine geringe Distanz voneinander.

Ähnlichkeitsfunktionen sind Funktionen $s : E \times E \to \mathbb{R}$ welche aus zwei Vektoren beispielsweise von einem Objekt $q \in E$ und einem Nutzer $x \in E$ ein Skalar berechnen, welches die Ähnlichkeit dieser zwei beschreibt $s(q,x)$.

Hierfür werden mindestens eine der folgenden Funktionen verwendet:

- Cosinus-Funktion
- Skalarprodukt
- Euklidischer Abstand

Cosinus-Funktion Hier wird einfach der Winkel zwischen beiden Vektoren berechnet: $s(q,x) = \cos(q,x)$

Skalarprodukt Je größer das Skalarprodukt, desto ähnlicher sind sich die Vektoren. $s(q,x) = q \circ x = \sum_{i=1}^{d} q_i x_i$

Euklidischer Abstand $s(q,x) = ||q - x|| = [\sum_{i=1}^{d}(q_i - x_i)^2]^{\frac{1}{2}}$
https://dl.acm.org/doi/pdf/10.1145/3383313.3412488 http://www.microlinkcolleges.net/elib/files/undergraduate/Photography/504703.pdf

4 Konzept?

5 Funktionen/Komponenten

5.1 Swipe/Aussuchen/Voting

5.2 Matches/Chat

5.3 Film-/Serienvorschläge

5.4 Gruppenorgien

5.5 Gespeicherte Filme/Filmliste

5.6 Zugänglichkeit/Behindertenfreundlichkeit

6 Benutzeroberflächen

6.1 Home-Screen

6.2 Gruppen

6.3 Chat

6.4 Filmliste

7 CodeBeispiele

8 Probleme

9 Fazit

Literaturverzeichnis

[1] Aggarwal, C. C. (2016). Recommender systems (Vol. 1). Cham: Springer International Publishing.

[2] Fentaw A. E. (2020). Cross platform mobile application development: a comparison study of React Native Vs Flutter.

[3] Flutter Flutter architectural overview [Online] Verfügbar: https://flutter.dev/docs/resources/architectural-overview, Aufgerufen am: 04.03.2021

[4] Cisco (2020) Cisco Annual Internet Report (2018–2023) White Paper [Online] Verfügbar: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-interr white-paper-c11-741490.html