

Duale Hochschule Baden-Württemberg Karlsruhe  
Fachbereich technische Informatik

# **Entwicklung und Erstellung einer Dating-App auf Basis von Film- und Serienpräferenzen für mobile Endgeräte**

Studienarbeit von

**Leon Gieringer, Robin Meckler & Vincent Schreck**

betreut durch

**Prof. Dr. Kai Becher**

abgegeben am

16. Mai 2021

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Methode . . . . .	1
<b>2 Theoretische Grundlagen</b>	<b>2</b>
2.1 Netzwerkprotokolle . . . . .	2
2.1.1 Schichtenmodell . . . . .	2
2.1.2 HTTP . . . . .	3
2.1.3 HTTPS . . . . .	5
2.2 JavaScript . . . . .	6
2.2.1 Historie . . . . .	6
2.2.2 Wesentliche Programmereigenschaften . . . . .	6
2.2.3 Anwendungsbereiche . . . . .	7
2.3 Node.js . . . . .	8
2.3.1 Architektur . . . . .	8
2.3.2 Module . . . . .	10
2.4 Representational State Transfer - Application Programming Interface . . . . .	16
2.5 NoSQL-Datenbank . . . . .	17
2.5.1 NoSQL-Datenbanktypen . . . . .	17
2.5.2 BASE . . . . .	19
2.5.3 CAP-Theorem . . . . .	19
2.5.4 MongoDB . . . . .	20
2.6 Firebase . . . . .	27
2.6.1 Firebase Authentifizierung . . . . .	27
2.6.2 Cloud Firestore . . . . .	27
2.6.3 Cloud Storage . . . . .	30
2.6.4 Cloud Functions . . . . .	30
2.6.5 Cloud Messaging . . . . .	31
2.6.6 Google AdMob . . . . .	32
2.7 Anwendungsentwicklung für mobile Endgeräte . . . . .	33
2.7.1 Begriffe . . . . .	33
2.7.2 Plattformspezifische native Apps . . . . .	33
2.7.3 Plattformübergreifende Anwendungen . . . . .	34
2.8 Frameworks zur mobilen, plattformübergreifenden Entwicklung . . . . .	36
2.8.1 React Native . . . . .	37
2.8.2 Flutter . . . . .	42
2.9 Recommender System . . . . .	46
2.9.1 Nutzerinformation . . . . .	46
2.9.2 Content-based filtering . . . . .	47
2.9.3 Collaborative Filtering . . . . .	47
2.9.4 Ähnlichkeit von Objekten und Nutzern . . . . .	48
<b>3 Planung</b>	<b>49</b>
3.1 Konzept . . . . .	49
3.2 Komponenten . . . . .	49

<b>4 Auswahl geeigneter Technologie</b>	<b>52</b>
4.1 Server . . . . .	52
4.2 Datenbank . . . . .	52
4.3 Kommunikationsschnittstelle . . . . .	53
4.4 Film-Datenbank . . . . .	53
<b>5 Backend-Implementierung</b>	<b>54</b>
5.1 Server . . . . .	54
5.1.1 Einrichtung . . . . .	54
5.1.2 Sicherheit . . . . .	54
5.1.3 Webserver . . . . .	54
5.2 Datenbank . . . . .	54
5.2.1 Einrichtung . . . . .	54
5.3 Kommunikationsschnittstelle . . . . .	54
5.3.1 Implementierung . . . . .	54
5.4 Swipe/Aussuchen/Voting . . . . .	54
5.5 Matches/Chat . . . . .	54
5.6 Film-/Serienvorschläge . . . . .	54
5.7 Gespeicherte Filme/Filmliste . . . . .	54
5.8 Barrierefreiheit . . . . .	54
5.8.1 Barrierefreiheit in mobilen Anwendungen . . . . .	55
5.8.2 Barrierefreiheit in Filmen und Serien . . . . .	55
5.8.3 Barrierefreiheit bei StreamSwipe . . . . .	55
<b>6 Benutzeroberflächen</b>	<b>58</b>
6.1 Aspekte von Benutzeroberflächen . . . . .	58
6.2 Oberflächen von StreamSwipe . . . . .	59
6.2.1 Login-Screen . . . . .	59
6.2.2 Home-Screen . . . . .	61
6.2.3 Swipe-Screen . . . . .	62
6.2.4 Chat . . . . .	64
6.2.5 Benutzerprofil . . . . .	65
<b>7 Probleme</b>	<b>68</b>
<b>8 Anwendbarkeit</b>	<b>70</b>
<b>9 Ausblick</b>	<b>71</b>
<b>10 Fazit</b>	<b>72</b>
<b>A Verfasser einzelner Abschnitte</b>	<b>73</b>

## Abbildungsverzeichnis

1	OSILAYER [1.1] . . . . .	2
2	HTTP-Nachrichtenaufbau . . . . .	4
3	JavaScript Objekt [1.29] . . . . .	7
4	Multithreaded / Blocking I/O [Nodejs 1.1] . . . . .	8
5	Single Threaded / Non Blocking I/O [Nodejs 1.1] . . . . .	9
6	Middleware [nodejs 2.0] . . . . .	12
7	Graphikdatenbank Beispiel [NoSql 1.2] . . . . .	17
8	Spaltenorientierte Datenbank Beispiel [NoSql 1.5] . . . . .	18
9	Zeilen- und spaltenorientierte Speicherung [NoSql 1.5] . . . . .	19
10	Visualization-of-CAP-theorem [NoSql 1.67] . . . . .	19
11	Graphikdatenbank Beispiel [NoSql 1.9] . . . . .	20
12	MongoDB Replica Set [3.8] . . . . .	25
13	CRUD-Bespielfunktionen eines Mongoose-Models . . . . .	26
14	Datenmodell in Firebase <sup>1</sup> . . . . .	28
15	Cloud Functions Anwendungsfall Benachrichtigung . . . . .	31
16	Firebase Cloud Messaging Architektur . . . . .	32
17	Google AdMob Anzeigemöglichkeiten . . . . .	32
18	Kategorisierung verschiedener plattformübergreifender Ansätze . . . . .	34
19	Struktur einer hybriden Anwendung . . . . .	35
20	Beliebtheit nach Framework im Bereich der plattformübergreifenden mobilen Entwicklung . . . . .	37
21	Alte Architektur von React Native . . . . .	38
22	Neue Architektur von React Native . . . . .	39
23	Kompatibilität der Dart Plattform . . . . .	42
24	Bibliotheken und Ebenen der Flutter Plattform <sup>2</sup> . . . . .	44
25	Widget Baum einer beispielhaften Anwendung <sup>3</sup> . . . . .	45
26	Deklarative Benutzeroberfläche . . . . .	45
27	Komponentendiagramm . . . . .	51
28	Screenshots als Beispiele für Barrierefreiheit . . . . .	56
29	Screenshots der Anmeldeseiten . . . . .	60
30	Screenshots des Home-Screens . . . . .	61
31	Screenshots des Swipe-Screen . . . . .	63
32	Screenshots der Chat-Seiten . . . . .	65
33	Screenshots der Profilseite . . . . .	67

## Quellcodeverzeichnis

1	Benannter Export von Modulen . . . . .	10
2	Import von Modulen . . . . .	10
3	Einfacher Webserver [nodejs 1.8 . . . . .	11
4	Express.json Middleware benutzen . . . . .	12
7	Mongoose Schema - Beispiel . . . . .	14
8	Model erstellen und exportieren . . . . .	15
9	Model importieren - Objekt instanziieren und persistent speichern . . . . .	15
10	CRUD-Beispieldfunktionen eines Mongoose-Models . . . . .	15
11	Mongoose: Verbindung zur Datenbank aufbauen . . . . .	16
12	Mongoose Verbindungssoptionen -nodejs 3.3- . . . . .	16
13	MongoDB Read . . . . .	21
14	MongoDB Read Modifikation . . . . .	21
15	MongoDB Aggregate . . . . .	22
16	MongoDB Create . . . . .	23
17	MongoDB Update . . . . .	23
18	MongoDB Remove . . . . .	24
19	Beschränkung des Zugriffs auf Dokumente der Sammlung <code>cities</code> . . . . .	28
20	Hierarchische Zugriffsbeschränkung . . . . .	29
21	Datenvalidierung für atomare Operationen . . . . .	29
22	Validierung nach Dateigröße . . . . .	30
23	JSX Hello World Element . . . . .	39
24	Native Komponenten . . . . .	39
25	Eigene Komponenten . . . . .	40
26	State mit <code>useState</code> Hook . . . . .	41
27	Codeausschnitt in Dart von einem Button mit Semantiken. . . . .	57

## 1 Einleitung

Die Zahl der Scheidungen in Deutschland hat sich während den Corona-Einschränkungen 2020 verfünfacht [14]. Neben dem erhöhten Ansturm auf Rechtskanzleien haben sich auch unverheiratete Paare zu Zeiten des Lockdowns getrennt und die Paartherapien sind flächendeckend ausgebucht. Die Menschen suchen sich Partner aus, die sie zwar attraktiv finden, mit denen sie jedoch kaum gemeinsame Interessen und Ansichten teilen. Sind diese Leute gezwungen Zeit miteinander zu verbringen realisieren sie, dass ihre Beziehung nicht passt.

Wir wollen diesen gravierenden Fehler in seinem Keim erstickten und revolutionieren das Dating-Game mit einem Verfahren, bei dem persönliche Vorlieben im Vordergrund stehen und das Aussehen zweitrangig ist.

### 1.1 Motivation

Bereits vor tausenden von Jahren haben sich die Menschen Partner gesucht und mit der ersten Monogamie kam auch die erste Beziehung und wahrscheinlich auch die ersten Beziehungsprobleme.

Eine der wichtigsten Grundlagen einer Beziehung sind gleiche Ansichten, Interessen und Vorlieben, anstatt oberflächliche Merkmale wie Aussehen und Geld, denn Schönheit vergeht und Charakter besteht. Jedoch ist es nahezu unmöglich einen Partner nach diesen Kriterien auszusuchen, denn man weiß erst wie gut man zueinander passt, nachdem man sich kennengelernt hat. Viele möglicherweise sehr glückliche Beziehungen finden gar nicht statt, da die Person durch ein eigentlich weniger wichtiges Kriterium herausgefiltert wurde. Sucht man die Ursache dieses Problems, ist man schnell bei der Art des Kennenlernens. Der erste Eindruck ist gewöhnlicherweise optischer Natur. Dementsprechend ist Aussehen in der Realität das erste Filterverfahren, was durch StreamSwipe an eine spätere Position tritt.

Wir bieten die Lösung zu einem jahrtausendealten Problem der Menschheit.

### 1.2 Methode

Gerade in den letzten Jahren genießt das Medium Film und Serie einen immer höheren Stellenwert in der Gesellschaft. Durch Video-on-Demand Plattformen wie Netflix, Disney+ und Amazon Prime Video sind Filme und Serien omnipräsent geworden und der Nachschub scheint endlos zu sein. Der Zugriff auf komplette Serien wurde dadurch stark vereinfacht und der Nutzer kann einer Serie oder Filmreihe treu bleiben, da er keine Folge mehr verpassen kann. So kann dieses Medium bereits bei vielen Menschen eine Charaktereigenschaft werden und Charaktereigenschaften vieler Zuschauer passen sich an Filmcharaktere an.

Bereits 2017 haben die 18- bis 39-Jährigen an durchschnittlich 4 Tagen pro Woche eine Serie angeschaut [15]. Aus diesen Vorlieben lässt sich sehr viel auslesen. Bei StreamSwipe wird dies ausgenutzt und über eine Film- und Serienauswahl des Nutzers ein Geschmack berechnet, der über einen Algorithmus mit anderen ähnlichen Geschmäcken gematcht wird. Sobald ein Match entstanden ist, öffnet sich ein privater Chat und die beiden Personen können sich austauschen und verabreden.

## 2 Theoretische Grundlagen

### 2.1 Netzwerkprotokolle

#### 2.1.1 Schichtenmodell

Eine der gängigsten Arten der Kommunikation findet heutzutage über das Internet statt. Dabei handelt es sich um ein weltweit verbundenes Netz von Rechnern. Zur Gewährleistung einer effizienten und geregelten Datenübertragung der heterogenen Computer im Internet wurden Regelwerke, die sogenannten Netzwerkprotokolle, benötigt. Um das Jahr 1980 wurden daraufhin von verschiedenen Computerherstellern modularisierte Protokolle entwickelt, die fortan als Standard für die digitale Übertragung innerhalb von Rechnernetzen gelten sollen.[1.0] Es musste eine Vielzahl von Aufgaben bewältigt und Anforderung bezüglich Zuverlässigkeit, Sicherheit, Effizienz etc. erfüllt werden. Die Aufgaben reichten dabei von der elektronischen Übertragung der Signale bis zur geregelten Reihenfolge der in der Kommunikation abstrakteren Aufgaben.[1.05] Aus den zu lösenden Problemen und Anforderung kristallisierten sich sieben Schichten bzw. Ebenen heraus. Jede einzelne Schicht setzt dabei separat eine Anforderung um und kann dabei durch verschiedene Protokolle realisiert werden. In dem sich etablierten OSI-Schichtenmodell bauen die einzelnen Schichten aufeinander auf, wobei die unterste Schicht das Fundament ist. Die Open Systems Interconnection (OSI) wurde von der International Organization for Standardization (ISO), der Internationalen Organisation für Normung, als Grundlage für die Bildung von offenen Kommunikationsstandards entworfen.

Zusätzlich zum OSI-Modell existiert das in den 1960er-Jahren entwickelte TCP/IP-Referenzmodell. Entwickler dieses Schichtmodells war das Verteidigungsministerium der Vereinigten Staaten, auch bekannt als das Departments of Defense (DoD). Dementsprechend trägt das TCP/IP-Referenzmodell auch den Namen DoD-Schichtenmodell. [1.06]

OSI-Schicht	TCP/IP-Schicht	Beispiel
Anwendungen (7)		HTTP, UDS, FTP, SMTP, POP, Telnet, DHCP, OPC UA
Darstellung (6)	Anwendungen	
Sitzung (5)		TLS, SOCKS
Transport (4)	Transport	TCP, UDP, SCTP
Vermittlung (3)	Internet	IP (IPv4, IPv6), ICMP (über IP)
Sicherung (2)		
Bitübertragung (1)	Netzzugang	Ethernet, Token Bus, Token Ring, FDDI

Abbildung 1: OSILAYER [1.1]

Tabelle 1: Kurzbeschreibung der OSI-Schichten [1.2]

OSI-Schicht	Aufgabe
Anwendungen	Funktionen für Anwendungen, sowie die Dateneingabe und -ausgabe.
Darstellung	Umwandlung der systemabhängigen Daten in ein unabhängiges Format.
Sitzung	Steuerung der Verbindungen und des Datenaustauschs.
Transport	Zuordnung der Datenpakete zu einer Anwendung.
Vermittlung	Routing der Datenpakete zum nächsten Knoten.
Sicherung	Fehlererkennungsmechanismen / Segmentierung der Pakete in Frames und Hinzufügen von Prüfsummen.
Bitübertragung	Umwandlung der Bits in ein zum Medium passendes Signal und physikalische Übertragung.

**IP** In der Vermittlungsschicht des OSI-Schichtenmodells findet, unabhängig des Übertragungsmediums und der genutzten Topologie, die logische Adressierung der Endgeräte statt. Das geläufigste Protokoll dafür ist das Internet Protocol (IP). Jedem am Netz verbundenen Teilnehmer wird eine IP-Adresse zugewiesen. Die bekannteste Notation ist die 32 Bit lange IPv4-Adressen und die IPv6-Adressen mit einer Größe von 128 Bit.

**TCP/UDP** In der Transportschicht wird eine Ende-zu-Ende-Kommunikation ermöglicht. Sie ist das Bindeglied zwischen den anwendungsorientierten und den transportorientierten Schichten. Die geläufigsten Protokolle sind das verbindungslose, unzuverlässige, aber weniger Overhead belastete User Datagram Protocol (UDP) und das verbindungsorientierte und datentransfer-zuverlässige Transmission Control Protocol (TCP). Jedes netzwerkfähige Gerät enthält eine Vielzahl von Ports, die primär zur Unterscheidung zwischen Datenströmen aus Anwendungen bei Netzwerkverbindungen genutzt werden. Anhand des genutzten Ports bei Netzwerkanfragen wissen Webserver, welches Protokollverfahren genutzt werden soll.

### 2.1.2 HTTP

Das Hypertext Transfer Protocol, kurz HTTP, ist ein zustandloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht.

**Kommunikation** Unter einer Nachricht versteht man in HTTP die Kommunikationseinheiten zwischen dem Zentralrechner (Server) und dem, der einen Dienst vom Server abruft (Client). Man unterscheidet dabei zwischen der Anfrage (Request) vom Client an den Server und der Antwort (Response) als Reaktion vom Server zum Client.

Eine Nachricht besteht aus dem Nachrichtenkopf (Message Header, kurz Header) und dem Nachrichtenrumpf (Message Body, kurz Body). Der Header enthält generelle Informationen über die Nachricht wie zum Beispiel den Methodentyp, das Datenformat, den genutzten Kompressions-

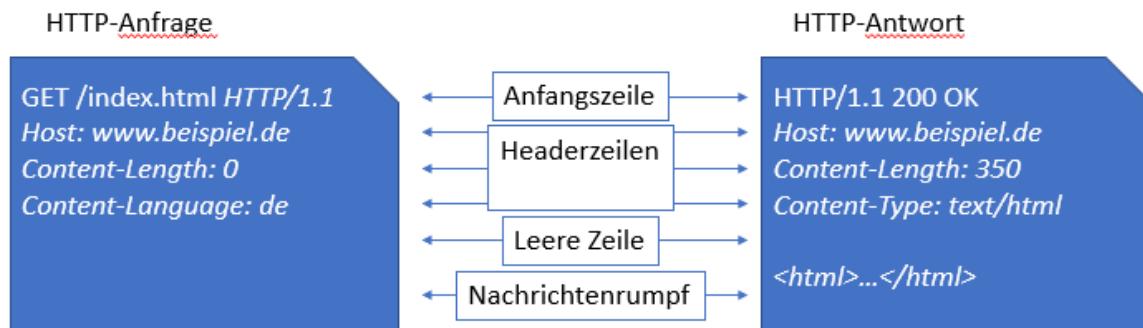


Abbildung 2: HTTP-Nachrichtenaufbau

algorithmus, die Länge der Nachricht oder die verwendete Kodierung im Body.

Die erste Zeile des Nachrichtenkopfs ist dreiteilig und besteht bei der Anfrage aus dem Namen der Anfragemethode, dem Pfad zur angeforderten Ressource (Uniform Resource Locator, kurz URL) und der verwendeten HTTP-Version. Die Anfangszeile einer HTTP-Antwort dagegen besteht zunächst aus der verwendeten HTTP-Version, gefolgt von dem zweiteiligen Status-Code. Der Anfangszeile beider Nachrichtentypen folgt eine Reihe von Headerzeilen, wobei jede Zeile aus einem Schlüsselwort/Wert-Paar besteht und die für die Datenübertragung wichtigen Informationen übermittelt. Der Nachrichtenrumpf, der mit den Nachrichtenkopf über einen Zeilenumbruch syntaktisch voneinander getrennt wird, enthält schließlich die Nutzdaten.

**Methoden** HTTP bietet fest definierte Standard-Methoden für Anfragen, die für verschiedene Aufgaben gedacht sind. Im Folgenden werden die wichtigsten Methoden beschrieben:

1. *GET* ist die gebräuchlichste Methode. Sie fordert vom Server eine Ressource, die bei Erfolg in der Antwort im Body zurückgegeben wird.
2. *POST* ist für die Änderung oder Erzeugung einer Ressource vorgesehen. Dafür werden bei der Anfrage zusätzlich Daten im Body der Nachricht übertragen.
3. *PUT* dient dazu, eine Ressource zu verändern, oder bei Nichtexistenz zu erstellen.
4. *PATCH* ändert eine bestehende Ressource ohne diese wie bei PUT vollständig zu ersetzen.
5. *DELETE* löscht die angegebene Ressource auf dem Server.
6. *OPTIONS* liefert eine Liste von Methoden und Merkmale, die vom Server unterstützt werden.

Tabelle 2: Module

Typ	Status-Code	Beispiele
Informational	1xx	100 Continue, 101 Switching
Success	2xx	200 OK, 201 Created, 202 Accepted
Redirection	3xx	300 Multiple Choice, 301 Moved Permanently
Client Error	4xx	400 Bad Request, 403 Forbidden
Server Error	5xx	500 Internal Server Error

**Statuscodes** HTTP-Antworten senden in der Anfangszeile ihrer Nachricht Statuscodes. Die Angabe ist zweiteilig und besteht aus einer standardisierten Statuskennzahl sowie einer kurzen textuellen Beschreibung, die zusammen Auskunft über den Bearbeitungszustand der zugehörigen Anfrage geben.

### 2.1.3 HTTPS

Das HTTP-Protokoll hat den großen Nachteil, dass die Nachrichten unverschlüsselt und ungesichert übertragen werden. Die Daten können bei der Übertragung von Dritten empfangen, gelesen und verändert werden. Hypertext Transfer Protocol Secure, kurz HTTPS, soll dem entgegenwirken und die Sicherheit bei der Kommunikation gewährleisten. Dafür dienen zwei Konzepte:

Ersteres ist das Verschlüsseln der Kommunikation von Sender und Empfänger. Die zugrundeliegende Technik nennt sich Transport Layer Security (TLS), ist aber auch als Secure Sockets Layer (SSL) bekannt. Die Idee dahinter ist, dass jeder Teilnehmer der Kommunikation einen öffentlich bekannten Schlüssel (Public Key) und einen geheimen, nicht-öffentlichen Schlüssel (Private Key) besitzen. Über den Public-Key des Empfängers verschlüsselt der Sender seine Nachricht. Diese kann nur über den Private-Key des Empfängers entschlüsselt werden, der vom Empfänger nicht weitergegeben werden sollte.

Das zweite Konzept von HTTPS ist die Webserver-Authentifizierung. Ein Zertifikat, dass zu Beginn der Kommunikation an den Webclient gesendet wird, bescheinigt die Vertrauenswürdigkeit des Senders. Dafür vertrauen Browser- und Betriebssystemhersteller bestimmten Zertifizierungsstellen, deren Zertifikate sie in ihrem Browser bzw. Betriebssystem hinterlegen. Die Kommunikation zwischen Webserver und Webclient findet folglich erst nach vollständiger Authentifizierung statt.

## 2.2 JavaScript

In den nächsten Unterkapiteln soll zunächst ein historischer Überblick über die Programmiersprache JavaScript gegeben werden. Im Anschluss wird auf die Bedeutung und Nutzung von JavaScript eingegangen.

### 2.2.1 Historie

Ihren Ursprung findet die Programmiersprache JavaScript im Jahr 1995, als Brendan Eich, ein damaliger Ingenieur des US-amerikanischen Software-Unternehmens „Netscape Communications Corporation“, innerhalb von zehn Tagen diese Sprache für den Browser „Netscape Navigator“ entwickelt hat. [1] Das Ziel dabei war es, eine Skriptsprache zu entwickeln, die es Entwicklern möglich machen sollte, auf ihren Webseiten Skripte umzusetzen. Zunächst noch unter dem Namen Mocha und LiveScript änderte sich der Name zu JavaScript aufgrund der Kooperation von Netscape und Sun, der Firma hinter der Programmiersprache Java, und Marketinggründen. Netscape wollte von der damaligen Popularität von Java profitieren. [1.05]

Netscapes Veröffentlichung des Netscape Navigator 2.0, der erste Browser der JavaScript unterstützte, brachte Microsoft dazu, Netscape als ernstzunehmenden Konkurrenten zu sehen. Microsoft antwortete im August 1995 mit der Veröffentlichung des ersten Internet Explorer zusammen mit der Skriptsprache JScript, die einen Dialekt der Sprache JavaScript darstellt. Dies ist ferner als der Beginn der „Browserkriege“ bekannt. [1.06]

Im Jahre 1997 reichte Netscape JavaScript an die European Computer Manufacturers Association (kurz ECMA[ABK]), einer privaten, internationalen Normungsorganisation zur Normung von Informations- und Kommunikationssystemen und Unterhaltungselektronik, ein. Das Ziel war es, von der ECMA einen einheitlichen Standard für die Sprache schaffen zu lassen, die fortan weiterentwickelt werden und von weiteren Browserherstellern genutzt werden soll. Das resultierende Standard nennt sich ECMAScript, wobei JavaScript die bisher bekannteste Implementierung dieses Standards ist. [1.07] Andere Implementierungen sind zum Beispiel ActionScript von Macromedia, JScript von Microsoft und ExtendScript von Adobe.

Jährlich wird dieser Standard seit Juni 2015 erweitert. ECMAScript Version 11 beziehungsweise ECMAScript 2020 bildet zum Zeitraum dieser Dokumentation [TODO??] den aktuellen Standard. [1.08] Im Juni 2021 soll die neueste Version ECMAScript 2021 veröffentlicht werden. [1.09]

### 2.2.2 Wesentliche Programmereigenschaften

„JavaScript is Not Java“ [1.091 ??]. Die Programmiersprache JavaScript wird aufgrund ihrer Namensgebung oft in falsche Zusammenhänge zu Java gebracht. Das häufigste Missverständnis sei, JavaScript wäre eine vereinfachte Version von Java. [1.091]

JavaScript ist eine interpretierte Programmiersprache mit objektorientierten Umsetzungsmöglichkeiten. Interpretation ist in diesem Zusammenhang so zu verstehen, dass der Quellcode zur Laufzeit eines Programms gelesen, übersetzt und ausgeführt wird. Syntaktisch ähnelt JavaScript komplizierten Programmiersprachen wie C, C++ und Java durch gleiche Umsetzung der Kontrollstrukturen wie den Bedingungen, Schleifen oder den booleschen Operatoren. [1.1] Wesentliche Unterschiede sind dagegen, dass JavaScript zum einen eine schwach-typisierte Sprache ist. Durch die schwache Typisierung haben Variablen keinen festen Datentyp und können diesen dynamisch zur Laufzeit ändern. Des Weiteren findet bei JavaScript die Objektorientierung prototypenbasiert statt. Diese

Form der Programmierung wird auch klassenlose Objektorientierung bezeichnet. Anders als bei der klassenbasierten Programmierung, bei der Objekte aus vordefinierten Klassen instanziert werden, werden hier Objekte durch Klonen bereits existierender Objekte erzeugt. Die Objekte, die geklont werden, sind dabei als Prototyp-Objekte zu verstehen. Beim Klonen werden alle Attribute und Methoden des Prototyp-Objekts in das neue Objekt übernommen und können dort überschrieben sowie erweitert werden. Objekte in JavaScript sind eher als Zuordnungslisten, ähnlich wie assoziative Arrays oder Hash-Tabellen, anzusehen, da bei der Eigenschaftszuweisung lediglich ein Mapping eines Schlüsselworts (Key) zu seiner zugehörigen Eigenschaft (Value) stattfindet.



Abbildung 3: JavaScript Objekt [1.29]

Ein weiterer Unterschied zu den anderen Programmiersprachen ist, dass alle Funktionen und Variablen außer der primären Datentypen Boolean, Zahl und Zeichenfolge, als Objekte verstanden werden können.

### 2.2.3 Anwendungsgebiete

Ursprünglich fand JavaScript seinen Einsatz hauptsächlich darin, dynamische Webseiten im Webbrowser anzuzeigen. Die Verarbeitung erfolgte dabei meist clientseitig durch den Webbrowser (dem sogenannten Frontend). [1.3]

Heutzutage findet sich die Sprache dagegen in wesentlich größeren Einsatzgebieten wieder. Bis vor einigen Jahren war die Serverseite anderen Programmiersprachen wie Java oder PHP vorbehalten. Die Veröffentlichung von Node.js, einer plattformübergreifenden Laufzeitumgebung, die JavaScript außerhalb eines Webbrowsers ausführen kann, führte zu einer immer größeren Verbreitung von serverseitigen Anwendungen (dem Backend), die auf JavaScript basieren. Auf Node.js wird ausführlicher im nächsten Kapitel eingegangen. Ferner findet JavaScript heutzutage aber auch seinen Einsatz in mobilen Anwendungen, Desktopanwendungen, Spielen oder 3D-Anwendungen. [1.4]]

## 2.3 Node.JS

Im Jahr 2009 veröffentlichte Ryan Dahl das Framework Node.js, das auf Googles V8-Engine, welche auch als JavaScript-Engine in Googles Browser Chrome zum Einsatz kommt, basiert und sich hervorragend für hochperformante, skalierbare und schnelle Webanwendungen eignet. Zudem ermöglicht es Webentwicklern die Entwicklung von serverseitigem JavaScript-Code. - [NodeJS 1.0]

### 2.3.1 Architektur

Eine wesentliche Eigenschaft von Node.js ist die hohe Performance. Im Folgenden soll der Unterschied der Node.js-Architektur zu traditionellen Webserversen und der damit verbundenen höheren Performance dargestellt werden.

Herkömmliche Webserver erstellten zunächst für jede ankommende Anfrage einen neuen Thread. Dieses Vorgehen ist eng mit steigendem Speicher- und Rechenaufwand verbunden. Um sich Rechenzeit, die durch die Erstellung und Zerstörung von Threads entstanden, zu sparen, wurden Threadpools eingerichtet. Dieser Threadpool enthält mehrere Threads, denen Aufgaben zugewiesen werden können. Nach erfolgreicher Abarbeitung einer Operation kann einem Thread eine weitere Aufgabe zugeordnet werden.

Es bleibt aber ein weiteres Problem: Bei der Anfragenabarbeitung kann es zu einer Form von blockierender Ein- und Ausgabe (Blocking Input/Output kurz Blocking I/O) kommen: zum Beispiel beim Suchen in einer Datenbank oder dem Laden einer Datei im Dateisystem. Während der Abarbeitung wartet der Thread solange, bis die Operation ein Ergebnis zurückwirft und belegt dabei weiterhin Speicherplatz. Bei hohem Aufkommen von Anfragen kommt es dadurch zu einer hohen Speicherauslastung des Servers. Zudem kosten die Kontextwechsel zwischen den Threads im Betriebssystem weitere Rechenzeit. [Node.js 1.05] Man spricht bei diesem Architekturkonzept auch vom Multi-Threaded Server.

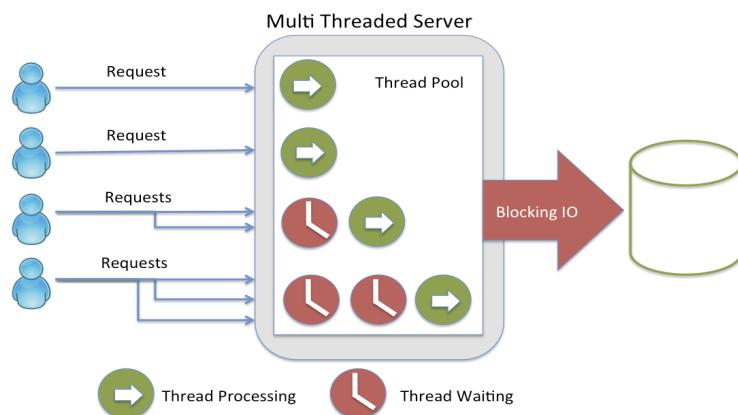


Abbildung 4: Multithreaded / Blocking I/O [Nodejs 1.1]

Node.js verfolgt einen anderen Ansatz: Anfragen werden nur in einem einzigen Thread, dem Hauptthread, abgearbeitet und in einer Warteschlange verwaltet. Dadurch bleiben Kontextwechsel zwischen Threads erspart. Hierbei handelt es sich also um einen Single-Threaded Server. Der Hauptthread verwaltet eine Schleife, die sogenannte Event Loop, die permanent Anfragen aus der Event-Warteschlange überprüft und Ereignisse, die von Ein- und Ausgangsoperationen ausgerufen werden, verarbeitet.

Bei Ankommen einer Nutzeranfrage an einen Node.js Server wird zunächst in der Event Loop geprüft, ob diese Anfrage Blocking I/O benötigt. Falls nicht, kann die Anfrage direkt bearbeitet werden und die Antwort an den Nutzer zurückgesendet werden.

Im anderen Fall wird einer von Node.js interner Workern, welche prinzipiell auch Threads sind, aufgerufen, um die jeweilige Operation auszuführen. Dabei wird eine Callback-Funktion mitgegeben, die vom Worker aufgerufen wird, sobald die Operation ausgeführt wurde. Diese Callback-Funktion kann anschließend als Ereignis von der Event Loop registriert werden. Man spricht hierbei auch von ereignisgesteuerter Architektur. [1.4]

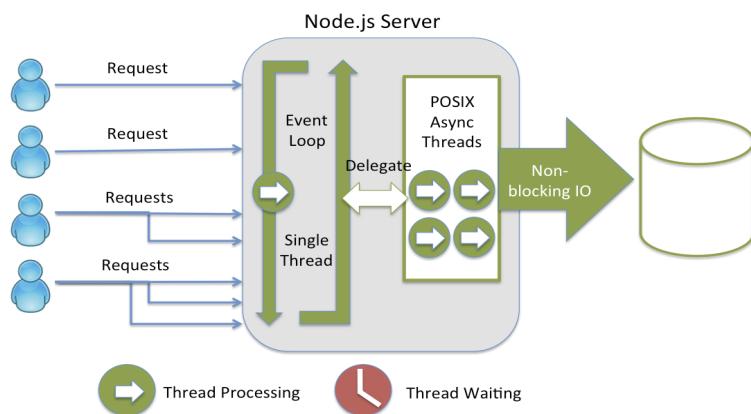


Abbildung 5: Single Threaded / Non Blocking I/O [Nodejs 1.1]

Der große Vorteil hierbei ist, dass der Hauptthread trotz der blockierenden Ein- und Ausgabeoperationen nicht anhält, und weitere Anfragen bearbeiten kann. (Non Blocking I/O - Prinzip)

### 2.3.2 Module

Module stellen in Node.js Software-Komponenten dar, die Objekte und Funktionen nach außen hin bereitstellen sollen. Sie können aus einer Skriptdatei oder einem Verzeichnis von Dateien bestehen. Module können als einzelne Default-Komponenten, die den Hauptteil des Moduls repräsentiert, exportiert werden. Bei der anderen Möglichkeit, des sogenannten ‚benannten Exports‘ werden die zu exportierenden Komponenten dagegen explizit angegeben. Letzteres ist in nachfolgender Abbildung dargestellt.

```

1 function foo(){}
2 function bar(){}
3
4 //Obige Funktionen exportieren:
5 module.exports.foo = foo;
6 module.exports.bar = bar;
```

Listing 1: Benannter Export von Modulen

Für den Import stehen verschiedene Möglichkeiten zur Verfügung. Im folgender Abbildung ist ein Import über die require()-Funktion dargestellt. Mit mitgeliefertem Modul-Pfad als Parameter gibt diese Funktion ein Objekt des Moduls wieder, das die exportierten Objekte (und Funktionen) enthält.

```

1 //Importieren der Funktion einer anderen Datei:
2 const foo = require('./module/path');
3 const bar = require('./module/path');
```

Listing 2: Import von Modulen

Eine wichtige Besonderheit ist, dass importierte Module beim ersten Aufruf gecached werden. Das bedeutet, dass jeder require()-Aufruf auf ein Modul dasselbe Objekt zurückliefert. [nodejs 1.21]

#### 2.3.2.1 npm

Ehemals als Node Package Manager bekannt, ist npm ein Paketmanager für Node.js, entwickelt 2010 von Isaac Z. Schlueter [nodejs 1.3] Es verwaltet ein öffentliches Repository (ein digitales Software-Verzeichnis im Internet) unter dem Name npm Registry. In dem Verzeichnis werden weit über 1 Millionen Pakete (Module) angeboten. [1.4] Der Großteil kann unter freier Lizenz verwendet werden. Mit npm können Module installiert, aktualisiert, entfernt und gesucht werden. Node.js liefert seit seiner Version 0.6.3 npm standardmäßig bei der Installation mit. [1.5]

#### 2.3.2.2 Express

„Express ist ein einfaches und flexibles Node.js-Framework von Webanwendungen, das zahlreiche leistungsfähige Features und Funktionen für Webanwendungen und mobile Anwendungen bereitstellt.“ [nodejs 1.6] Es wurde im November 2010 von Douglas Christopher Wilson und weiteren Entwicklern veröffentlicht und erweitert Node.js um das Abarbeiten verschiedener HTTP-Methoden, das separate Abarbeiten von Anfragen mit verschiedenen URL-Pfaden sowie weiterer nützlicher Möglichkeiten. Im Grunde handelt es sich bei Express um ein Modul, dass durch den npm Package Manager heruntergeladen werden kann. Die aktuelle Version zum Zeitpunkt der Dokumentation [??] ist 4.17.1. [nodejs 1.65]

## Beispiel

Das Erstellen einer einfachen Express-Applikation wird im folgenden Beispiel dargestellt:

```
1 const express = require('express');
2 const app = express();
3 const port = 3000;
4
5 app.get('/', (req, res) => {
6   res.send('Hello World')
7 });
8
9 app.listen(port, () => {
10   console.log("Example app listening on port ${port}!")
11});
```

Listing 3: Einfacher Webserver [nodejs 1.8]

Die require()-Funktion importiert das Express-Modul und gibt ein Express-Objekt zurück. Dieses Objekt als Funktion aufgerufen gibt wiederum ein Objekt der Express-Applikation zurück, welche traditionell „app“ genannt wird, das Kernstück des Express-Frameworks ist und sämtliche Methoden wie das Weiterleiten von HTTP Anfragen, das Konfigurieren von Middleware oder das Modifizieren des Webserver-Verhaltens beinhaltet. [nodejs 1.8]

Im mittleren Block befindet sich eine Routendefinition. Die app.get() Funktion spezifiziert eine Callback-Funktion, die ein „request“- und „response“-Objekt als Parameter erhält und aufgerufen wird, sobald eine HTTP Anfrage der Methode GET mit dem Pfad ,/‘ empfangen wird. Das Request-Objekt enthält sämtliche Informationen über die HTTP-Anfrage. Das Response-Objekt kann dagegen in der Callback-Funktion mit Informationen gefüllt werden und über die send()-Funktion als HTTP-Antwort an den Sender zurückgesendet werden.

Der unterste Block startet den Webserver auf dem mitgegebenen Port über die Funktion app.listen(). Ihr kann auch eine Callback-Funktion mitgegeben werden, die aufgerufen wird, sobald der Server erfolgreich gestartet ist.

**Middleware** Express arbeitet nach dem Middleware-Konzept. Darunter versteht man Funktionen, die für die Verarbeitung von Anfragen hintereinander geschaltet werden können. Jede Middleware hat Zugriff auf das Anfrageobjekt, das Antwortobjekt und die jeweils nächste Middleware-Funktion. [nodejs 1.9] Dabei kann die HTTP-Request direkt terminiert oder an die nächste Middleware gesendet werden. Die Verkettung der Middleware-Funktionen wird in folgender Abbildung illustriert:

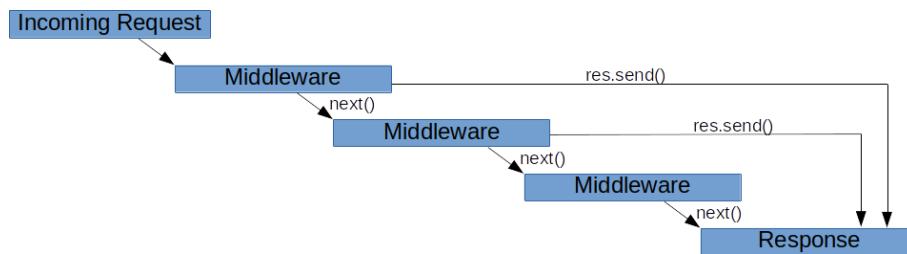


Abbildung 6: Middleware [nodejs 2.0]

**Middleware: express.json** Hierbei handelt es sich um eine in express eingebaute Middleware, die die in JSON formatierten Daten im Nachrichtenrumpf aus einer eingehenden HTTP-Anfrage grammatisch analysiert. Dabei ist zu beachten, dass der Nachrichtenrumpf nur dann analysiert wird, wenn bei der Anfrage eine Header-Informationen namens „Content-Type“ mit dem entsprechenden JSON-Typ als Wert übergeben wird. Nach erfolgreicher Analyse erstellt die Middleware aus den JSON-Informationen eine neue body-Objekt innerhalb des übergebenen request-Objekts. [nodejs 2.1]

```

1 const express = require('express');
2 const app = express();
3 app.use(express.json());
  
```

Listing 4: Express.json Middleware benutzen

**Middleware: Router** Unter dem Begriff Routing (Weiterleitung) versteht man im Kontext von Express „[...] die Definition von Anwendungsendpunkten (URIs) und deren Antworten auf Clientanforderungen.“ [nodejs 2.15]

Die in express eingebaute Middleware express.Router ermöglicht es, modular einbindbare Routenhandler (Weiterleitungsrouter) zu erstellen. Eine Router-Instanz ist als vollständiges Middleware- und Routingsystem zu sehen und wird deshalb auch als „Mini-App“ angesehen. Der sich durch die Modularität herausziehende Vorteil ist, dass folglich unterschiedliche Anwendungsendpunkte auf entsprechende Dateien ausgelagert werden können.

```

12 var express = require('express');
13 var router = express.router();
14
15 // middleware that is specific to this router
16 router.use(function timeLog(req,res,next) {
17   console.log('Time: ', Date.now());
18   next();
19 });
20
21 // define the home page route
22 router.get('/', function(req,res){
23   res.send('Birds home page');
24 });
25
26 // define the about route
27 router.get('/about', function(req,res){
28   res.send('About birds');
29 });
30 module.exports = router;

```

Listing 5: Routinghandler erstellen[nodejs 2.2]

In oberem Beispiel wird ein Routerhandler für das Verzeichnis „/birds“ mit eigen implementierter Middleware und zwei Anwendungsendpunkte „/“ (bezieht sich auf das Stammverzeichnis) und „/about“ erstellt. Der Code wird unter der Datei birds.js abgespeichert. Abschließend kann das Routermodul in die Anwendung geladen werden:

```

31 var birds = require('./birds');
32 ..
33 app.use('birds', birds);

```

Listing 6: Routinghandler benutzen[nodejs 2.2]

### 2.3.2.3 Mongoose

Mongoose ist ein öffentliches Modul, das zum Zeitpunkt der Dokumentation[TODO??] im npm Package Manager in der Version 5.12.3 zur Verfügung steht. [nodejs 2.4] Bei diesem Modul handelt es sich um ein Object-Document Mapper (ODM), der es ermöglicht, asynchron mit einer NoSql-Datenbank zu kommunizieren. Mongoose ist der populärste und am weitesten von MongoDB unterstützte ODM. [nodejs 2.55] Es unterstützt neben transparenter Persistenz auch die Datenvalidierung, das Erstellen von Abfragen (Queries), das Schreiben von logischem Business Code und die Übertragung zwischen Objekten im Code und der Repräsentierung dieser Objekte in der Datenbank.

**Object Document Mapping (ODM)** Object-Relational Mappers (ORM) finden hauptsächlich Einsatz in objektorientierten Anwendungen, dessen Daten in relationalen Datenbanken sind. Dabei werden die Tabellen in persistente Objekte gemappt. Das Mappen ist aber auch für NoSQL-Datenbanken nützlich. [nodejs 2.56] Die meistverbreiteten NoSQL-Datenbanken basieren auf Dokument-Systemen. Dementsprechend werden für diese Datenbanken Object-Document Mapper für das Mappen zwischen Dokumenten und Objekten genutzt. Einige ODM's sind Mongoose[nodejs 2.7], Morphia [nodejs 2.8], Doctrine[nodejs 2.9] und Mandango[nodejs 3.0]. NoSQL Mapper nutzen vom Entwickler definierte Datenschemata, die das Objekt beschreiben. Ein daraus abgeleitetes Model-Objekt ermöglicht dann die Kommunikation zwischen dem im Schema beschriebenen Objekt und der entsprechenden Datenbank-Collection.

**Schema** Mongoose-Schemata definieren die Struktur der gespeicherten Daten einer MongoDB-Collection in der Anwendungsschicht und werden in der JSON-Notation beschrieben. Dokumentenbasierte Datenbanken wie MongoDB enthalten für jede Wurzelentität eine Collection. Mongoose Schemata werden für jede Collection definiert. Innerhalb der JSON-notierten Schreibbeschreibung können den einzelnen Eigenschaften bestimmtes Verhalten zugeordnet werden. Zum Beispiel lässt sich explizit der Datentyp angeben (type), eine Eigenschaft verpflichtend (required) oder in Kleinbuchstaben einstellen (lowercase).

```

1 const schema = new Schema({
2   attributeX: {
3     type: String, // Datentyp
4     required: true, // Verpflichtendes Attribut?
5     lowercase: true; // Kleinbuchstaben?
6 })

```

Listing 7: Mongoose Schema - Beispiel

**Model** Ein Model in Mongoose ist ein aus einer Schemadefinition erstellter Konstruktor, aus denen Objekte instanziert werden können. Diese Instanzen werden auch ‚documents‘ genannt. Sie stehen in direkter Verbindung zu den jeweiligen Collections der verbundenen Datenbank und enthalten Methoden für die persistente Speicherung, Bearbeitung oder Löschung. Beispielsweise wird beim Abspeichern einer Mongoose Instanz eines Models die entsprechende Collection in der Datenbank erzeugt, sofern sie noch nicht vorhanden ist. Eine Konvention in Mongoose sieht vor, dass der Name eines Models dem Singular eines Nomens entspricht, während die Collections nach dem Plural dieses Namens beschrieben werden. [nodejs 3.2] Im folgenden Beispiel wird ein Model über die mongoose.model()-Funktion erstellt unter Angabe des Modelnamens und dem zu verwendenden Schema. Dieses Model wird über module.exports nach außen zur Verfügung gestellt.

```

1 const mongoose = require('mongoose');
2 const testSchema = new mongoose.Schema({
3   attributeX: {
4     type: String,
5     required: true,
6     lowercase: true
7   }
8 });
9 module.exports = mongoose.model('test', testSchema);

```

Listing 8: Model erstellen und exportieren

An anderer Stelle kann das Model nun importiert werden. Aus dem Model kann ein Objekt instanziert werden, welches über die save()-Funktion in der Datenbank gespeichert werden kann.

```

1 const testModel = require(test);
2
3 var testInstanz = new testModel();
4 await testInstanz.save();

```

Listing 9: Model importieren - Objekt instanziieren und persistent speichern

Mongoose Models enthalten ohne Instanziierung des Weiteren auch Schnittstellen, um Daten der zugehörigen Collection zu kreieren, abfragen, bearbeiten oder löschen. (Create, Receive, Update, Delete oder auch kurz CRUD).

```

1 const testModel = require(test);
2
3 //Create
4 testModel.Insert({attributeX: "abc"})
5 //Receive
6 var testObjects = await testModel.find();
7 var testObject = await testModel.findOne({attributeX: "abc"})
8 //Update
9 await testModel.updateOne({X: "abc"}, {X: "cba"});
10 //Delete
11 await testModel.deleteMany({X: "abc"})

```

Listing 10: CRUD-Beispieldfunktionen eines Mongoose-Models

Tabelle 3: Module

Express-Modul	Beschreibung
fs	Erlaubt die Interaktion mit dem Dateisystem. Zum Beispiel Schreiben/Lesen von Dateien.
http	Ermöglicht Datentransfer über das Protokol HTTP und das Abhören eines Ports.
https	Gesicherte Variante zu HTTP mit SSL. Benötigt Private Key und Zertifikat.
firebase-admin	Ermöglicht die Verbindung zu Google Firebase Cloud.
node-cron	Ermöglicht das Einstellen von sich wiederholenden Aufgaben zu bestimmten Zeitintervallen.

**Verbindung** Verbindung zur Datenbank kann über die connect()-Funktion mit Angabe der genutzten Datenbank und des Datenbankpfads hergestellt werden. Über das mongoose.connection-Objekt können auf Verbindungsereignisse reagiert werden.

```

1 const mongoose = require('mongoose');
2 await mongoose.connect("mongodb://127.0.0.1:27017/TestDB");
3 mongoose.connection.on('error',(error) => console.log(error));
4 mongoose.connection.on('open',() => console.log('Connected'));

```

Listing 11: Mongoose: Verbindung zur Datenbank aufbauen

Für den Verbindungsauflauf können weitere Optionen übergeben werden. Dafür kann ein Objekt wie in folgendem Beispiel erstellt werden, dass die zugehörigen Optionen als Attribute beinhaltet.

```

1 const options = {
2   useNewUrlParser: true,
3   useUnifiedTopology: true,
4   useCreateIndex: true,
5   useFindAndModify: false,
6   autoIndex: false,
7   poolSize: 10, // Anzahl der max. Socket Connections
8   serverSelectionTimeoutMS: 5000, // TimeOut bis verbunden
9   socketTimeoutMS: 45000, // Schliesse Socket bei 45s
10   // Inaktivitaet
11   family:4 // Use IPv4
12 }

```

Listing 12: Mongoose Verbindungsoptionen -nodejs 3.3-

### 2.3.2.4 Weitere Module

## 2.4 Representational State Transfer - Application Programming Interface

Hier steht mein Database Text.

Es gibt verschiedene Datenbankmodelle.

## 2.5 NoSQL-Datenbank

Unter NoSQL („Not only SQL“) werden Datenbanksysteme bezeichnet, die einen nicht-relationalen Ansatz verfolgen. Im Vergleich zu relationalen Datenbanken, welche die Daten in tabellenförmigen Strukturen mit Spalten und Zeilen speichern, nutzt eine NoSQL-Datenbank andere Strukturkonzepte für die Speicherung der Daten wie zum Beispiel Wertpaare, Dokumente, Objekte oder Listen und Reihen. Da NoSQL einige der bekannten Schwächen von relationalen Datenbanken, wie Performance-Schwierigkeiten bei hohem Lastaufkommen oder bei dem Umgang mit großen Datenmengen, vermeidet, erfreut sich diese Technologie in der heutigen Zeit des großen Datenaufkommens immer größerer Beliebtheit. [NoSQL 1.1] Zu den bekanntesten NoSQL-Datenbanken gehören beispielsweise Apache Cassandra, MongoDB und CouchDB.

### 2.5.1 NoSQL-Datenbanktypen

NoSQL-Datenbanken werden hauptsächlich in vier verschiedene Kategorien unterteilt, die unterschiedliche Konzepte verfolgen.

**Graphendatenbanken** Dieses Datenbankkonzept speichert die Informationen in Netzstrukturen, den sogenannten Graphen ab. Die einzelnen Informationselemente werden durch Knoten mit Eigenschaften repräsentiert. Um die Beziehungen zwischen den Knoten darzustellen, werden Kanten genutzt, die gerichtet und benannt sein können und ebenfalls Eigenschaften besitzen.

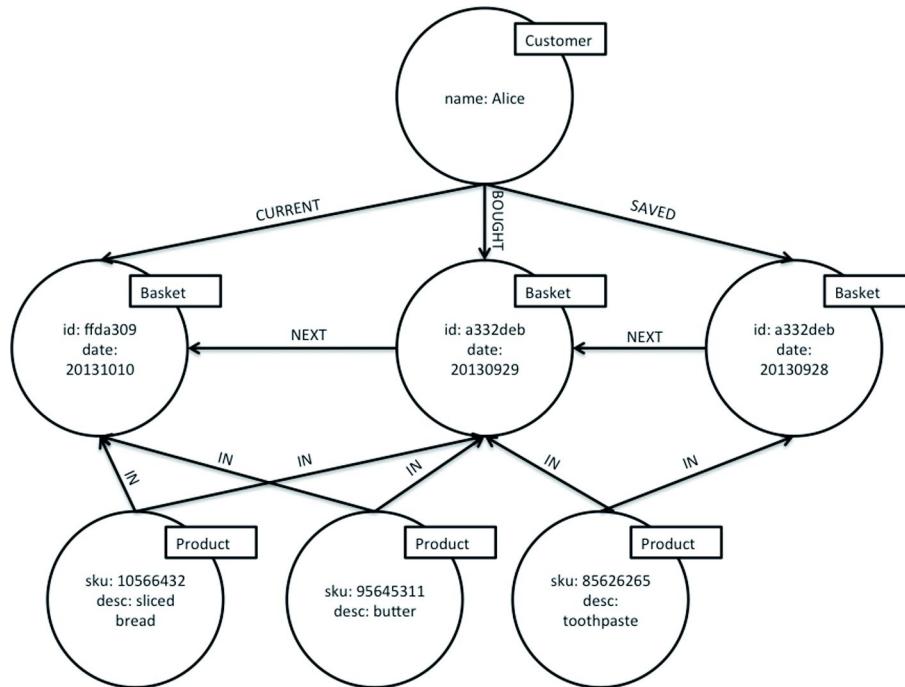


Abbildung 7: Graphikdatenbank Beispiel [NoSql 1.2]

**Dokumentenorientierte Datenbanken** Im Kontext der dokumentenorientierten Datenbanken sind Dokumente Objekte mit Eigenschaften, die in einer Sammlung gespeichert werden. Während eine Sammlung eine Tabelle im relationalen Datenbank widerspiegelt, ist ein Dokument als ein Eintrag beziehungsweise einer Zeile dieser Tabelle gleichzusetzen, mit dem großen

Tabelle 4: Key-Value Beispiel [NoSql 1.5]

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015

Unterschied, dass dokumentenorientierte Datenbanken schemafrei sind und kein bestimmtes Datenschema voraussetzen. Dokumente können weitere Dokumente als Attribut oder in einer Liste einbetten und bieten so die Möglichkeit, komplexe Datenstrukturen zu speichern. In aktuellen Datenbanksystemen wie CouchDB und MongoDB nutzen diese Dokumente Datenformate wie JSON oder XML.

```

1 {
2   "Vorname": "Max",
3   "Nachname": "Mustermann",
4   "Telefon-Nr": "0124567",
5   "Alter": 33,
6   "Adresse": "Musterstrasse 34, Musterstadt",
7   "Kinder": ["Junior", "Elenor"]
8 }
```

**Key-Value-Datenbanken** Key-Value-Datenbanksysteme bilden mit einer Abbildung der Daten in Schlüssel- und Wertpaare die einfachste NoSQL-Datenbankumsetzung ab. Bei diesem Konzept werden eindeutigen Schlüsselattributen (Key) jeweils ein beliebiger Wert (Value) zugeordnet.

**Spaltenorientierte Datenbanken** Spaltenorientierte Datenbanken, oder auch „Wide-Column“-Datenbanken genannt, speichern ihre Datensätze in Form von Tabellen. Sie wirken zunächst den Tabellen der relationalen Datenbanksysteme sehr ähnlich, unterscheiden sich grundlegend aber in der Speicherung der Daten, die nicht zeilenorientiert, sondern spaltenorientiert abgelegt werden.

Artikelnummer	Artikelbezeichnung	Umsatz Tsd.EUR
1	Buntlack RAL 7035	25
2	Heizkörperfarbe	50
3	Grundierfarbe	15

Abbildung 8: Spaltenorientierte Datenbank Beispiel [NoSql 1.5]

Nachfolgend ist die zeilen- und die spaltenorientierte Speicherung dargestellt. Zeilenorientierung bei der Speicherung bietet vor allem Vorteile bei Abfragen, bei denen Informationen aus mehreren Spalten benötigt werden. Spaltenorientierung dagegen eignet sich sehr gut für die Auswertung von Aggregaten.

zeilenorientierte Speicherung

1	Buntlack RAL 7035	25	2	Heizkörperfarbe	50	3	Grundierfarbe	15
---	-------------------	----	---	-----------------	----	---	---------------	----

spaltenorientierte Speicherung

1	2	3	Buntlack RAL 7035	Heizkörperfarbe	Grundierfarbe	25	50	15
---	---	---	-------------------	-----------------	---------------	----	----	----

Abbildung 9: Zeilen- und spaltenorientierte Speicherung [NoSql 1.5]

## 2.5.2 BASE

TODO

## 2.5.3 CAP-Theorem

Der Informatiker Eric Brewer von der Universität Berkeley stellte Anfang des Jahres 2000 die Annahme auf, dass ein System nicht gleichzeitig die drei Kerneigenschaften Consistency (Konsistenz), Availability (Verfügbarkeit) und Partition Tolerance (Ausfalltoleranz) abdecken kann [NoSQL 1.6].

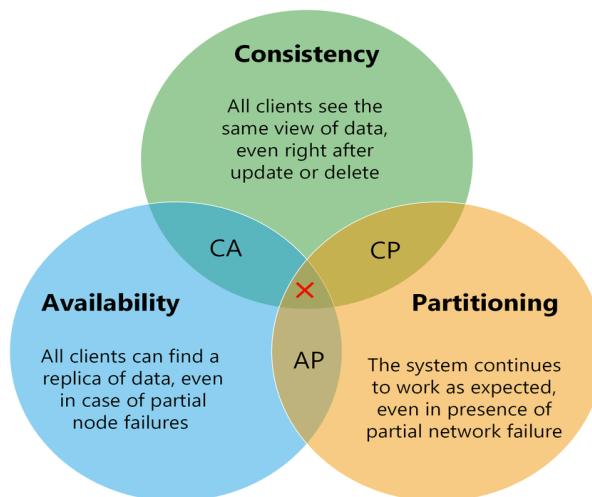


Abbildung 10: Visualization-of-CAP-theorem [NoSql 1.67]

Die **Konsistenz (C)** beschreibt, dass Datenzustände, die in einem verteilten System geändert werden, in jedem zusammenhängenden System gleich sein müssen. Der Zustand der Daten soll somit im gesamten System übereinstimmen. Ein System, das einen ununterbrochenen Betrieb und eine akzeptable Antwortzeit aufweisen kann, besitzt eine hohe **Verfügbarkeit (A)**. Die **Ausfalltoleranz (P)** steht für ein Verhalten, bei dem es bei Ausfallen eines Bestandteiles innerhalb eines Systems zu keinem Gesamtausfall kommt.

## 2.5.4 MongoDB

MongoDB ist ein in C++ geschriebenes, dokumentenorientiertes NoSQL-Datenbanksystem, das im Jahre 2009 von den Entwicklern Horowitz und Merriman als Open-Source Datenbank veröffentlicht wurde und die am weitest-verbreitete NoSQL-Datenbank. (Stand April 2021) [MongoDB1.7] Die Intention der Gründer war es, eine Datenbank mit höherer Skalierbarkeit, Flexibilität und Performance zu entwerfen, die auf einer einfachen Handhabung beruht. [MongoDB1.65] Gründe der Popularität der Datenbank ist neben den oben erwähnten Eigenschaften die flexible Gestaltungsmöglichkeit der Datenstrukturen sowie die Unterstützung durch zahlreiche Programmiersprachen und Betriebssysteme.

Dem Konzept des CAP-Theorems folgend steht MongoDB für Konsistenz und Partitionstoleranz, dafür ordnet sich die Verfügbarkeit den anderen Eigenschaften unter.

### 2.5.4.1 Struktur

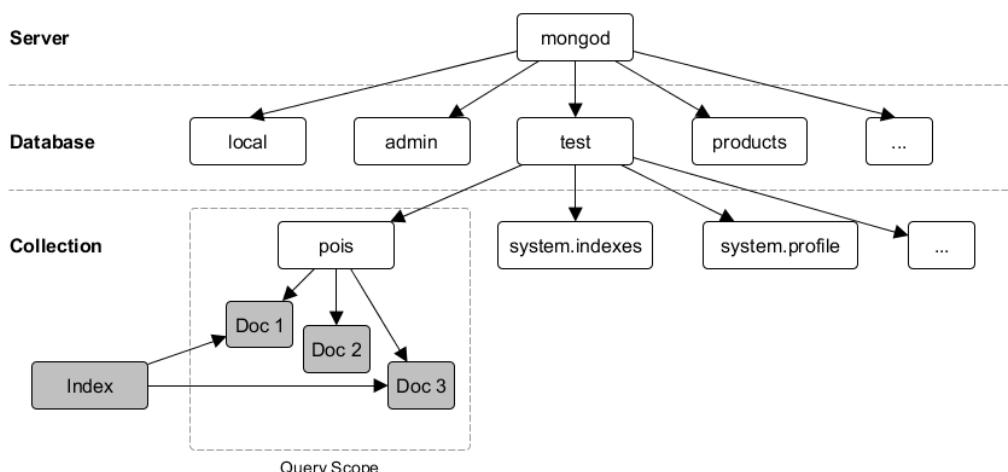


Abbildung 11: Graphikdatenbank Beispiel [NoSql 1.9]

Letztere Abbildung stellt die grundsätzliche Struktur einer MongoDB-Instanz dar. Ein **Server** kann mehrere logische **Datenbanken** verwalten, die ihrerseits einen oder mehrere logische Namensräume enthalten, die sogenannten **Collections**. Eine Collection verwaltet die einzelnen Datensätze, die als **Dokumente** bekannt sind.

**Schema-Freiheit** Collections sind schemafrei. Dadurch gibt es für die zugehörigen Dokumente kein vorausgesetztes Schema. Aufgrund der Schema-Freiheit dürfen dennoch Architekturentscheidungen bei der Datenmodellierung nicht ignoriert werden. Stattdessen werden diese Entscheidungen des Schema-Managements auf die Anwendungsentwicklung verlagert.

**BSON** Während MongoDB für Datenaustausch das JSON-Format nutzt, hält es seine Dokumente im Binary JSON-Format (BSON), einer binärcodierter Erweiterung des JSON-Formats. Daten im BSON-Format enthalten zusätzlich Informationen zum Typ und zur Länge der Informationen, wodurch schnelleres Parsen von Daten möglich ist. Des Weiteren ist BSON um zusätzliche Datentypen wie 32- und 64-bit Integer oder das Datum erweitert. [MongoDb1.8]

<https://www.mongodb.com/json-and-bson>

```

1  /* JSON */
2 { "hello": "world" } // "key": "value"
3
4  /* BSON */
5 \x16\x00\x00\x00          // total document size
6 \x02                      // 0x02 = type String
7 hello\x00                  // field name
8 \x06\x00\x00\x00world\x00  // field value
9 \x00                      // 0x00 = type E00 ('end of object')

```

**ObjectID und Primärschlüssel** Für die eindeutige Identifikation eines Dokuments vergibt MongoDB automatisch erstellte '\_id'-Felder. Dabei wird bei der Generierung des BSON-Dokuments für den Wert das '\_id'-Feld ein Datentyp 'ObjectId' hinzugefügt. Dieses Identifikationsfeld ist gleichzusetzen mit den Primärschlüsseln aus relationalen Datenbanken. [MongoDB1.85] Ist eine automatische Generierung des eindeutigen Identifikationsfeldes nicht erwünscht, kann der Anwendungsentwickler dieses Feld auch selbst erzeugen, indem er die Eigenschaft explizit dem zu erstellenden Objekt hinzufügt und mit einem Wert verseht.

#### 2.5.4.2 Datenbankabfrage

Im Vergleich zu relationalen Datenbanken benutzt MongoDB keine Abfragesprache wie SQL. Stattdessen ermöglicht dieses Datenbanksystem drei Arten von Abfragen, wobei eine Abfrage sich immer auf genau eine Collection bezieht. Ein Bezug einer Abfrage zu mehreren Collections, wie es relationale Datenbanken mit „Join“-Operationen erlauben, ist hier nicht gegeben und muss in der Anwendung realisiert werden. [1.9] Abfragen in MongoDB werden grundsätzlich in Form von Dokumenten formuliert. Dieses Verfahren ermöglicht schnelles Abhandeln komplexer Abfragen von tief geschachtelten Dokumentenstrukturen. Nachfolgend werden die drei Abfragemöglichkeiten erläutert.

**Query-By-Example** Das folgende Beispiel zeigt einen Aufruf aller Einträge, die als Ort Karlsruhe eingespeichert haben.

```
1 db.pois.find( {"adresse.ort": "Karlsruhe" } )
```

Listing 13: MongoDB Read

Dabei kommt das Prinzip Query-by-Example [MongoDB2.0] zum Einsatz, bei dem die als JSON-Dokument beschriebenen Suchkriterien als Filter auf die durchsuchte Collection wirkt. Zusätzlich stehen für diese Art von Abfragemöglichkeit sämtliche logische Verknüpfungen und Vergleichsoperatoren zur Verfügung. [ siehe MongoDB2.1] Das Ergebnis der obigen Find-Operationen liefert dabei einen Cursor zurück, über den die aufrufende Anwendung durch die einzelnen zurückgelieferten Dokumente iterieren kann. Außerdem kann der Cursor modifiziert werden, um beispielsweise Beschränkungen der Trefferanzahl oder Sortierung vorzunehmen.

```
1 db.pois.find().limit(5).sort({ "adresse.ort": -1 })
```

Listing 14: MongoDB Read Modifikation

Tabelle 5: Aggregation Framework: Pipeline-Operatoren [MongoDB1.9]

Operator	Beschreibung
\$match	Sucht Dokumente analog zu find(). Sollte idealerweise mindesten 1x zu Beginn der Pipeline ausgeführt werden, um die Ergebnismenge einzuschränken.
\$project	Schränkt auf eine Teilmenge von Feldern ein und verändert die Feldwerte.
\$sort	Sortiert die Dokumente. Analog zu sort() bei find(). . Benötigt Private Key und Zertifikat.
\$skip	Überspringt n Dokumente. Analog zu skip() bei find().
\$limit	Begrenzt auf n Dokumente. Analog zu limit() bei find().
\$group	Gruppert nach einem oder mehreren Feldern.
\$unwind	Wird auf ein Array angewendet. Jeder Array-Eintrag generiert dann ein neues Dokument für die nächste Pipeline-Stufe.
\$redact	Filtert Felder des Dokuments in Abhängigkeit vom Inhalte anderer Felder.
\$out	Leitet das Ergebnis der Aggregation in eine Collection um. Kann nur als letzter Operator verwendet werden.

**MapReduce** MapReduce ist ein allgemeines Programmiermodell zur verteilten und parallelen Verarbeitung von großen Datenmengen in aggregierte Ergebnisse. Dabei unterteilt sich der Algorithmus im Wesentlichen in zwei Operationen. [MongoDB2.3]

- Map: Emittieren der beliebig vielen Key-Value-Paare für jedes Dokument
- Reduce: Zusammenfassen aller Daten, die einem Kriterium entsprechen.

**Aggregation-Framework** Das Aggregation-Framework bietet als Alternative zum MapReduce-Verfahren den Vorteil der besseren Performance. [MongoDB2.4] Dabei wird eine Pipeline genutzt, die das resultierende Dokument einer Operation an die Eingabe der nächsten Operation weiterleitet. Die entsprechende Funktion ist die Aggregate()-Operation, deren Parameter als Array von Dokumenten die Pipeline-Operatoren abbilden. [MongoDB2.5]

```

1 db.pois.aggregate([
2   {$group: {_id: "$adresse.ort", n: {$sum:1}}},
3   {$sort: {n: -1}}
4 ])

```

Listing 15: MongoDB Aggregate

Es stehen folgende Pipeline-Operatoren zur Verfügung:

Tabelle 6: Aggregation Framework: Pipeline-Operatoren [MongoDB1.9]

Operation	Beschreibung	MongoDB Methode
Create	Datensätze erstellen	.Insert()
Read	Datensätze lesen	.Find()
Update	Daten aktualisieren	.Update()
Delete	Datensätze entfernen	.Remove()

#### 2.5.4.3 CRUD

TODO TABLE :( MongoDB unterstützt eine Vielzahl an Operationen zum Erzeugen, Lesen, Updaten und Löschen von Daten. Über folgende Kommandos lassen sich die CRUD-Operationen realisieren:

**Create** „InsertOne()“ erzeugt ein einzelnes Dokument in der jeweiligen Collection der Datenbank. Dabei wird ein Dokument im JSON-Format als Parameter mitgegeben. ObjectId wird, wenn nicht als Eigenschaft im Dokument angegeben, automatisch von MongoDB erzeugt. Ebenfalls wird die angesprochene Collection automatisch erzeugt, falls sie nicht bereits existiert. Die Funktion „InsertMany()“ erlaubt das Hinzufügen mehrerer Datensätze über ein Array von JSON-Dokumenten als Parameter. Die Methode „Insert“ ist flexibler und bietet beide Parametrisierungsmöglichkeiten. [Mongo2.55 ]

```
1 db.personen.insertOne({ "vorname" : "Robin"});
```

Listing 16: MongoDB Create

**Read** Um Datensätze aus der Datenbank zu lesen, wird die find()-Methode zur Verfügung gestellt. Sie gibt alle Dokumente einer Kollektion zurück. Wie im Kapitel Query-By-Example beschrieben, kann die Funktion mit Filtern versehen werden. Die Funktion findOne() bietet die gleiche Funktionsweise, die Dokumentenausgabe ist aber auf ein einzelnes Dokument begrenzt. Ein Beispiel ist dargestellt in Listing 13.

**Update** Die Update()-Methode ermöglicht es, Änderungen an Datensätzen vorzunehmen. Die Methode erhält zwei Parameter, einen zur Angabe der gesuchten Dokumente und den anderen mit der vorzunehmenden Änderung. Simultan zu den Insert-Methoden gibt es die UpdateOne()-Methode für Änderungen an einem Dokument und UpdateMany()-Methode für mehrere Dokumente.

```
1 db.personen.updateOne({ "vorname" : "Robin"} , currentDate("last_login") );
```

Listing 17: MongoDB Update

**Delete** Für das Löschen von Datensätzen aus einer Collection gibt es die DeleteOne()- und DeleteMany()-Methode. Über mitgegebenen Parameter können Filter eingestellt werden. Eine ganze Collection kann mit der drop()-Methode entfernt werden.

```
1 db.personen.deleteOne( {"vorname" : "Robin"} );
```

Listing 18: MongoDB Remove

#### 2.5.4.4 Atomare Operationen

Als atomare Operationen wird ein Verbund von Einzeloperationen bezeichnet, der als logische Einheit betrachtet wird und nur als Ganzes erfolgreich abläuft oder fehlschlägt. In Bezug auf Datenbanken spricht man von Transaktionen, die entweder als Ganzes erfolgreich ablaufen (Commit) oder nach einer fehlerhaften Einzeloperation rückgängig gemacht werden (Rollback). Ohne atomare Operationen können Probleme auftreten, die zu einer Inkonsistenz der Datenzustände führen. Beispielsweise würde ein Abbruch inmitten mehrerer zusammenhängender Operationsabläufe dazu führen, dass nur ein Teil der Operationen ausgeführt wurde, während die restlichen Operationen und somit die verbleibenden Datenänderungen verworfen werden.

Bis vor dem Jahre 2018 unterstützte MongoDB keine Transaktionen. Mit der Veröffentlichung der Version 4.0 wurde der Funktionsumfang von MongoDB stark erweitert. Eines der neuen Erweiterungen war die Möglichkeit, Transaktionen durchführen zu können. Dafür werden sogenannte „Sessions“ genutzt. Sämtlichen CRUD-Operationen, die einer Transaktion zugehören sollen, werden eine Session als zusätzlicher Parameter hinzugefügt. Bei Abschluss der Transaktion kann sie mit der Methode „session.commitTransaction()“ bestätigt werden. Andernfalls, sollte ein Fehler aufgetreten sein, kann die Methode „session.abortTransaction()“ die Operationen rückgängig machen. Eine wichtige Voraussetzung, um Transaktionen in MongoDB nutzen zu können, ist ein Replica Set aufzusetzen. Darauf wird im weiteren Kapitel eingegangen.

#### 2.5.4.5 Architektur

Bei der Inbetriebnahme einer MongoDB Datenbank spielen zwei Prozesse eine wichtige Rolle. Der „mongod“-Prozess ist der primäre Hintergrundprozess für das Datenbanksystem. Er behandelt Datenabfragen, Datenzugriffe und führt benötigte Hintergrundoperationen durch. [mongodb3.0] Beim Starten des mongod-Prozesses können über Flags Konfigurationen vorgenommen werden, beispielsweise ändert „–port 5000“ den Port. Nach erfolgreichem Start des Prozesses kann über den Standard Port der MongoDB (27017) bzw. über den konfigurierten Port eine Verbindung hergestellt werden.

Der Prozess, der als Controller für sich auf mehreren Datenbanken befindenden verteilten Daten dient, nennt sich „mongos“. [mongodb3.1] Dieser Prozess findet seinen Einsatz hauptsächlich in Kombination mit Sharding, auf die im weiteren Unterkapitel eingegangen wird.

**Engine** Als Storage Engine, oder auch Datenbank-Engine, wird die zugrundeliegende Softwarekomponente eines Datenbanksystems zum Verwalten der Daten bezeichnet. Die gebräuchlichsten Engines in MongoDB sind die MMAPv1-Engine und die WiredTiger-Engine.

Bis vor Version 3.2 war MMAPv1-Engine die Standard-Storage Engine von MongoDB. Eine Kompression der Daten wurde nicht unterstützt und Transaktionen waren nur auf einem Dokument ausführbar. Ab MongoDB 3.2 wird standardmäßig die Wired Tiger-Engine eingesetzt. Neben der Kompression der Daten und der miteingehenden Verringerung des benötigten Speicherplatzes bietet diese Engine auch eine Verschlüsselung der Daten an. Abhängig des benutzten

Kompressionsalgorithmus kann der benötigte Verbrauch um 70% für Daten und um 50% für Indizes reduziert werden. [Mongo3.5] Die WiredTiger-Engine skaliert im Vergleich zur MMAPv1 mit der Anzahl der CPU-Kerne und ermöglicht Transaktionen über mehrere Dokumente hinweg. Die genutzte Engine kann in den Konfigurationen eingestellt werden.

**Replica Sets** Im standardmäßigen Standalone-Modus besteht bei Ausfall des Servers die potenzielle Gefahr des Datenverlusts. Das Problem lässt sich durch Replikation beheben. Hierbei spricht man von der bloßen Herstellung von Mehrexemplaren(Kopien) derselben Daten, die meistens regelmäßig abgeglichen werden. [3.6] MongoDB realisiert die Replikation der Daten über Replica Sets. [3.7] Dabei handelt es sich um eine Gruppe von mongod-Prozessen, die dieselben Daten enthalten. Nach dem CAP-Theorem ist MongoDB nicht auf Verfügbarkeit ausgelegt. Dennoch umgeht die Datenbank diesen Nachteil über die Anwendung von Replica Sets. Das Replica Set besteht aus einem primären Knoten und daraus replizierenden Sekundären Knoten. Nur der primäre Knoten führt datensatzändernde Operationen aus. Veränderungen werden in sogenannten 'Oplogs' (Operations logs) gespeichert, die zum Austausch der Daten innerhalb des Replica Sets genutzt wird. [3.8] Bei den Oplog-Dateien handelt es sich um Collections mit festgelegten Speichergrößen, den sogenannten „capped collections“. Während die Oplog-Dateien ähnlich wie ein Logbuch mit allen Veränderungen in den Datensätzen der Datenbank beschrieben werden, werden bei erreichter Maximalgröße der Collection die ältesten Daten von den neuen Daten überschrieben. Dabei erhält jeder Knoten des Replica Sets seine eigene Kopie des Oplogs und gleicht ihn mit dem des primären Knotens ab.

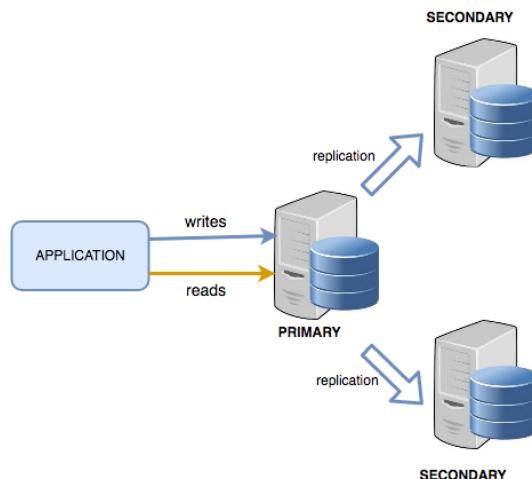


Abbildung 12: MongoDB Replica Set [3.8]

**Sharding** Hierbei ist die Rede von der Methodik zur Aufteilung der Daten auf mehrere Datenbanken.[4.1 ] Der Datenbestand wird nach logischen Kriterien in Teilstücke, die sogenannten „Shards“, zerlegt und über mehrere Replica Sets verteilt. In Hinblick auf das hinzufügen weiterer Server gewährt das Sharding eine horizontale Skalierbarkeit. Des Weiteren werden durch die Tatsache, dass jeder Shard für seinen Bestandteil zuständig ist, Suchanfragen schneller durchgeführt. Sharding ist besonders bei großen Datenmengen in Bezug auf Performance von Vorteil. Der mongos-Prozess leitet Anfragen an die jeweiligen Shards weiter und dient als Schnittstelle zwischen den Clientanwendungen.

The screenshot shows the MongoDB Compass interface for the 'test.books' collection. At the top, it displays statistics: DOCUMENTS 5, TOTAL SIZE 345B, AVG. SIZE 69B; INDEXES 1, TOTAL SIZE 32.0KB, AVG. SIZE 32.0KB. Below this is a toolbar with FILTER, ADD DATA, VIEW, and various search/refresh buttons. A green FIND button is prominent. The main area shows three document entries:

- `_id: 7000  
author: "Homer"  
copies: 10`
- `_id: 7020  
title: "Iliad"  
author: "Homer"  
copies: 10`
- `_id: 8645  
title: "Eclogues"  
author: "Dante"  
copies: 2`

At the bottom right, it says "Displaying documents 1 - 5 of 5".

Abbildung 13: CRUD-Bespielfunktionen eines Mongoose-Models

#### 2.5.4.6 Verwaltungswerkzeuge

**Mongo Shell** Der „mongo“-Prozess ist eine interaktive JavaScript Schnittstelle auf der Kommandozeile. Er bietet umfassende Funktionalitäten für die Systemadministration des Datenbankensystems als auch Zugriff auf die Datensätze. [mongo3.2] Dazu erhält man eine Eingabeaufforderung, auf dem Befehle in der Sprache JavaScript ausgeführt werden können.

**Treiber** MongoDB bietet für viele Programmiersprachen bzw. Frameworks Softwarebibliotheken zum Zugriff auf die Datenbank an. [4.3 <https://docs.mongodb.com/drivers/>] Eine offiziell unterstützte ist Mongoose für das Node.js-Framework [4.4] <https://mongoosejs.com/docs/doubled?>

**Grafische Oberflächen** Es gibt einige Anwendungen zur visuellen Darstellung und Bearbeitung der Datenbanken in MongoDB, die eine grafische Benutzeroberfläche bieten. Zum Beispiel MongoDB Compass, Studio3T oder Fang of Mongo.

## 2.6 Firebase

Firebase ist eine Backend as a Service-Plattform (BaaS) von Google für mobile und Web-Anwendungen. Sie soll es dem Entwickler ermöglichen, einfacher und effizienter Funktionen auf verschiedenen Plattformen bereitzustellen statt Tools und Infrastruktur zur Verfügung. Mit dem Firebase SDK bietet die Plattform API Schnittstellen zu den jeweiligen Tools, welche direkt in die Anwendung integriert werden können, ohne dass serverseitiger Code dafür notwendig ist. Die Firebase Inc. wurde 2011 von James Tamplin und Andrew Lee gegründet und letztendlich 2014 von Google übernommen.<sup>4</sup> Teile der SDK stehen seit der Google I/O 2017 unter der Apache 2.0 Lizenz, sind somit also Open-Source.<sup>5</sup>

Es existieren zwei Kostenmodelle für die Nutzung von Firebase: Ein kostenloses Modell „Spark Plan“ und ein pay-as-you-go „Blaze Plan“. Das kostenlosen Modell beinhaltet die wichtigsten Tools, viele dieser Tools sind jedoch begrenzt durch beispielsweise Bandbreite oder Speicherplatz. Der Pay-as-you-go Plan ist eine Erweiterung des kostenlosen Plans. Er bietet daher das Nutzen von Tools bis zu einem gewissen Limit kostenfrei an; darüber hinaus kostet es jedoch dann pro Nutzung.

Ein Firebase Projekt ist die oberste Ebene in Firebase. Ein Projekt ist letztendlich ein *Google Cloud Projekt*, welches mit speziellen Konfigurationsmöglichkeiten und Services ausgestattet ist. Es beinhaltet die Verknüpfung zu den einzelnen Anwendungen (also bspw. Android-, iOS- oder Webanwendung). Nun können variabel Tools, sog. Firebase products hinzugefügt werden. Diese Produkte lassen sich grundlegend in drei Kategorien einteilen. Die hier relevantesten werden im Folgenden besprochen.[13]

### 2.6.1 Firebase Authentifizierung

Die Authentifizierung gehört zu den „Build“Produkten und bietet eine Token-basierte Nutzerauthentifizierung. Hierbei kann zwischen verschiedenen Anmeldeoptionen gewählt werden: klassisch mit E-Mail und Passwort, mit OAuth2.0 Integration für Social Media (Google, Facebook, Twitter, Github, ...) oder per Telefonnummer. Jeder Nutzer erhält eine einzigartige ID und ein zugehöriges Nutzerobjekt in einer NoSQL Datenbank. Grundlegende Werte wie E-Mail Adresse oder Name können hier abgespeichert werden; zusätzliche Informationen müssen über einen weiteren Datenbank Service abgespeichert werden. Für die Verwaltung eines Accounts bietet dieses Tool auch eingebaute E-Mail Aktionen an - bspw. Passwort zurücksetzen oder E-Mail Adresse bestätigen.

Ein Firebase Nutzer Objekt repräsentiert den Account eines Nutzers, welcher sich von einer Anwendung aus beim zentralen Firebase Projekt angemeldet hat. Die Instanz eines Firebase Nutzers ist somit unabhängig von der Authentifizierungsinstanz der Anwendung, also kann eine Anwendung mehrere Nutzer anmelden, jedoch kann sich auch ein Nutzer auf mehreren Anwendungen anmelden. Ist ein Nutzer authentifiziert, erhält die Anwendung eine Referenz des Nutzers, welche so lange existiert, bis er wieder abgemeldet ist.[13]

### 2.6.2 Cloud Firestore

Als Datenbank Lösung bietet Firebase zwei unterschiedliche Produkte an: Cloud Firestore und Realtime Database. Firestore ist hier neuer, jedoch ersetzt es Realtime Database nicht.

Cloud Firestore ist eine flexible und auf Skalierung ausgesetzte NoSQL Cloud Datenbank, wel-

---

<sup>4</sup>[firebase.googleblog.com](https://firebase.googleblog.com), zuletzt aufgerufen am 03.05.2021

<sup>5</sup>[opensource.googleblog.com](https://opensource.googleblog.com), zuletzt aufgerufen am 03.05.2021

che unter anderem die Echtzeitsynchronisierung der Daten zwischen Anwendung und Server ermöglicht. Zusätzlich zu REST und RPC APIs in iOS, Android und web SDKs ist Firestore auch in nativen Node.js, Java, Python und Go SDKs verfügbar.

Das Datenmodell ist hierarchisch aufgebaut, wobei Daten in Dokumenten (documents) und Dokumente in Sammlungen (collections) gespeichert sind. Mit Hilfe von Sammlungen werden die Daten voneinander abgetrennt und hierüber können Abfragen erstellt werden. Grundlegende Datentypen sind String, Integer und Boolean, jedoch können auch komplexe Datentypen wie Maps, Arrays oder Geopoints. Unter Sammlungen und darin verstaute Dokumente sind ebenfalls möglich.

Abfragen werden auf Dokumentenebene erstellt, damit nicht eine gesamte Sammlung aufgerufen werden muss. Dies kann über direkte Sortierung, Filter und/oder Limitierung bzw. genaue Auswahl eines Dokumentes bewerkstelligt werden. Bei einer Abfrage erhält man einen *Data Snapshot*, wodurch über Änderungen in Echtzeit informiert und diese angezeigt werden können. Damit es jedoch zu keinen fehlerhaften Daten führt, gelten hier atomare Eigenschaften für Transaktionen. Eine Transaktion ist eine Folge von Datenbankanweisungen, welche entweder alle gemeinsam oder gar nicht ausgeführt werden. Eine Transaktion ist nur dann erfolgreich, wenn alle Anweisungen auf eine Datenbank vollständig geschlossen sind. Ist dies nicht der Fall, werden alle Anweisungen bis zum Stand vor der Transaktion rückgängig gemacht. Das nennt man Rollback.

Die Sicherheit der Daten stellt Cloud Firestore für Mobil- und Webclient-Bibliotheken über die Firestore-Sicherheitsregeln her. Diese bieten sowohl Zugriffsverwaltung und -authentifizierung, jedoch könnte auch Daten hiermit für die Konsistenz der Datenbank validiert werden.

```

1  service cloud.firestore {
2      match /databases/{database}/documents {
3          match /cities/{city} {
4              allow read, write: if request.auth != null;
5          }
6      }
7  }
```

Listing 19: Beschränkung des Zugriffs auf Dokumente der Sammlung `cities`

Im Beispiel 19 wird der Lese- und Schreibzugriff auf ein Dokument der Sammlung `cities` beschränkt. Nur falls der anfragende Nutzer eine valide Authentifizierung besitzt, erhält er Zugriff auf das angefragte Dokument. Diese simple Darstellung ist jedoch für den wirklichen Produktionsseinsatz mit Vorsicht zu nutzen. Oftmals müssen `read` und `write` in detailliertere Vorgänge aufgeteilt werden. Ein `read` wird spezialisiert in `get` und `list`, wobei ein `write` in `create`, `update` und `delete` unterteilt werden kann. Ein `list` ermöglicht es hierbei auf Sammlungen, also die einzelnen Dokumenten IDs lesend zuzugreifen, jedoch nicht auf die Daten einzelner

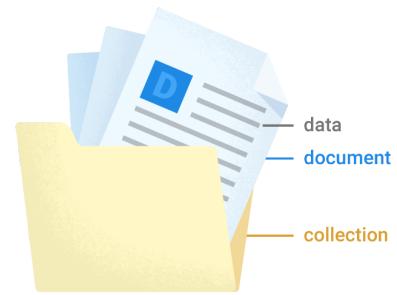


Abbildung 14: Datenmodell in Firebase<sup>6</sup>

<sup>6</sup>Quelle: [13]

Dokumente. Hierfür wird dann ein `get` benötigt. Mittels `create` erhält man Schreibzugriff auf nicht existierende Dokumente, durch `update` auf bereits vorhandene und `delete` ganze Dokumente erhält man über den `delete` Operator.

Sicherheitsregeln werden gleich dem Datenmodell hierarchisch aufgebaut und ermöglichen differenzierte Zugriffsbeschränkungen auf jeder Ebene. In Codebeispiel 20 beinhaltet jedes Dokument (Stadt) der Sammlung `cities` eine Unter-Sammlung `landmarks`. Nun lässt sich der Zugriff auf beide separat regeln. Bei der Sammlung `villages` hingegen wurde der rekursive Platzhalter verwendet. Hiermit sind Zugriffsregeln auf allen tieferen Ebenen gleich. Beim Verschachteln von `match` ist der innere Pfad immer relativ zum äußeren.

Wichtig zu wissen ist hierzu noch, dass falls mehrere `allow` Ausdrücke auf eine Anfrage zutreffen, wird der Zugriff erlaubt sobald **eine** Bedingung wahr, also erfüllt ist.

```

1  service cloud.firestore {
2      match /databases/{database}/documents {
3          match /cities/{city} {
4              allow read, write: if <condition>;
5
6              // Explicitly define rules for the 'landmarks'
7              // subcollection
8              match /landmarks/{landmark} {
9                  allow read, write: if <condition>;
10             }
11            match /villages/{document=**} {
12                allow read, write: if <condition>;
13            }
14        }
15    }

```

Listing 20: Hierarchische Zugriffsbeschränkung

Wie bereits oben besprochen können diese Regeln auch zur Validierung von Daten genutzt werden, damit die atomare Eigenschaft von Transaktionen bestehen bleibt. Hierzu kann die `getAfter()` Funktion genutzt werden. Mit dieser kann man auf Zustand eines Dokumentes zugreifen und diesen validieren, nachdem einer Folge von Anweisungen ausgeführt, jedoch diese noch nicht auf der Firestore Datenbank abgeschlossen wurde. Im Beispiel 21 existieren zwei Sammlungen: `cities` und `countries`. Jedes `country` Dokument beinhaltet das Feld `last_updated` um zu wissen, welche Stadt innerhalb eines Landes zuletzt aktualisiert wurde. Hierzu wird in den Sicherheitsregeln nach jedem Schreibzugriff auf ein `city` Dokument gleichzeitig auch das Feld des zugehörigen Landes aktualisiert.[13]

```

1  service cloud.firestore {
2      match /databases/{database}/documents {
3          // If you update a city doc, you must also
4          // update the related country's last_updated field.
5          match /cities/{city} {
6              allow write: if request.auth != null &&
7                  getAfter(
8                      /databases/${database}/documents/countries/$(request.
9                          resource.data.country)
9                  ).data.last_updated == request.time;

```

```

10    }
11
12    match /countries/{country} {
13        allow write: if request.auth != null;
14    }
15 }
16 }
```

Listing 21: Datenvlidierung für atomare Operationen

### 2.6.3 Cloud Storage

Um Filme, Videos oder andere Nutzer-generierte Inhalte abspeichern zu können, bietet Firebase Cloud Storage an. Durch das Firebase SDK für Cloud Storage können Dateien direkt von Client-Anwendungen hoch- bzw. heruntergeladen werden. Aufgrund von möglicher schlechter Verbindung kann mithilfe von robusten Operationen der Prozess des Hoch- bzw. Herunterladens bei besserer Verbindung an der Stelle weiter geladen werden, an welcher dieser unterbrochen wurde. Ähnlich wie bei Cloud Firestore in Kapitel 2.6.2 bestimmen auch hier Sicherheitsregeln den Zugriff auf bestimmte Dokumente.

Zusätzlich hierzu sind weitere Metadaten verfügbar: `contentType` und `size`. Mit ihnen lassen sich die Dateien beispielsweise validieren. Im Code 22 können Dateien nur hochgeladen werden, falls sie eine Größe kleiner 5 MB besitzen.

```

1  service firebase.storage {
2      match /b/{bucket}/o {
3          match /images/{imageId} {
4              allow write: if request.resource.size < 5 * 1024 * 1024
5                  && request.resource.contentType.matches('image/*');
6          }
7      }
```

Listing 22: Validierung nach Dateigröße

Außerdem lassen sich durch Cloud Functions aus dem nächsten Kapitel Prozesse automatisieren. Beispielsweise lässt sich beim Upload eines Bildes direkt ein individuelles Thumbnail erstellen lassen.[13]

### 2.6.4 Cloud Functions

Da Firebase - bis auf vereinfachte Sicherheitsregeln - eigentlich keinen Backend Code benötigt, jedoch manche Features eben genau diesen brauchen, um beispielsweise Benachrichtigungen an Nutzer zu senden oder Bilder zu komprimieren, existieren Cloud Functions.

Diese ermöglichen es, als Antwort auf ein Event automatisch oder durch HTTPS Anfrage manuell Backend Code auszuführen. Der gesamte Code ist hierbei in der Google Cloud gespeichert und wird in einer verwalteten Umgebung ausgeführt. Als Programmiersprache kann sowohl JavaScript als auch Typescript verwendet werden.

„Google Cloud Functions ist die serverlose Computerlösung von Google zum erStellen ereignisgesteuerter Anwendungen.“[13] Es kann sowohl auf der Google Cloud Platform (GCP) also auch für Firebase genutzt werden. Es ist bei beiden ein Verbindungsglied zwischen Logik und entsprechenden Diensten, welche dadurch mit serverseitigen Code erweitert und kombiniert werden. In Abbildung 15 ist ein typischer Anwendungsfall beschrieben. Ein Event auf der Datenbank

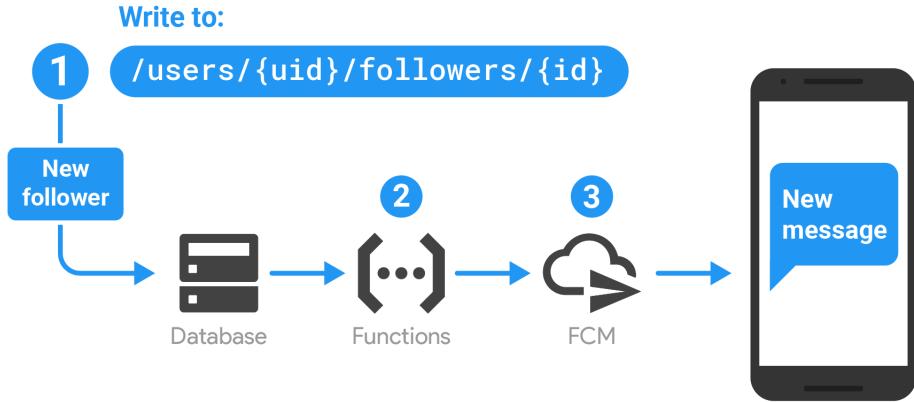


Abbildung 15: Cloud Functions Anwendungsfall Benachrichtigung

wird ausgelöst, hier ein neuer Nutzer folgt einem weiteren Nutzer. Es wird also ein Dokument in der Unter-Sammlung `followers` erzeugt. Diese Unter-Sammlung befindet sich innerhalb des Dokumentes `uid` der Sammlung `users`. Im zweiten Schritt erstellt die Funktion eine Nachricht, welche über Firebase Cloud Messaging (FCM) versendet werden soll. Über abgespeicherte Tokens sendet FCM die Benachrichtigung an das Gerät des Nutzers `uid`.[13]

## 2.6.5 Cloud Messaging

Firebase Cloud Messaging ist eine plattformübergreifende Messaging-Lösung zum zuverlässigen Versenden von Nachrichten an Nutzergeräte. In Abbildung 16 ist die Architektur dieses Tools dargestellt. Hierbei wird es grundlegend in das Erstellen, Transportieren und Empfangen der Nachrichten unterteilt.[13]

- **Erstellen** Die zu versendenden Nachrichten können, wie in Kapitel 2.6.4 beschrieben, manuell oder automatisiert erzeugt werden. Bei der Automatisierung ist wichtig, dass die Nachrichten in einer vertrauenswürdigen Serverumgebung erstellt werden, damit alle Nachrichtentypen unterstützt werden (Schritt 1). Das FCM Backend akzeptiert dann in Schritt 2 Nachrichtenanfragen, ordnet die Nachrichten verschiedenen Themen zu und erzeugt unter anderem Metadaten für Nachrichten, wie bspw. die Nachricht ID.
- **Transportieren** Die Nachrichten werden hierbei an die entsprechenden Geräte weitergeleitet. Da verschiedene Geräte auf unterschiedlichen Plattformen basieren, muss die Transportschicht auf Plattformebene arbeiten. Hierfür werden folgende Ebenen genutzt:
  - Android Transport Layer (ATL) für Android-Geräte mit Google Play-Diensten
  - Apple Push Notification Service (APNs) für iOS-Geräte
  - Web-Push-Protokoll für Web-Apps
- **Empfangen** Das FCM SDK behandelt die Benachrichtigung oder Nachricht. Dies ist abhängig vom Vorder-/ Hintergrundstatus der Anwendung und der jeweiligen Anwendungslogik.

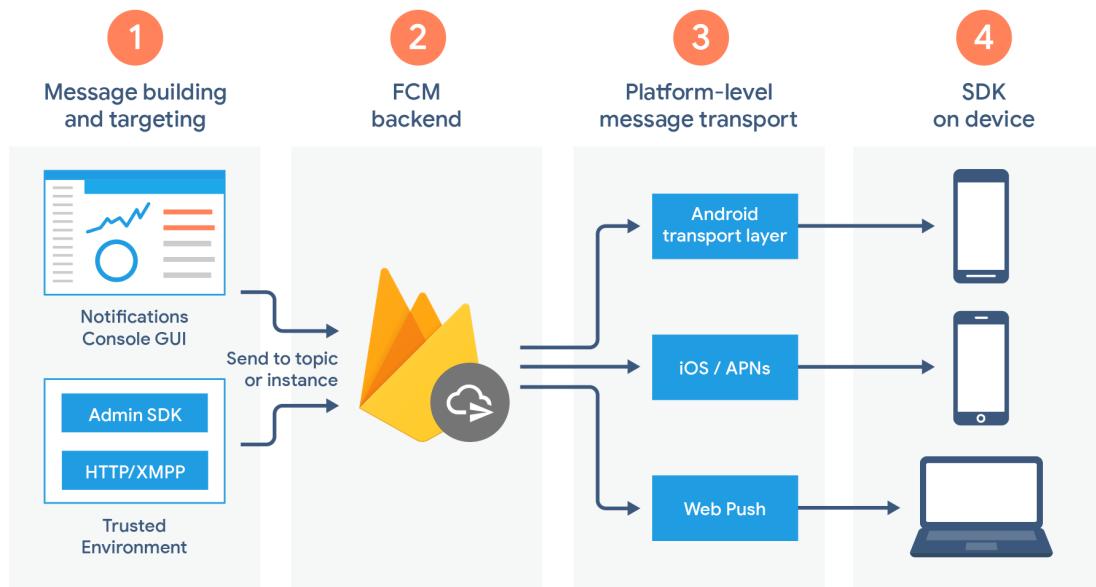


Abbildung 16: Firebase Cloud Messaging Architektur



Abbildung 17: Google AdMob Anzeigemöglichkeiten

## 2.6.6 Google AdMob

Google AdMob bietet eine einfache Art, gezielte Werbung innerhalb der Anwendung zu schalten und somit die Anwendung zu monetarisieren. Zusätzlich bietet das Tool in Kombination mit Google Analytics<sup>7</sup> zusätzliche Anwendungsdaten und Analysefähigkeiten.

Werbung lässt sich in unterschiedlicher Weise anzeigen (siehe Abbildung 17) und lässt sich reibungslos in UI Komponenten integrieren. Verschiedene Features sind hier jedoch plattformabhängig. Auf der Android Plattform ist es für Nutzer möglich, beworbene Produkte direkt aus der Anwendung heraus zu kaufen.

Ein weiteres Werbetooll *Google Mobile Ads SDK* ist eine alleinstehende SDK, hingegen Google AdMob bietet einfache Integration in Firebase und weitere Tools.[13]

<sup>7</sup>Ein freies Analysetool, welches über alle Tools hinweg Ereignisse sammelt und diese Werte direkt graphisch darstellt. Da es für die Implementierung nicht weiter relevant ist, wird es nicht detaillierter besprochen. Zusätzliche Informationen unter [firebase.google.com](https://firebase.google.com).

## 2.7 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%).<sup>8</sup> Laut Cisco wird dieser Markt sich jedoch nicht nur auf Industrieländer beruhen, sondern bis 2023 sollen weltweit 71% der Bevölkerung mobile Konnektivität haben. [8] 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-) ineffizienten Entwicklungsprozessen auszuruhen. Dabei bieten jährliche, wenn nicht sogar halbjährliche Design- und Performanceänderungen von den Geräten selbst oder der Betriebssysteme Herausforderungen an die mobilen Anwendungen - *apps* - 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 Anwendungen.

### 2.7.1 Begriffe

**Eine Plattform** besteht aus der Hardware (System und zusätzlicher Peripherie, wie Sensoren oder Aktoren), dem Betriebssystem, den spezifischen *Software Development Kits (SDK)* und den jeweiligen Basisbibliotheken. Zusammen bietet eine Plattform die Grundlage um Software für sie zu entwickeln.

**Ein Framework** definiert eine Architektur für Anwendungen und stellt Komponenten bereit, mit welchen das Entwickeln einer Anwendung erleichtert sein soll. [6] Ein plattformübergreifendes Framework muss somit Anwendungscode für mehrere Plattformen wiederverwenden, jedoch müssen auch plattformspezifische Funktionen, wie Architektur oder Benutzeroberflächen API, bereitgestellt werden. Mehr dazu in Kapitel 2.7.3.

**Eine mobile Anwendung** ist eine Anwendung, geschrieben für eine Plattform eines mobilen Endgerätes, welche die jeweiligen Features nutzen könnte - dazu zählen Kamera(s), Beschleunigungssensoren oder auch *Global Positioning System (GPS)*. Webseiten als solches sind demnach keine mobilen Anwendungen.

### 2.7.2 Plattformspezifische native Apps

Plattformspezifische oder auch native Anwendungen sind Programme, welche auf eine gewisse Plattform abzielen und in einer der davon unterstützten Programmiersprachen geschrieben wurden. Da diese Art der (mobilen) Anwendung mit plattformspezifischen 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 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

<sup>8</sup><https://www.emarketer.com/content/us-time-spent-with-mobile-2019>, zuletzt aufgerufen: 26.02.2021

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].

### 2.7.3 Plattformübergreifende Anwendungen

Die Entwicklung einer plattformübergreifenden Anwendung zeichnet sich generell durch die Möglichkeit aus, nur einmal Code schreiben zu müssen, diesen jedoch auf mehreren Plattformen ausführen zu können.

Verschiedene Ansätze einer solchen Anwendung sind in Abbildung 18 kategorisiert. Im Folgenden werden jene Entwicklungsmöglichkeiten detaillierter besprochen.

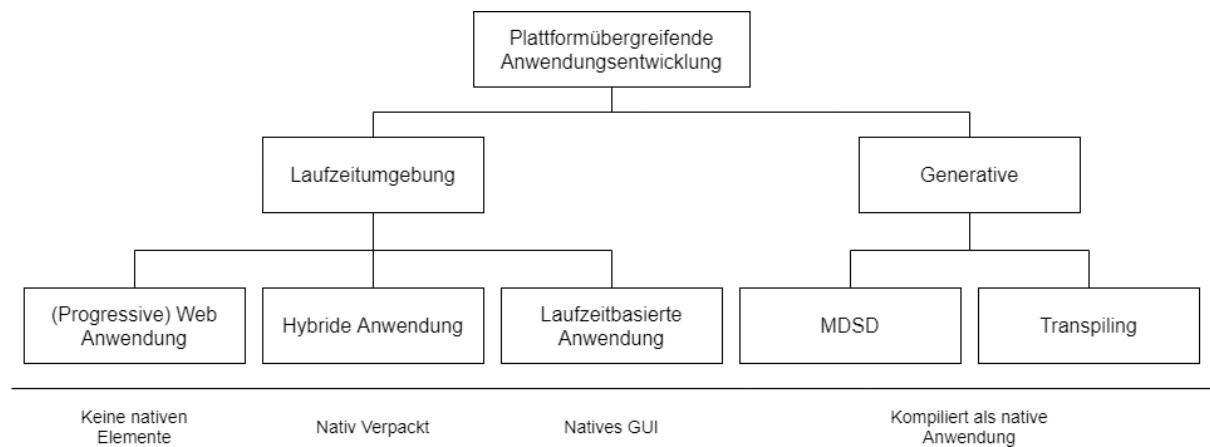


Abbildung 18: Kategorisierung verschiedener plattformübergreifender Ansätze. Erstellt nach [7]

#### 2.7.3.1 (*Progressive*) Web Apps

Eine mobile Webanwendung ist eigentlich eine Webseite, welche sich an die Größe und Auflösung von unterschiedlichen Bildschirmen anpasst - hier speziell an die Bildschirmgrößen der mobilen Geräte. Diese Anwendung ist mit Standard Webentwicklungstools geschrieben (HTML, CSS & JavaScript) und läuft somit theoretisch auf jedem Gerät mit einem Internet Browser [9]. Aufgrund der steigenden Unterstützung von jeglichen APIs in mobilen Browsern, ist es auch möglich geworden auf Geräteeigenschaften, wie bspw. den Standort zuzugreifen.

Jedoch kann diese App logischerweise nicht im jeweiligen *Appstore* heruntergeladen werden, da es sich weiterhin um eine Webseite handelt. Aus gleichem Grund kann hiermit auch kein „natives Design und Leistung“ erzeugt werden.

Abhilfe hierfür sorgt jedoch die von Google vorgestellte Design Idee *Progressive Web Apps* (PWA). Sie bietet die Möglichkeit Code in sogenannte *service worker* als Hintergrundthread ausführen zu lassen, ein Webseiten Manifest anzugeben, die App offline bedienen zu können und bieten die Möglichkeit die PWA zu installieren. Gleichzeitig kann mit diesem Design eine zu

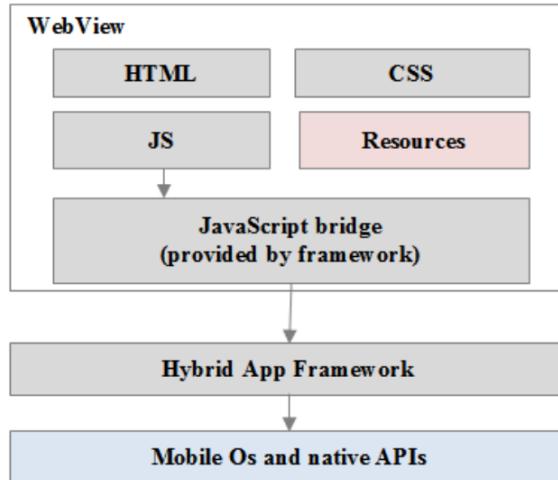


Abbildung 19: Struktur einer hybriden Anwendung<sup>9</sup>

nativen Apps vergleichbare Leistung erreicht werden [11].

Generell ist der Ansatz sehr simpel, da hiermit plattformübergreifende Anwendungen geschrieben werden können, welche sich auf allen Geräten mit Browser bedienen lassen. Hierfür wird zudem keine zusätzliche Programmiersprache oder wissen über die jeweilige Plattform benötigt. Eine große Schwierigkeit hieran ist weiterhin der Zugriff auf Gerätefeatures, da nicht alle über den Browser verfügbar sind.

### 2.7.3.2 Hybride Anwendungen

Eine hybride Anwendung kombiniert die native Vorgehensweise mit der einer normalen Webseite. In einer nativen WebView ist eine Webanwendung verpackt, welche nun in einer *HTML-Rendering-Engine* gerendert wird. Bei Android und iOS ist das WebKit. Diese WebView funktioniert ähnlich wie ein normaler Browser, jedoch werden Kontrollfenster nicht angezeigt, wie zum Beispiel Adresszeile, Einstellungen oder Lesezeichen. Ähnlich wie bei Web Anwendungen werden über JavaScript APIs Gerätefeatures eingebunden.

Ein sehr frühes Framework für diese Art von Anwendung war Adobe Cordova, eher bekannt als das ursprüngliche PhoneGap von Nitobi. Viele weitere Frameworks basieren auf ihren Anfängen.

Eine hybride Anwendung kann also normal als App im *Appstore* heruntergeladen auf dem Gerät installiert und offline genutzt werden - also sehr ähnlich zu nativen Lösungen. Daher ist dieser Ansatz auch sehr beliebt. Daher liegt auch hier die Leitung der Applikation deutlich hinter der der Nativen [10] [11].

### 2.7.3.3 Runtime basierte Anwendungen

Im Gegensatz zur in Kapitel 2.7.3.2 beschriebenen hybriden Anwendung, nutzen Runtime basierte Anwendungen keinen Browser des Gerätes mit einer WebView, sondern jede App besitzt eine eigene Runtime Ebene. Jedes Framework muss also eine solche Ebene für alle Plattformen in jeweiliger Programmiersprache mitliefern, damit seine Anwendung hierauf laufen können. Die Anwendungen hingegen sind dann beispielsweise in JavaScript (bspw. React Native oder NativeScript), C# (Xamarin) oder sonstigen Programmiersprachen (bspw. Qt) geschrieben.

<sup>9</sup>Quelle: [10]

Jedem Framework-Entwickler ist die Freiheit gegeben, wie man die Anbindung an native Funktionen regelt. Bei hybriden Anwendungen ist dies durch die WebView Cordova festgelegt. Typisch für Anwendungen dieser Art jedoch ist ein Plug-In-basiertes *bridging System*. Es ermöglicht den Aufruf von fremden Funktionsinterfaces in plattformspezifischem Code. Somit können beispielsweise React Native und NativeScript mit Sprachinterpretoren (bspw. JavaScriptCore und V8) auf den Geräten Auszeichnungssprache (hier HTML (Hypertext Markup Language)) interpretieren und plattformspezifische Komponenten der Benutzeroberflächen erzeugen.

Ein großer Nachteil dieser Strategie ist jedoch zugleich ihr Vorteil: Jedes Framework besitzt seine eigene Architektur. Dadurch sind Plug-ins des einen Frameworks trotz gleicher Anwendungssprache nicht unbedingt funktionstüchtig im anderen. Bei projektspezifischen Plug-ins macht es einen späteren Systemwechsel daher besonders schwer, da nicht nur Benutzeroberfläche und Businesslogik neu geschrieben werden müssen, sondern auch jeweilige Plug-ins [11].

#### 2.7.3.4 Model-driven Software Entwicklung

Die Grundsätze der modellgetriebenen Softwareentwicklung beschäftigen sich mit der Abstraktion des Modells als (Teil eines) System, von welchem die eigentliche Software abgeleitet wird [12].

Das bedeutet in der Realität, dass eine höhere Abstraktion als Quellcode in Form von textuellen oder grafischen domänenspezifischen Sprachen oder universell einsetzbaren Modellierungssprachen (Unified Modeling Language(UML)) zum beschreiben der Software verwendet wird. Codegeneratoren übersetzen diese Modelle nun jeweils in Programmiersprachen der gewählten Zielplattform, auf welcher sie kompiliert werden.

Theoretisch kann hierdurch der komplette Funktionsumfang wie bei einer nativen Anwendung erreicht werden. Bekannte Frameworks dieser Methode sind zum Beispiel MD<sub>2</sub>, MAML, WebRatio Mobile, BiznessApps und Bubble.

Der große Nachteil hieran ist, dass Entwickler sehr selten modellgetriebene Entwicklung verwenden, sondern Quellcode-basierte Programmiermethoden bevorzugen [11].

#### 2.7.3.5 Kompilierte Anwendungen

Kompilierte plattformübergreifende Anwendungen basieren auf einer einzigen Codebasis und können für mehrere Plattformen vollständig kompiliert werden. Dies kann entweder von der Codebasis einer nativen Anwendung für mindestens eine andere Plattform (bspw. J2ObjC), oder von einer unabhängigen Codebasis direkt für mehrere Plattformen (bspw. Flutter) geschehen. Hierbei ist Flutter für diesen Anwendungsfall am interessantesten und wird in Kapitel 2.8.2 näher behandelt.

Ein Hindernis dieser Art ist die erhöhte Komplexität der einzelnen Frameworks.

### 2.8 Frameworks zur mobilen, plattformübergreifenden Entwicklung

In den folgenden Kapiteln werden einzelne Frameworks zur mobilen, plattformübergreifenden Entwicklung vorgestellt.

In der Statistik 20 aus dem Jahr 2020 ist React-Native das beliebteste Framework, dicht gefolgt von Flutter. Wie man deutlich hier auch sehen kann, haben andere, bisher auch sehr erfolgreiche Frameworks einen enormen Rückgang von teilweise über einem Drittel ihrer Nutzer erleben müssen.

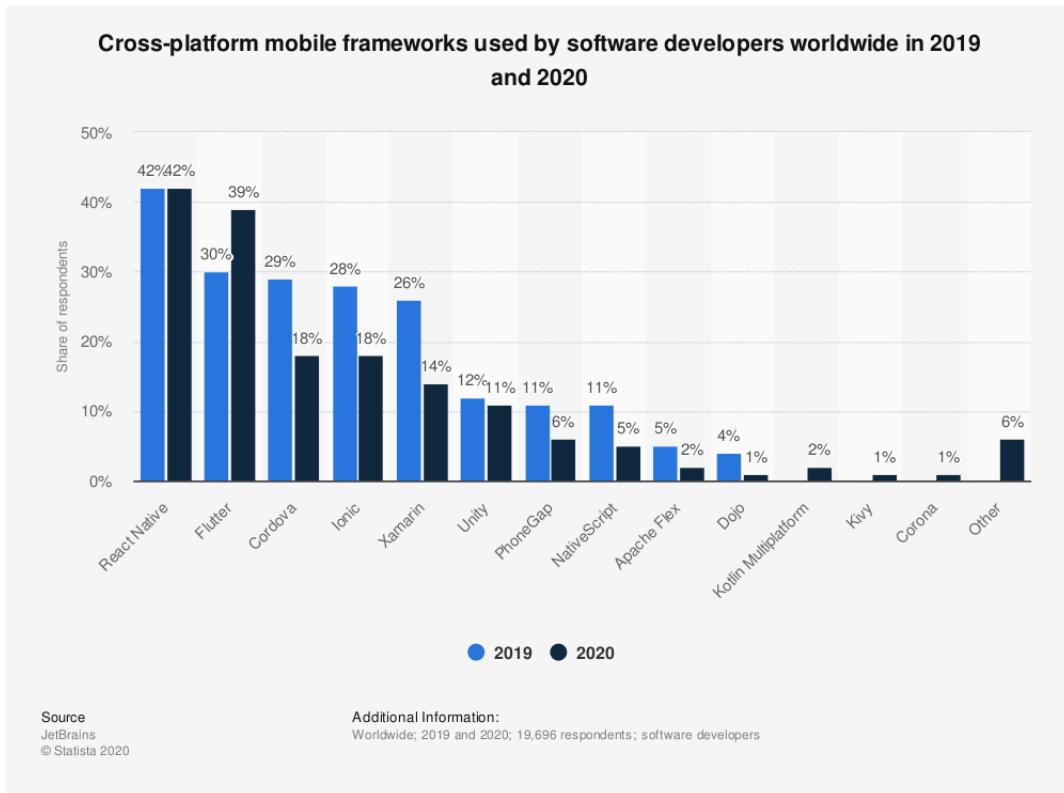


Abbildung 20: Beliebtheit nach Framework im Bereich der plattformübergreifenden mobilen Entwicklung<sup>10</sup>

Aus diesen Gründen werden im weiteren Verlauf nur die Frameworks React-Native und Flutter weiter besprochen.

### 2.8.1 React Native

React Native ist ein open source Framework, welches von Facebook 2015 veröffentlicht wurde. Es basiert auf dem bekannten Web-Framework *React* (ebenfalls von Facebook) und bringt daher den deklarativen und Komponenten-basierten Stil mit sich. Die Programmiersprache ist aus diesem Hintergrund auch logischerweise JavaScript. Das Framework an sich ist in verschiedenen Sprachen implementiert: JavaScript, Swift, Objective-C, C++ und Python.

Allgemein bietet das Framework die Möglichkeit plattformübergreifende Apps für iOS, Android und für Windows zu schreiben. Hierbei wird der geschriebene Code in einer JavaScript Laufzeitumgebung ausgeführt (React Native selbst verwendet generell JSC (JavaScriptCore), seit neuestem kommt auch Hermes zum Einsatz, jedoch sind auch andere bekannte Umgebungen denkbar - bspw. V8 in Chrome) es lässt sich zu den Runtime-basierten Anwendungen in Kapitel 2.7.3.3 zuordnen [3].

<sup>10</sup>Quelle: [www.statista.com](http://www.statista.com), zuletzt aufgerufen am 22.04.2021

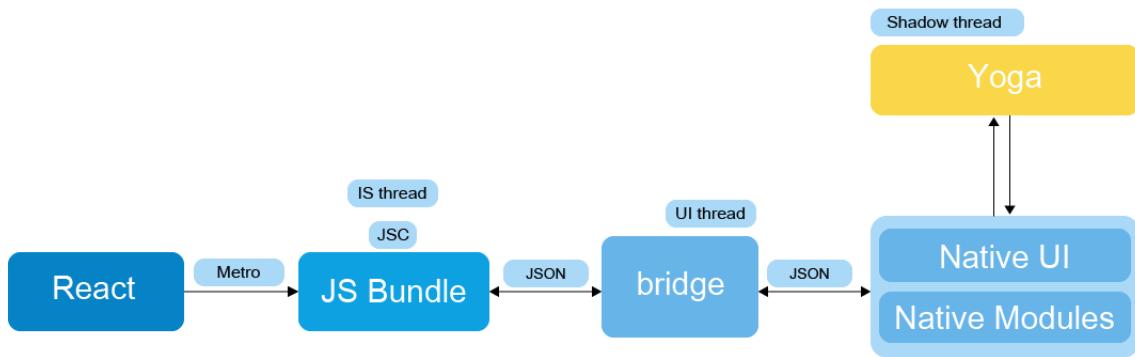


Abbildung 21: Alte Architektur von React Native<sup>11</sup>

### 2.8.1.1 Architektur

Grundlegend wurde React Native als Plattform-agnostisch designet. Entwickler schreiben also plattformunabhängigen JavaScript React Code, während das Framework den erstellten React Baum in Plattform-spezifischen Code umschreibt. Hierbei wurde 2013 (noch intern) die Web-Technologie React mit nativen Plattformen (nur interne) vereint, jedoch war dieses Design aufgrund eines einzigen Threads sehr langsam. Um dies zu verbessern basierte das Framework lange Zeit auf drei unterschiedlichen Threads, welche über eine Brücke verbunden sind.

- *JavaScript Thread*. Hier wird der gesamte JavaScript Code abgelegt und interpretiert. Alles wird über die JSC Engine ausgeführt.
  - *Native Thread*. Die Benutzeroberfläche und Kommunikation mit dem JavaScript Thread steht hier im Mittelpunkt. Der gesamte native Code wird hier ausgeführt. Die Benutzeroberfläche wird dann aktualisiert, sobald die eben ein Änderung vom JS Thread vermittelt wird.
  - *Shadow Thread*. Hier wird das gesamte Layout der Anwendung berechnet. Zugrunde liegt die Facebook-eigene layout engine „Yoga“.

Ein Hauptproblem dieses Ansatzes ist, dass die Brücke grundlegend eine asynchrone Warteschlange ist, da der JS Thread und der native Thread unabhängig voneinander arbeiten. Zusätzlich werden während der gesamten Datenübertragung die Daten im JSON Format serialisiert und deserialisiert. Daher kann es zu Performance-Einbrüchen und somit zu schlechter Nutzererfahrung, durch bspw. Eingabeverzögerung, kommen.

Nach der Ankündigung 2018 veröffentlichte Facebook im Juli 2020 die neue Architektur. Mit ihr wurde der Bottleneck (die Brücke) ersetzt durch das JavaScript Interface. Es ermöglicht nicht nur die komplette Synchronisierung der beiden Threads, sondern auch die direkte Kommunikation untereinander - vor allem das Konzept von „shared ownership“ ist hier tragend, weshalb auch keine Serialisierung mehr nötig ist. Zudem ist man nun nicht mehr an JSC gebunden, sondern kann auch jegliche hoch-performante JavaScript Engines als Laufzeitumgebung verwenden. Native Module werden nun nur noch bei Bedarf geladen anstatt alle beim Start der App.

---

<sup>11</sup>Quelle: React Native Re-Architecture, zuletzt aufgerufen am 15.04.2021

<sup>12</sup>Quelle: React Native Re-Architecture, zuletzt aufgerufen am 15.04.2021

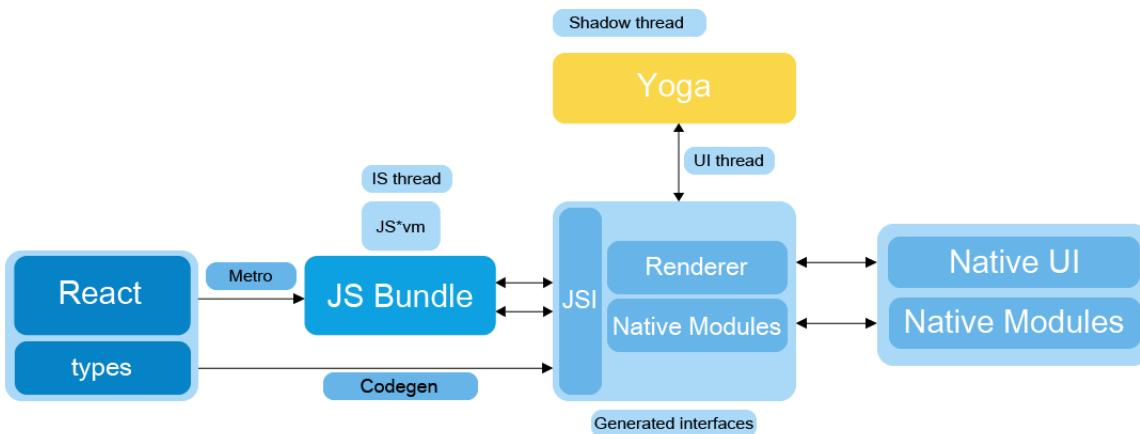


Abbildung 22: Neue Architektur von React Native<sup>12</sup>

Weiter wurde veralteter Legacy-Code aus dem Kern von React Native entfernt und nicht-essenzielle Teile aus dem Kern ausgelagert. Die aktuelle Architektur von React Native ist in Abbildung 22 zu sehen.<sup>13</sup>

### 2.8.1.2 JSX mit nativen Komponenten

React (Native) verwendet als Programmiersprache JSX. Diese ist eine syntaktische Erweiterung von JavaScript (**JavaScript eXtention**), welche zur fundamentalen Beschreibung der Nutzeroberfläche dient. JSX wird in normale JavaScript Objekte kompiliert, weshalb es nicht zwingend ist.

```
1 const element = <h1>Hello, world!</h1>;
```

Listing 23: JSX Hello World Element

Mithilfe dieser losen Kopplung von UI-Code und dazugehöriger Logik schlägt React eine optionale Lösung zur *Separation of Concerns* vor. Anstelle dessen ist es auch möglich die Technologien in Markup- und Logik-Dateien aufzuteilen.

Außerdem kann argumentiert werden, dass JSX nur eine weitere Template-Sprache sei, ähnlich HTML oder XAML. Dies ist nicht korrekt, da wie oben bereits erwähnt JSX lediglich eine syntaktische Erweiterung von JavaScript ist, also inmitten von JSX Objekten JavaScript geschrieben werden kann.

Weiterhin ist interessant, dass JSX Cross Site Scripting vorbeugt, indem der React DOM alle eingesetzten Werte zunächst in einen einfachen String konvertiert [4].

```
1 import React from 'react';
2 import { Text } from 'react-native';
3
4 const Cat = () => {
5   return (
6     // <Text> as native component
```

<sup>13</sup>Quelle: React Native's re-architecture (2020): <https://medium.com/swlh/react-natives-re-architecture-in-2020-9bb82659792c>

```

7   <Text>Hello, I am your cat!</Text>
8 );
9 }
10
11 export default Cat;

```

Listing 24: Native Komponenten

In dem Codebeispiel 24 wird ein Element `Cat` erzeugt, welches als Beschreibung dessen dient, was letztendlich auf dem Bildschirm angezeigt wird. In diesem einfachen Beispiel wird die native Komponente `<Text>...</Text>` verwendet. Native Komponenten sind in nativem Code (Kotlin oder Java für Android, bzw. Swift oder Objective-C für iOS) implementierte Komponenten und können in JavaScript Code aufgerufen werden. Diese werden dann während der Laufzeit für die jeweilige Plattform erstellt.

React Native bringt die wichtigsten Komponenten mit sich, die **Core Components**. Zusätzlich erlaubt das Framework jedoch auch eigene Komponenten nativ zu implementieren, welche dem jeweiligen Anwendungsfall angepasst werden können.

### 2.8.1.3 Komponenten

Gleichzeitig erlaubt React Native aber auch wiederverwendbare Komponenten in JavaScript aus den Kernkomponenten zusammenzustellen. Hierzu lassen sich einzelne Komponente ineinander verschachteln, um hier im Beispiel 25 einen `Text` innerhalb einer `View`<sup>14</sup> anzeigen zu lassen. Zusätzlich ist es möglich sogenannte „props“ also Eigenschaften (engl.: properties), ähnlich einer normalen Funktion mitzugeben. In dem betrachteten Beispiel entspricht das den Namen einzelner Katzen, welche als Text angezeigt werden [3].

```

1 import React from 'react';
2 import { Text, View } from 'react-native';
3
4 // configurable props
5 const Cat = (props) => {
6   return (
7     <View>
8       <Text>Hello, I am {props.name}!</Text>
9     </View>
10   );
11 }
12
13 const Cafe = () => {
14   return (
15     <View>
16       // reusable
17       <Cat name="Maru" />
18       <Cat name="Jellylorum" />
19       <Cat name="Spot" />
20     </View>

```

<sup>14</sup>Eine `View` ist die Basiskomponente einer Benutzeroberfläche. In einer `View` wiederum können weitere Views verschachtelt sein.

```

21   );
22 }
23
24 export default Cafe;

```

Listing 25: Eigene Komponenten

#### 2.8.1.4 State

Für eine interaktive Benutzeroberfläche fehlt jedoch noch das Kernprinzip eines deklarativen UI. Ein sogenannter *State* wird verwendet um die Daten, welche sich mit der Zeit oder über Nutzerinteraktion ändern, auf der Oberfläche anzuzeigen. Dieses Konzept wird ebenfalls von dem zweiten Framework verwendet und ist in Abbildung 26 visualisiert.

Seit v14.8 ermöglicht React (und auch React Native) durch Hooks einer Funktion einen *State* hinzuzufügen. Hooks sind Funktionen, welche es Entwicklern ermöglichen sich in React Features einzuhaken. Bisher wurde dies über Klassenkomponenten ermöglicht, wodurch es jedoch komplizierter ist *stateful logic* zwischen Komponenten wiederzuverwenden. Daher entkoppelt man diese Logik von den Komponenten und erlaubt unter anderem das separate Testen.

```

1  import React, { useState } from "react";
2  import { Button, Text, View } from "react-native";
3
4  const Cat = (props) => {
5    // make isHungry stateful
6    const [isHungry, setIsHungry] = useState(true);
7
8    return (
9      <View>
10     <Text>
11       // show text dependent on isHungry state
12       I am {props.name}, and I am {isHungry ? "hungry" : "full"}!
13     </Text>
14     <Button
15       onPress={() => {
16         setIsHungry(false);
17       }}
18       disabled={!isHungry}
19       title={isHungry ? "Pour me some milk, please!" : "Thank you!"}
20     />
21     </View>
22   );
23 }
24
25 const Cafe = () => {
26   return (
27     <>
28     <Cat name="Munkustrap" />
29     <Cat name="Spot" />
30     </>
31   );

```

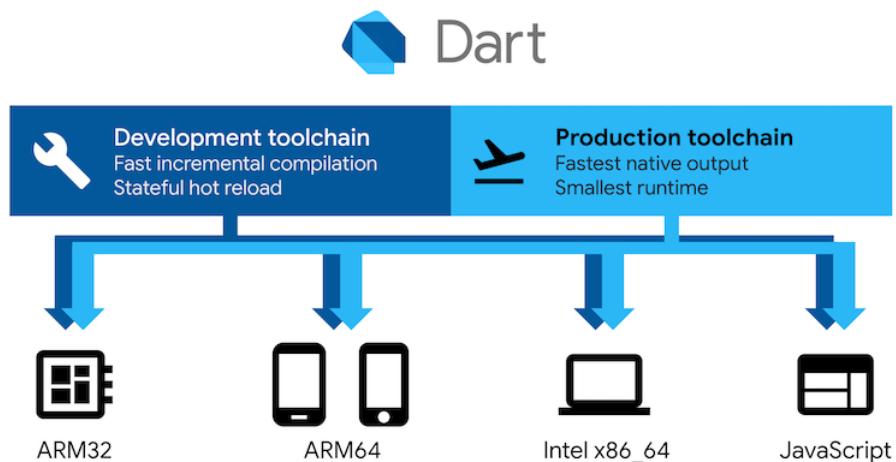


Abbildung 23: Kompatibilität der Dart Plattform <sup>16</sup>

```

32 }
33
34 export default Cafe;

```

Listing 26: State mit useState Hook

Im Beispiel 26 wird in Zeile 6 der `useState`-Hook verwendet. Diese Funktion erzeugt eine State-Variablen mit dem Initialwert `true` und erstellt gleichzeitig eine Funktion zur Änderung des States (`setIsHungry`). Daraufhin wird, abhängig ob die Katze hungrig ist, dies im Text angezeigt, der Knopf zum füttern (de-)aktiviert bzw. auch hier den Text verändert. [3]

## 2.8.2 Flutter

Flutter ist eine open-source SDK entwickelt von Google und ist geschrieben in C, C++ und Dart. Auf Basis eines Codes erlaubt Flutter es gleichzeitig Anwendungen für Android, iOS, Web und Desktop zu erstellen und ist zudem die primäre Methode für Google Fuchsia, Googles Betriebssystem<sup>15</sup>. Als 2D Grafikbibliothek verwendet Flutter Skia, welche auch von Chrome, Firefox und Android verwendet wird. Zudem basiert Flutter auf der Dart-Plattform welche das Kompilieren auf 32-bit und 64-bit ARM Prozessoren, auf Intel x64 Prozessoren und in JavaScript ermöglicht (siehe Abbildung 23). Daher ist Flutter eine wie in Kap. 2.7.3.5 beschriebene kompilierte, plattformübergreifende Anwendung.

Während der Entwicklung werden Flutter-Apps in einer virtuellen Maschine (VM) gestartet, welche *stateful hot reload* ermöglicht. Bei Änderungen muss die App so nicht komplett neu kompiliert werden, sondern sie erneuert sich während der Laufzeit. Bei der Veröffentlichung der App, wird sie in den Maschinencode der beschriebenen Plattformen übersetzt.

### 2.8.2.1 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

<sup>15</sup>Quelle: <https://fuchsia.dev/>

<sup>16</sup>Quelle: <https://github.com/flutter/flutter>

the layer below, and every part of the framework level is designed to be optional and replaceable<sup>17</sup>.

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 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 [5].

### 2.8.2.2 Dart

Während der Entwicklung von Flutter standen sicherlich mehrere Sprachen zur Auswahl: Wie in Kapitel 2.7.3 gelernt, gibt es viele unterschiedliche Ansätze mit beispielsweise webbasierten Sprachen wie JavaScript, mit nativen Sprachen wie Java oder Swift, oder auch mit anderen objektorientierten Sprachen wie C#. Wieso wurde also genau Dart als Programmiersprache und Runtime ausgewählt?

Die Programmiersprache Dart ist in allgemeinen Zügen der Sprache C ähnlich, sodass vielen Entwicklern das Lesen erleichtert wird. Es ist eine objektorientierte Sprache und besitzt einen Garbage Collector.

Dart ist designet als eine client-fokussierte Sprache, welche gleichermaßen Entwicklung (sub-second stateful hot reload) und Produktion in allen möglichen Zielplattformen (Web, Mobile und Desktop) priorisiert. Dadurch erhält man eine effizienzoptimierte Entwicklungsphase, sowie ebenfalls die Möglichkeit eine Code-Basis in unterschiedliche Plattformen zu kompilieren (siehe Abbildung 23).

Sie bietet zudem auch *sound-null-safety*, was bedeutet dass Werte nicht null sein können, außer sie werden so festgelegt. Hiermit werden Null-Exceptions während der Laufzeit durch statische Codeanalyse vorgebeugt.

---

<sup>17</sup><https://flutter.dev/docs/resources/architectural-overview>

<sup>18</sup>Quelle: <https://github.com/flutter/flutter>

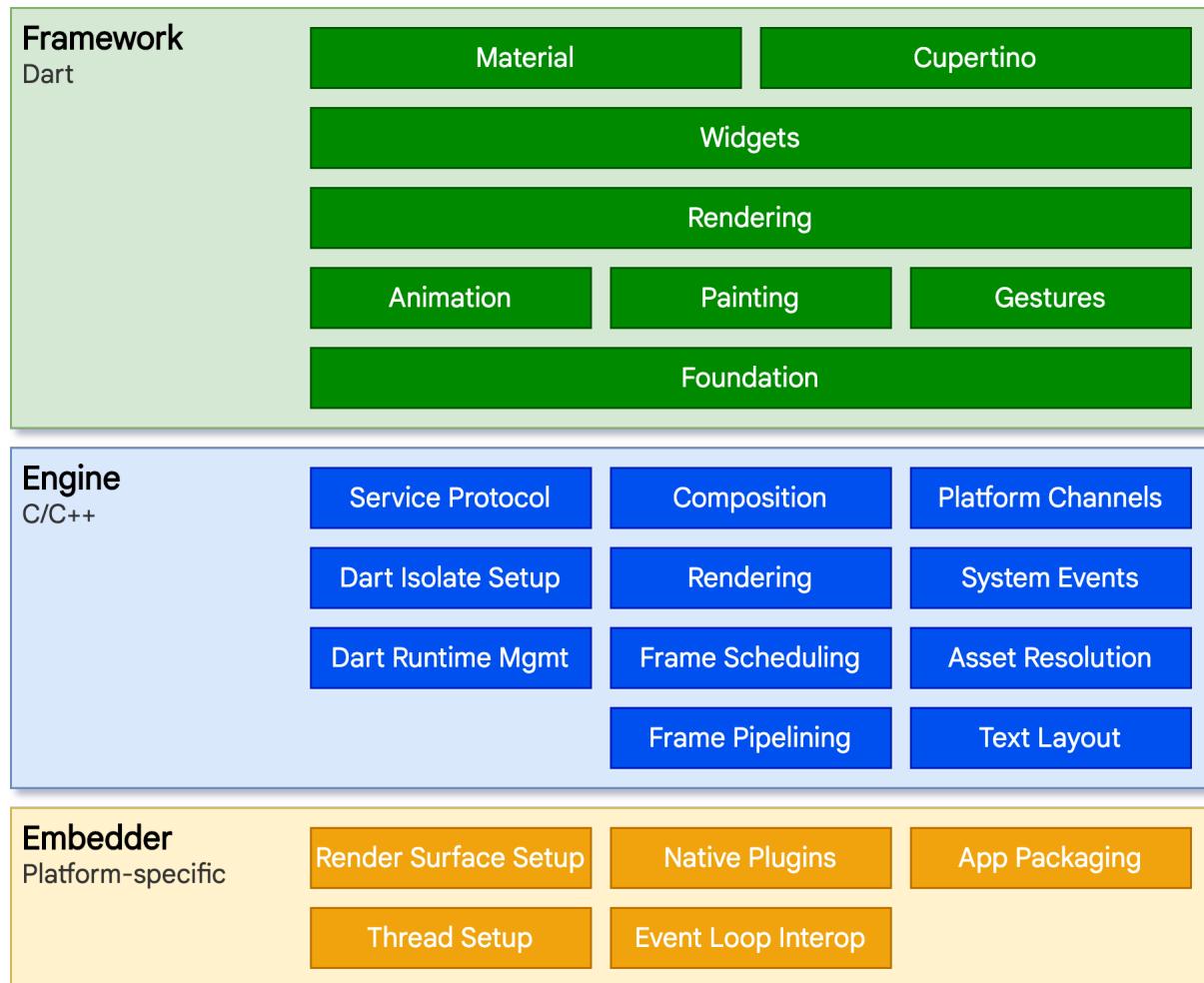


Abbildung 24: Bibliotheken und Ebenen der Flutter Plattform <sup>18</sup>

### 2.8.2.3 Widgets

Wie bereits beschrieben sind Widgets wiederverwendbare Kompositionsbauusteine, mit welchen Benutzeroberflächen in Flutter zusammengebaut werden. Jedes einzelne ist ein *immutable declaration* eines Teils der Benutzeroberflächen - also ein konstanter Bestandteil.

Flutter arbeitet mit der Devise:

***Everything is a widget.***

Auf diesem Satz baut die Einfachheit von Flutter auf. Jedes Objekt, jede Animation, jede Reihe, einfach alles ist ein Widget. Somit baut man eine App von der Wurzel aus auf und beschreibt die einzelnen Abzweigungen exakt. Die Anordnung von Widgets ist daher hierarchisch aufgebaut. Ein Widget wird also immer in einem Elternteil verschachtelt sein und erhält bei seiner Erstellung den *build context* übergeben. Das „äußerste“ Widget, also die Wurzel, enthält somit die gesamte App. Typischerweise ist das ein *MaterialApp* oder *CupertinoApp* Widget.

OEM Widgets, also Widgets von und für eine spezifische Plattform werden von Flutter gemieden. Hierfür erzeugt Flutter eigene Widgets mithilfe der oben genannten, eigener Rendering Plattform. Da diese Widgets jedoch komplett individualisierbar sind, bietet man somit native

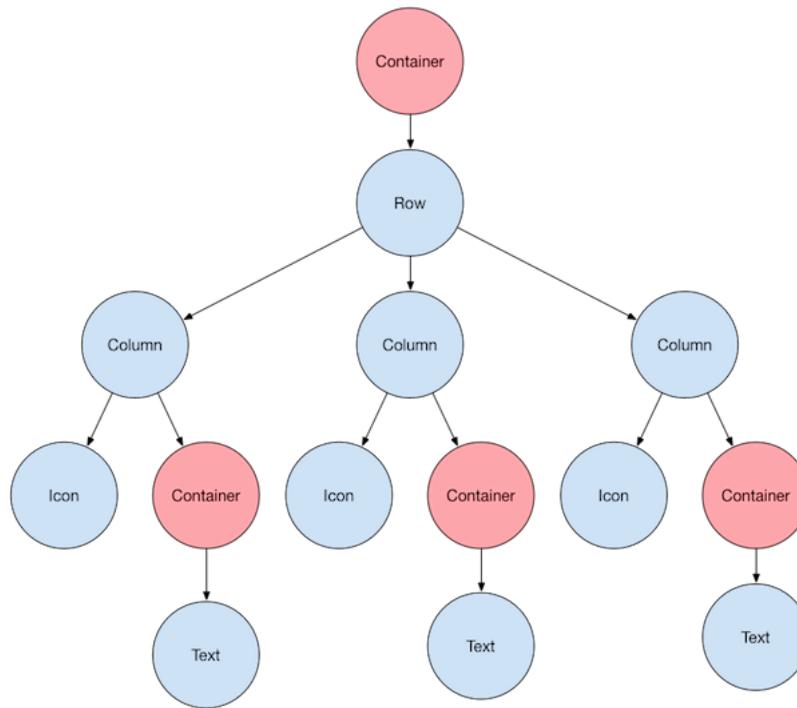


Abbildung 25: Widget Baum einer beispielhaften Anwendung<sup>19</sup>

Möglichkeiten für jegliche Stile. Es gibt auch Pakete, welche Plattform-ähnliche Widgets zur Verfügung stellen.

#### 2.8.2.4 States

Flutter ist deklarativ - das bedeutet, die Benutzeroberfläche wird anhand von dem aktuellen *State* der App erzeugt. Im Gegensatz dazu muss der Entwickler beim imperativen Stil die Übergänge der einzelnen *States*. Hier wird dies durch das Framework gelöst.



Abbildung 26: Deklarative Benutzeroberfläche<sup>20</sup>

<sup>19</sup>Quelle: <https://flutter.dev/docs/development/ui/layout>

<sup>20</sup>Quelle: <https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative>

Tabelle 7: Nutzer-Item Matrix mit Bewertungen. Jede Zelle  $r_{u;i}$  steht hierbei für die Bewertung des Nutzers  $u$  an der Stelle  $i$

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

## 2.9 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 *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 7 als Matrix dargestellt werden.

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

#### 2.9.1.1 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.

### 2.9.1.2 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.

### 2.9.2 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 "verfängt" quasi in eine Richtung.[1]

### 2.9.3 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.

## 2.9.4 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  $s : E \times E \rightarrow \mathbb{R}$  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

### 2.9.4.1 Cosinus-Funktion

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

### 2.9.4.2 Skalarprodukt

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

### 2.9.4.3 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>

## 3 Planung

### 3.1 Konzept

Aus Perspektive der Benutzer ist die Idee hinter StreamSwipe trivial. Dem User werden nacheinander Filme und Serien vorgeschlagen, die er bewerten kann. Auf Basis seiner Präferenzen erhält er Matches, mit denen er über sich eine Chatfunktionen austauschen kann. Jeder dieser Schritte muss möglichst schnell und unkompliziert erfolgen.

Aus Sicht des Anbieters ist die Realisierung wesentlich komplexer als es dem User erscheint, da eine Zusammenstellung an Komponenten benötigt wird. Diese besteht aus einer Smartphone-App, einer Userdatenbank, einer Datenbank mit den Filminformationen und einem Server, der das Matching ausführt. Zusätzlich wird eine mögliche Komponente für den Messenger benötigt. Die Auswahl dieser Komponente wird jeweils über individuell gestellte Anforderungen getroffen.

Bei der Smartphone-App muss bereits während der Entwicklung auf Benutzerfreundlichkeit und Barrierefreiheit geachtet werden. Sie sollte intuitiv bedienbar, attraktiv design und ressourcensparend für leistungsschwache Smartphones programmiert sein. Um ein möglichst großes Publikum ansprechen zu können, muss sie für iOS und Android erhältlich sein.

Die Datenbank mit den Benutzerinformationen muss jederzeit erreichbar sein und eine gewisse Form von Sicherheit und Verschlüsselung bieten, da da hier persönliche Angaben gespeichert werden.

Die Datenbank mit den Filminformationen muss sehr umfangreich sein, also auch ältere Filme und Nischenfilme enthalten, und ständig aktualisiert werden. Neben den offensichtlichen Informationen wie Filmtitel und Filmposter muss sie auch weitere Daten zu den Filmen enthalten wie Erscheinungsdatum, Handlung, Regie, etc. trotzdem sollte der Zugriff schnell sein und nichts kosten, sodass die Nutzung der App möglichst preiswert ist.

Der Server muss durchgehend laufen und über eine Internetverbindung erreichbar sein. Er muss leistungsstark genug sein um alle Nutzeranfragen bedienen zu können, sollte jedoch so klein gehalten werden, dass die Anschaffungs- und Unterhaltskosten minimal sind.

Der Messenger muss ebenfalls über einen zentralen Punkt gesteuert werden, da sich nur zwischen Personen ein Chat öffnen darf, die gematcht wurden. Es sollte keine Zugriffsbegrenzung auf diesen Server geben, um einen unbegrenzten Nachrichtenwechsel garantieren zu können. Nachrichten müssen gespeichert werden, um versendete Nachrichten auch nach dem Schließen der App zustellen zu können. Das System muss ebenfalls verschlüsselt sein und benötigt Zugriff auf Benutzerinformationen wie ID, Name und Profilbild.

X Server muss ständig laufen/erreichbar sein und Internetverbindung haben. In diesem Fall ein Raspberry Pi mit ... was eigentlich???

### 3.2 Komponenten

In StreamSwipe ist für jede dieser Komponenten eine Lösung gefunden worden. Bei der Entwicklung der App wurden Benutzerfreundlichkeit und Barrierefreiheit beachtet, wie in Abschnitt 6, bzw. Abschnitt 5.8 gezeigt wird. Wie in Abschnitt 2.8 beschrieben wird, können die genannten Anforderungen durch eine geschickte Wahl des Frameworks erfüllt werden. Als Benutzerdatenbank kann Firebase verwendet werden, da wie in Abschnitt 2.6 beschrieben, die benötigten Funktionen vorhanden sind. Es ist ebenfalls möglich den Messenger mit den Firebasefunktionen zu implementieren. Als Quelle der Filminformationen wird die API von „The Movie Database“ (TMDb) verwendet, auf welche in Abschnitt 4.4 genauer eingegangen wird. Der verwendete Server besteht aus einem Raspberry Pi, jedoch mehr dazu in Kapitel 4.1. Wie diese Komponenten aufeinander zugreifen wird in Abbildung 27 verdeutlicht.

Über die Smartphone-App werden die Benutzerdaten an Firebase weitergeleitet. Wird ein neuer Account angelegt, so werden diese Daten dort gespeichert und bei einem Anmeldevorgang sie mit den bestehenden Benutzerdaten verglichen. Nur wenn der User bereits vorhanden ist, kann eine Anmeldung stattfinden. So wird sichergestellt, dass nur angemeldete Benutzer auf die eigentlichen Funktionen der App Zugriff erhalten. Jede Aktion in der App wird auf diese Weise einem Benutzerkonto zugewiesen und kann später darüber identifiziert werden. Innerhalb der App können alle Funktionen frei genutzt werden weshalb es wichtig ist, dass ausschließlich eingeloggte User auf die Screens der App zugreifen können.

Der Server erhält die Film-IDs aller Filme aus der TMDb-API und erstellt somit eine eigene Film-Datenbank. Diese IDs werden an das Smartphone weitergeleitet und die App erhält über die IDs die Filminformationen direkt von der TMDb-API. Es werden gleichzeitig nur eine geringe Anzahl an Filminformationen aus der TMDb-API auf das Smartphone geladen, sodass dieser Vorgang möglichst wenig Ressourcen benötigt und unbemerkt im Hintergrund ablaufen kann. Hierzu zählen Filmposter, Rating, Besetzung, Veröffentlichungsdatum, übersetzte Sprachen und viele mehr, die teilweise nicht benötigt werden. Noch bevor der User über alle diese Filme abgestimmt hat, werden neue Filminformationen geladen, sodass es für den User keine Unterbrechungen gibt und es ihm wie ein einzelner unendlicher Fluss an Daten erscheint.

Die Filmbewertung wird mit den Film-IDs vom Smartphone an den Server geschickt, der diese beiden Informationen miteinander verknüpft. Das Rekommendationsverfahren auf dem Server verarbeitet diese Präferenzen und sucht wie in Kapitel 2.9 beschrieben nach Übereinstimmungen bei den anderen Benutzern. Da hierbei alle Präferenzen aller User miteinander verglichen werden, ist dieser Schritt sehr aufwändig und sollte nicht nach jeder neuen Präferenz durchgeführt werden. Bei StreamSwipe wird dieses Matching deshalb automatisch in regelmäßigen Zeitabständen durchgeführt, optimalerweise zu einer Tageszeit, an der die Benutzeraktivität gering ist. Die Anzahl der verglichenen Nutzer kann mit einer Filterung durch Geschlechterpräferenzen und Wohnort reduziert werden um das Matchingverfahren zu beschleunigen.

Werden zwei User gematcht, wird dies ihnen jeweils in der App angezeigt. Die beiden Benutzer können dann einzeln entscheiden, ob sie die Unterhaltung starten wollen. Ist dies der Fall, werden die Nachrichten in der Firebase-Datenbank gespeichert.

- Der Messenger kann ebenfalls über Firebase realisiert werden

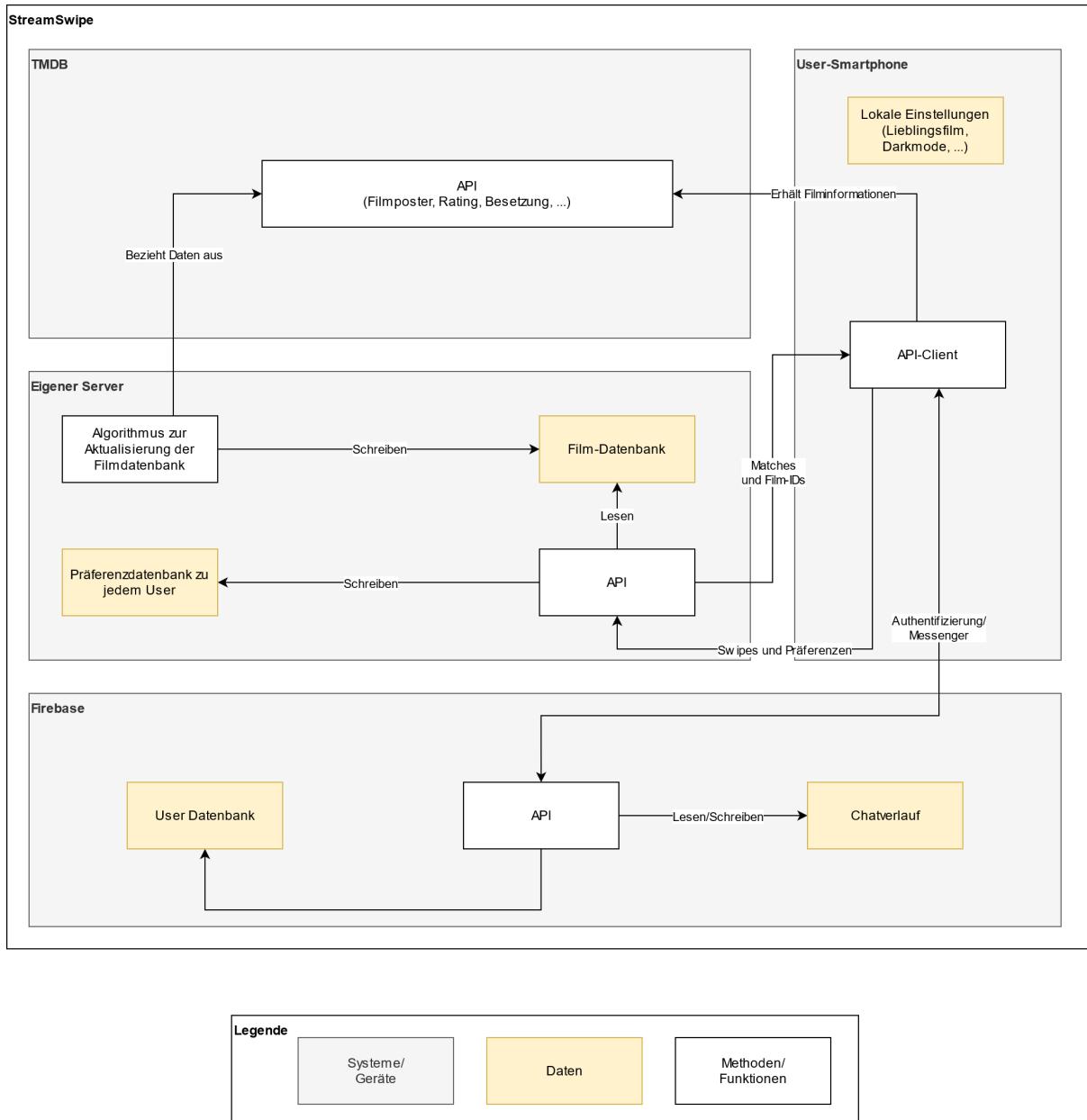


Abbildung 27: Komponentendiagramm

## 4 Auswahl geeigneter Technologie

Den Anforderungen der Entwicklung entsprechend wurde zunächst entschieden, welche Technologien zum Einsatz kommen.

### 4.1 Server

Der Webserver ist jener Dienst, der die zugrundeliegenden Funktionalitäten bezüglich der Filmabfragen, der Matching-Logik und der Filmempfehlung bietet. Anforderungen an die genutzte Webservertechnologie sind zum einen eine grundlegend hohe Performance, um mit einer hohen Anzahl an Serveranfragen umzugehen. Des Weiteren sollte die Technologie Skalierbarkeit in Bezug auf die Performance und wachsenden Ressourcen wie Hardware aufweisen und eine Unterstützung, Dokumentation und umfassende Funktionalitäten des Frameworks bieten. Ein Paketmanager, der umfassende Kern-Funktionalitäten zur Verfügung stellt, sollte vorhanden sein, damit diese nicht neu implementiert werden müssen. Um offen für das genutzte Zielbetriebssystem zu bleiben, sollten mehrere Betriebssysteme unterstützt werden. Im folgenden werden keine proprietären Webserver-technologien betrachtet.

Nach genauerer Recherche kamen drei Webservertechnologien in die engere Betrachtung:

- PHP
- Django
- Node.js

Im Hinblick auf die Performance sticht Node.js aufgrund seiner ereignisgesteuerter Architektur und dem Non-Blocking I/O-Mechanismus heraus und verspricht eine bessere Ressourcenutzung.

Große Firmen wie Uber [Tech1], Ebay und Netflix [Tech2] haben ihre Systeme bereits auf Node.js umgestellt. Die Wahl als Webservertechnologie fällt auf Node.js, da es breite Unterstützung erfährt, die auch durch den mächtigen Paketmanager npm ergänzt wird und eine hohe Performance errichtet.

### 4.2 Datenbank

Im Hinblick auf die Speicherung der potenziell hohen Anzahl an Nutzern, deren Swipe-Entscheidungen und deren Matches untereinander sowie der Anzahl von über 500.000 Filmen [Tech4] wird ein performanter Umgang der Datenbank mit vielen Datensätzen notwendig sein. Um massive Daten speichern zu können sind relationale Datenbanken nicht die passende Wahl. Es hat sich gezeigt, dass je größer die Menge an Daten ist und je mehr Tabellen in einer Anfrage enthalten sind, desto größer ist der Performanceverlust durch SQL. [Tech4.5]

Die Verwendung von dokumentenbasierten Datenbanken führt dagegen zu einer strukturlosen Zusammensetzung an Daten, bei denen ein Dokument ein einzelnes Objekt repräsentieren kann. Somit muss für die Wiedergabe eines Objekts nur ein Dokument angefragt werden. Die fehlenden Möglichkeiten zur Normalisierung können jedoch zu Redundanzen in den Daten führen, wodurch die Entwicklung der aufrufenden Anwendung komplexer werden kann. Die Redundanz wird jedoch in Kauf genommen, um schnelle Abfragen zu ermöglichen und eine hohe Performance zu erhalten. Da Datensätze in dokumentenbasierten NoSQL-Datenbanken schemilos als JSON-Objekte abgelegt werden, begünstigt dies den generischen Dokumentenaufbau in der Entwicklung.

Einige NoSQL-Datenbanken wie MongoDB verfolgen einen nicht-relationalen Ansatz und werden mit JavaScript-fähigen Schnittstellen bereitgestellt. Durch die Kommunikation im JSON-Format

eignen ist ein optimaler Einsatz mit Node.js gegeben. MongoDB ist über seine horizontale Skalierbarkeit darauf ausgelegt, in einem kurzen Zeitraum sehr viele Daten zu verarbeiten [Tech6]. Während relationale Systeme vertikal im Sinne von neuen Tabelleneinträgen skalieren, werden Dokumente hingegen werden in einer Kollektion, die horizontal erweitert werden können, indem Datenmengen im Sinne des Shardings auf mehreren Systemen verteilt werden, anstatt ein einzelnes System zu verwenden.

Als Datenbank für die Backend-Implementierung wurde aufgrund von Performance, der guten Einbindung an Node.js und der Skalierbarkeit MongoDB ausgewählt. Um die Vorteile der Datenkompression sowie der Transaktionen über mehrere Dokumente hinweg zu nutzen, ist die WiredTiger-Engine die Wahl für das genutzte Storage-Engine.

### 4.3 Kommunikationsschnittstelle

Als Kommunikationsschnittstelle wird eine WebApi entwickelt, dessen Kommunikation auf HTTP-Nachrichten basiert, deren Informationen im JSON-Format übergeben werden. Diese Technologie bietet eine einfache und dennoch effiziente Form der Kommunikation zwischen Server und Client.

### 4.4 Film-Datenbank

=<sub>z</sub>TMDb API, <https://developers.themoviedb.org/4/getting-started/authorization>

OMDb API, <http://www.omdbapi.com/>

Verschiedene kleinere Anbieter, <https://rapidapi.com/search/movie>  
vince?

rapid API: - Response Time von über 1 Sekunde (rapid am Arsch) [<https://rapidapi.com/de/webknox/api/response-time>] -

IMDb: - keine Dokumentation - Gratisversion mit begrenzter Zugriffszahl [<https://rapidapi.com/blog/how-to-use-imdb-api/>]

OMDb: - Poster API nur für Patreons - keine Dokumentation -

TMDb: - Gute Response Time - keine Zugriffsbegrenzung - Kostenlos - Umfangreiche Informationen wie Trailer und viele weitere Metadaten

Github.

## 5 Backend-Implementierung

Unter Backend versteht man... , Sie differenziert durch... von den ... .

### 5.1 Server

Hier steht mein BackendServer Text.

#### 5.1.1 Einrichtung

Zunächst...

#### 5.1.2 Sicherheit

Den Server gilt es zu schützen.

#### 5.1.3 Webserver

Den Server gilt es zu schützen.

### 5.2 Datenbank

Hier steht mein BackendDatenbank Text.

#### 5.2.1 Einrichtung

Hier steht mein Imlementierung Text.

### 5.3 Kommunikationsschnittstelle

Hier steht mein BACKENDKommunikationschnittstelle Text.

#### 5.3.1 Implementierung

#### 5.4 Swipe/Aussuchen/Voting

#### 5.5 Matches/Chat

#### 5.6 Film-/Serienvorschläge

#### 5.7 Gespeicherte Filme/Filmliste

#### 5.8 Barrierefreiheit

Barrierefreiheit im Allgemeinen bedeutet, dass ein Gegenstand, eine Einrichtung oder Informationsquelle für Menschen mit Behinderung ohne Unzulänglichkeiten nutzbar, zugänglich oder auffindbar ist ([17], §4). In der Softwareentwicklung versteht man darunter Applikationen für Menschen mit Einschränkungen zugänglich und bedienbar zu machen. Bezogen auf die Entwicklung von mobilen Apps gilt es dabei den akustischen, optischen oder motorischen Einschränkungen der Benutzer entgegenzuwirken.

### 5.8.1 Barrierefreiheit in mobilen Anwendungen

Mit der Verbreitung von Smartphones ist die Benutzung mobiler Apps stark angestiegen und mittlerweile in nahezu jedem Haushalt aufzufinden. Obwohl etwa 9,5% aller in Deutschland lebenden Menschen einen Schwerbehindertenausweis besitzen (Stand 24.06.2020)[16] was etwa 7,9 Millionen Menschen entspricht, ist die Implementierung von barrierefreier Bedienung nicht selbstverständlich. Gerade Programmierer/innen aus dem privaten Sektor sind diese Funktionen oft nicht bekannt, es besteht kein Interesse oder sie werden schlichtweg vergessen. Software, die für öffentliche Einrichtungen entwickelt wird, ist durch das Behindertengleichstellungsgesetz von 2002 dazu verpflichtet ihr Softwareangebot bis spätestens dem 23. Juni 2021 barrierefrei zu gestalten ([17], §12a Abs.1). Hierzu zählen sämtliche Webseiten sowie mobile Anwendungen.

### 5.8.2 Barrierefreiheit in Filmen und Serien

Auch die Zugänglichkeit von Filmen und Serien für Menschen mit eingeschränkter Wahrnehmung wurde in den letzten Jahren stark verbessert. Hierbei lässt sich zwischen optischer und akustischer Einschränkung differenzieren. Für hörgeschädigte Personen werden bereits seit mehreren Jahrzehnten Untertitel eingesetzt. Was früher für einzelne Filme durch eine Funktion des Teletextes erreicht wurde, wird heutzutage durch eine integrierte Funktion des Videoplayers verwirklicht. Immer mehr Videos werden mit Untertiteln veröffentlicht. Manche Anbieter wie beispielsweise die Internetplattform YouTube bieten durch Spracherkennung automatisch generierte Untertitel an, was eine flächendeckende Untertitelung ermöglicht.

Auch für Menschen mit eingeschränktem Sehvermögen werden Filme und Serien mithilfe von Audiodeskriptionen vermehrt zugänglich gemacht. Hierbei wird die bereits vorhandene Tonspur mit Bildbeschreibungen und Kommentaren versehen. Was bis vor wenigen Jahren noch etwas Besonderes war und nur für ausgewählte Filme bestimmt war, ist heutzutage Standard. Größere Video-On-Demand-Plattformen wie Netflix oder Amazon Prime bieten diese Möglichkeit bei nahezu allen Eigenproduktionen an. Zusätzlich werden bestehende Filme neu mit Audiodeskriptionen versehen.

Hieraus lässt sich leicht erkennen, dass Filme und Serien heutzutage auch von Menschen mit Einschränkungen genutzt werden. Was auf den ersten Blick vielleicht nicht bedacht wird oder als unwichtig abgestempelt wird, kann einen nicht unerheblichen Vergrößerungsfaktor für den Kundenstamm bewirken. Für die Entwicklung einer mobilen App, bei der Filme und Serien bewertet werden, spielt also die Barrierefreiheit eine wichtige Rolle und darf auf keinen Fall vernachlässigt werden.

### 5.8.3 Barrierefreiheit bei StreamSwipe

Bei der Entwicklung von StreamSwipe werden mehrere mögliche Einschränkungen der User betrachtet und entsprechend reagiert. Ziel ist es, dass sowohl der Kunde sowie der Anbieter maximal davon profitieren. Hierfür soll die App für ein möglichst großes Publikum zugänglich gemacht werden, jedoch auch sogenanntes Over-Engineering vermieden werden, da zu viele Funktionen eine App unübersichtlich, teuer und langsamer werden lassen.

Allgemein wird Leserlichkeit durch große Schriftgrößen, hohe Farbkontraste, große Schaltflächen oder universelles Design erreicht. Alleine in Deutschland tragen 44,5 Millionen Menschen regelmäßig eine Brille oder Kontaktlinsen und benötigen somit Sehhilfen [18]. Unterstützung auf Seiten der App kann hierfür durch vergrößerbaren Text geschehen. Da aber davon ausgegangen werden kann, dass Personen, die sich auf Sehhilfen verlassen, bereits eine Brille oder Kontaktlinsen besitzen, wird die Textgröße vorerst nicht variabel gehalten. Außerdem gibt es bei

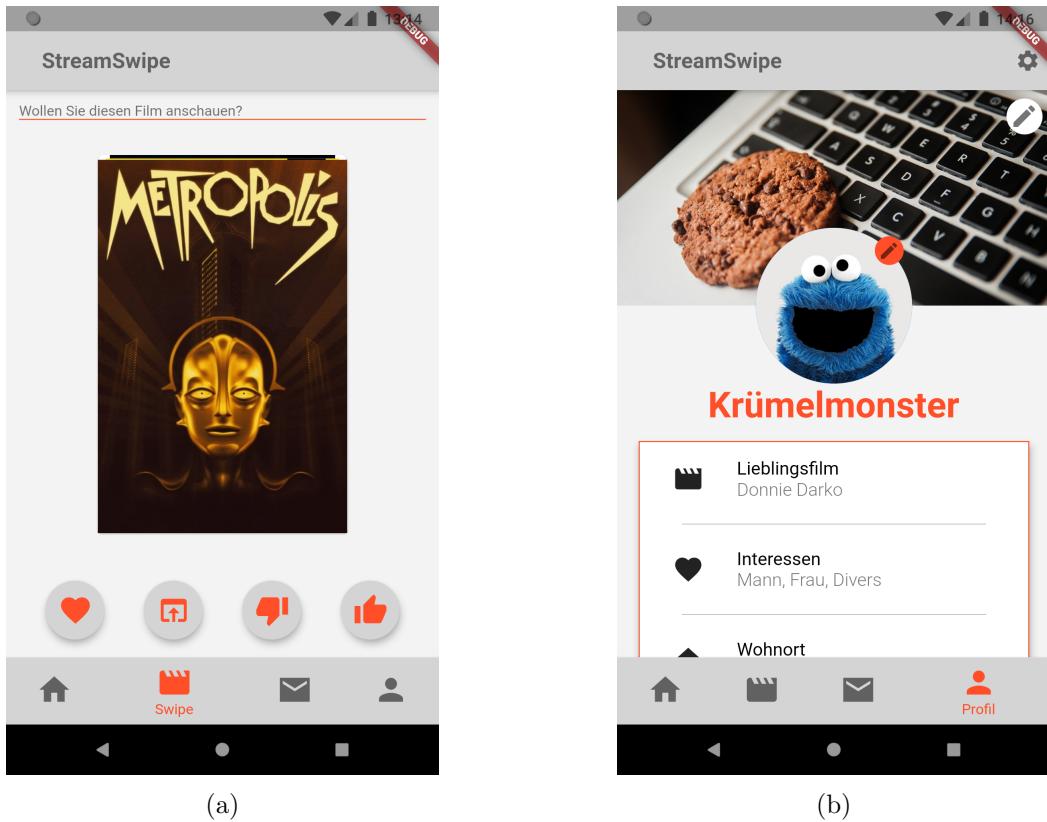


Abbildung 28: Screenshots aus der App StreamSwipe als Beispiele zu (a) schlichtem Design, bei dem farbige Akzente nicht der Informationenübertragung dienen um die Zugänglichkeit für farbblinde Menschen zu verbessern und für einen Icon in (b), welcher sonst durch sehgeschädigte Menschen nicht wahrnehmbar ist, wird exemplarisch eine Semantik implementiert.

Android- und Apple-Smartphones bereits eingebaute Vergrößerungsfeatures, die Bildausschnitte vergrößert darstellen können. Aus diesem Grund wird in diesem Projekt kein Fokus auf dieses Feature gelegt.

Farbblindheit kann jedoch in vielen Formen auftreten. Um der bekannten Farbfehlsicht entgegenzuwirken, werden Farben aus Problembereichen wie Rot und Grün nicht nebeneinander benutzt. Allgemein wird ein schlichtes Design gewählt und Farben nur zu Akzentuierung und als Stilmittel benutzt, statt als Informationsträger wie beispielsweise in den Abbildungen 28a erkennbar ist. Geringe Sehschärfe durch Achromatopsie kann wie weiter oben beschrieben umgangen werden.

Ist die Sehkraft noch weiter eingeschränkt oder gar nicht mehr vorhanden, werden Semantiken eingesetzt. Hierbei erhält jedes Element auf dem Bildschirm eine Beschreibung, die vorgelesen werden kann. Bei Zahlen und Texten werden diese vorgelesen, sofern keine weitere Information hinterlegt ist. Besonders hilfreich ist dies jedoch bei Abbildungen. Ausgeführt wird das Auslesen von einem Screenreader. Mobile Geräte haben diese Funktion bereits standardmäßig eingebaut (VoiceOver bei Apple und TalkBack bei Android) und wandeln die Semantiken mittels Sprachsynthese in akustische Signale um. Bei Desktopanwendungen wie z.B. JAWS für Windows können diese Informationen zusätzlich auch durch eine Braillezeile wiedergegeben werden.

Bei Flutter ist das Hinzufügen von Semantiken bereits eingebaut. Hierfür kann ein String dem jeweiligen Bereich zugeordnet werden. In Beispiel 27 ist hierfür der Code des Buttons, der zu den Einstellungen führt. In Abbildung 28b ist dieser Button ganz rechts oben im Eck zu sehen. Die Funktion `GestureDetector()` erkennt Interaktionen mit dem Touchscreen, wobei hier nur

auf Antippen reagieren soll, deshalb die Funktion `onTap: () {}`, die auf den Einstellungsbildschirm leitet. Diese Implementierung ist hier aber nicht von Relevanz und wird übersprungen. In dem `GestureDetector()` ist ein Icon eingebettet, von der Form *Settings*, was einem Zahnräder entspricht. Dieses Icon erhält eine Farbe und anschließend eine Semantik aus allem was in den Anführungszeichen steht. Ein Screenreader kann AE erkennen und ihn als den Umlaut Ä aussprechen.

So wird im kompletten Programm für jedes relevante Element vorgegangen. Teilweise müssen den Semantiken Variablen übergeben werden, da sich die vorzulesende Information ändert wie beispielsweise bei den Filmtiteln.

```

1 GestureDetector(
2   onTap: () {
3     ...
4   },
5   child: Icon(
6     Icons.settings,
7     color: Provider.of(context).colors.textSmall,
8     semanticLabel: "Einstellungen. Zum Auswählen doppeltippen.",
9   )
10),

```

Listing 27: Codeausschnitt in Dart von einem Button mit Semantiken.

Bei einer sauberen Implementierung wird auf diese Weise vorgegangen und eine bereits vorhandene Funktion verwendet. Dies vereinfacht nicht nur die Leserlichkeit des Codes, sondern bietet auch die höchste Modularität, da hierbei normalerweise standardisierte Schnittstellen für Betriebssysteme oder andere Anwendungen verwendet werden. In diesem Fall müssen die Screenreader von Android und Apple damit arbeiten können.

Um für Personen mit eingeschränktem Hörvermögen oder vollständiger Gehörlosigkeit die App zugänglich zu machen, wird auf akustisches Feedback als notwendige Informationsübertragung verzichtet. Innerhalb der App werden keine Geräusche erzeugt, außer der oben beschriebenen Funktion der Semantiken. Beim Erhalten einer neuen Nachricht oder eines neuen Matches kann weiterhin optional eine akustische Benachrichtigung erhalten werden. Hierbei wird die betriebssystemeigene Funktion übernommen, sodass in der App keine neuen Einstellungen vorgenommen werden müssen.

Auch feinmotorische Einschränkungen werden versucht zu umgehen. Die Navigation und die Filmbewertung in StreamSwipe können durch großflächige Wischbewegungen ausgeführt werden. Wo diese Lösung nicht möglich ist, werden verhältnismäßig große Buttons eingesetzt. Lediglich beim Registrieren und Einloggen werden feine Bewegungen erforderlich. Hierbei öffnet sich allerdings die als Standard eingestellte digitale Tastatur, die in vielen Fällen eine Spracheingabe besitzt, sodass die sehr kleinen Tasten nicht benutzt werden müssen.

Sollte sich in Zukunft jedoch Kritik in Form von negativen Nutzerbewertungen herauskristallisieren, kann eines der noch nicht implementierten Features über ein Update nachgerüstet werden.

## 6 Benutzeroberflächen

Die Benutzeroberfläche einer Software muss im Grunde genommen nur einen Informationsfluss in zwei Richtungen erzeugen. Die eine Richtung liefert Informationen an den User und über die andere kann der User Informationen an das System weitergeben. Um auf dem heutigen Markt Fuß fassen zu können, sollte eine Oberfläche jedoch wesentlich mehr Aspekte erfüllen.

### 6.1 Aspekte von Benutzeroberflächen

Die Vielschichtigkeit einer Benutzeroberfläche kann ausschlaggebend für den Erfolg einer Applikation sein, abhängig davon welche Erfahrungen der User mit der Oberfläche macht und welche Eindrücke sie hinterlässt. Hieraus resultiert wie lange ein User auf der App bleibt und wie oft er zurück kommt. Neben der Nutzungszeit erhöht eine positive User Experience die Weiterempfehlungsrate.

Bei erfolgreicher Software besteht ein großer Teil der Entwicklung in der Planung der Oberfläche, da die User Experience nicht zu umgehen ist. Auf die eine oder andere Art erlebt der User immer eine Erfahrung. Neben den offensichtlichen Aus- und Eingabefunktionen werden beispielsweise folgende Kriterien betrachtet:

**Simpel:** Ausgegebene Information kann zum Beispiel durch Icons, Farben oder Symbole vereinfacht werden. Eine Oberfläche sollte weder überladen sein, noch sollten alle Ein- und Ausgaben auf verschiedenen Screens verteilt sein. Bei der Entwicklung wird eine gesunde Mischung aus maximaler Funktionalität und einfacher, übersichtlicher Darstellung angestrebt.

**Einheitlich:** Die Bedienung und das Lesen von Applikationen kann erheblich vereinfacht werden wenn einheitliche Bedien- oder Ausgabeelemente verwendet werden. Nicht nur innerhalb einer App ist es sinnvoll konsistente Elemente in der Oberfläche zu verwenden, auch Funktionen von anderen Apps können die Bedienung vereinfachen. Bekannte Funktionen bei Smartphone-Applikationen sind zum Beispiel die Vergrößerung mit zwei Fingern oder das „Daumen nach oben“-Symbol als positive Rückmeldung. Durch das Einbauen solcher Features wird eine App intuitiv und ohne Einführung bedienbar.

**Benutzergesteuert:** Alle ausgeführten Aktionen sollten vom Benutzer ausgehen. Ein gutes Interface unterstützt den User lediglich bei seiner Bedienung, schränkt ihn aber nicht ein. Mit der heutigen Technologie ist die Verführung groß viele Funktionen automatisch ablaufen zu lassen. Was eigentlich der Sinn einer Applikation ist, kann jedoch auch negative Folgen haben. Zu viel Automatisierung verursacht das Gefühl von Kontrollverlust und Unsicherheit, was sich negativ auf das Vertrauen und somit auf die Nutzungszeit von dem User auswirkt.

**Klarheit:** Eine mobile App muss ohne Anleitung bedienbar sein. Sobald Unklarheiten beim User entstehen und Funktionen oder Ausgaben nicht erkannt werden können, verliert die Anwendung auf dem freien Markt.

Der User sollte zu jeder Zeit wissen welche Optionen ihm zur Verfügung stehen und welche Folgen seine Aktionen haben. Besonders wichtig ist das Feedback infolge einer Aktion. Auch wenn diese Aspekte offensichtlich erscheinen, können sie bei der Entwicklung einer App leicht übersehen werden. Verwendet werden einfache und für den User bekannte Funktionen, wie die Beschriftung aller Buttons oder das haptische, akustische oder optische Feedback beim drücken eines dieser Buttons.

**Benutzerfreundlich/Barrierefreiheit:** Die Bedienung der App sollte für Menschen mit Einschränkungen im vollen Umfang möglich sein. In Abschnitt 5.8 wird auf dieses Thema tiefer eingegangen. Aber auch Benutzer ohne Einschränkungen erwarten eine einfache und übersichtliche Bedienung, die auch beispielsweise Eingabefehler mit mehreren Versuchen verzeiht.

**Ästhetik:** Das Design spielt bei dieser Betrachtung gleich mehrere wichtige Rollen. Es sollte eine angenehme Arbeitsumgebung für den User erstellen, Ein- und Ausgaben verdeutlichen und gleichzeitig mithilfe eines eigenen Stils ein einzigartiges Image für die App schaffen (sogenanntes Branding) um deren Individualität und Wiedererkennungswert zu steigern. Das Design erschafft ein Erlebnis während der Benutzung und weckt unterbewusst Gefühle im User.

Gerade weil viele dieser Aspekte unterbewusst wirken, ist eine ausgiebige Betrachtung unumgänglich.

Eine Schwierigkeit, die sich bei der Entwicklung ergibt sind die zwei unterschiedlichen Ziele. Einerseits sollten bestehende Design- und Bedienelemente übernommen werden um die Bedienung intuitiv und übersichtlich zu gestalten, andererseits aber auch neue Ideen und Innovationen eingebracht werden, um sich von anderen Apps abzuheben und bleibenden Wiedererkennungswert aufzubauen.

## 6.2 Oberflächen von StreamSwipe

Die Smartphone-App lässt sich in mehrere Bereiche aufteilen, die sich in ihren Funktionen unterscheiden. Auf Basis der oben beschriebenen Grundlagen wurden diese Bereiche entworfen und werden in diesem Kapitel analysiert. Auch wenn manches davon als gewöhnlich oder naheliegend erscheint, so ist jedes Element mit Bedacht gewählt, erstellt und angepasst worden.

### 6.2.1 Login-Screen

Bei erstmaliger Benutzung der App öffnet sich der Login-Screen. An diesem Punkt wird der erste Eindruck für den Benutzer gesetzt, wobei bei StreamSwipe ein schlichtes Design gewählt wurde. Man sieht helle Grautöne mit einem Akzentfarbton, welche sich durch alle Bildschirme der App ziehen werden. Abhängig davon, ob der User in den Systemeinstellungen den dunklen Modus gewählt hat, werden anstatt den hellen Grautönen, dunkle bis schwarze Farben dargestellt, siehe auch Abbildungen 30c und 31e.

Auf dem Login-Screen (siehe Abbildung 29a) sind neben einer Überschrift mehrere beschriftete Textfelder und Buttons zu sehen, welche allesamt mit Semantiken versehen wurden, um durch einen Screenreader erkannt und identifiziert werden zu können. Die gewählte Anordnung wird universell bei Apps, Programmen und Webseiten benutzt, sodass die Felder auch ohne die eingetragenen Hinweistexte korrekt ausgefüllt werden könnten. Beim Antippen der Textfelder, öffnet sich die Standardtastatur des Betriebssystems. Sind alle Felder korrekt ausgefüllt, wird der User in die eigentliche App weitergeleitet, ansonsten wird durch individualisierte Fehlermeldung auf eventuelle Falscheingaben hingewiesen. Nach Erstellen eines neuen Accounts, durchläuft der User einen ähnlich aufgebauten Bildschirm (siehe Abbildung 29b) und wird danach aufgefordert weitere Informationen zur Profilvervollständigung einzugeben (siehe Abbildung 29c und 29d). Auch hierbei werden bekannte Bedienelemente wie Textfelder, Dropdownmenüs und Checkboxen verwendet, wie in der Abbildung 29e beispielhaft dargestellt ist. Falls der User in den Systemeinstellungen des Smartphones den Nachtmodus aktiviert hat, wird das Appdesign angepasst, siehe Abbildung 29f.

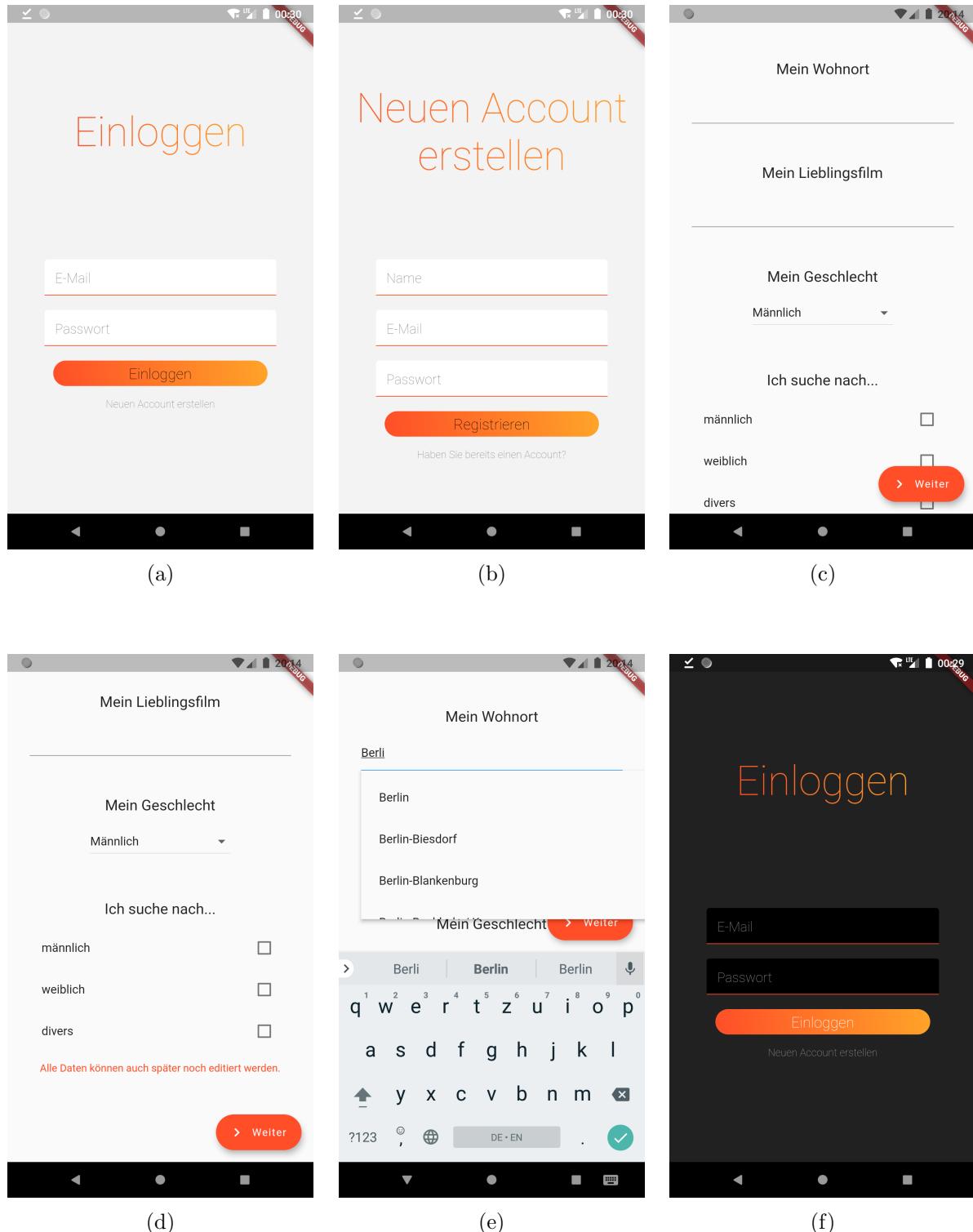


Abbildung 29: Die Anmeldeseiten von StreamSwipe und alle damit zusammenhängenden Screens. Man sieht (a) das Einloggen bei bestehendem Account, (b) das Erstellen eines Accounts, (c) und (d) das Formular für die benötigten Profildaten, (e) ein Texteingabefeld mit Auto vervollständigung als Dropdownmenü und (f) das Farbschema der Anmeldeseiten im Dark-mode am Beispiel des Login-Screens.

### 6.2.2 Home-Screen

Da davon ausgegangen wird, dass der User sich nicht nach jeder Nutzung ab- und wieder anmeldet, erscheint im alltäglichen Gebrauch der in Abbildung 30 dargestellte Bildschirm zuerst. Demnach bietet es sich an Ereignisse wie neue Matches und neue Nachrichten hier anzuzeigen. Diese werden wie Abbildungen 30a und 30b zeigen klar strukturiert in Abschnitte eingegliedert, welche mit Überschriften kenntlich gemacht sind. Die einzelnen Matches befinden sich mit allen dazugehörigen Funktionen und Informationen jeweils auf einer Karte. Durch diese Karten kann mit einer von anderen Apps bekannten horizontalen Swipemechanik navigiert werden. Weiterführende Funktionen wie das Starten eines Chats oder das Löschen des Matches werden durch Antippen von allgemein verständlichen Icons ausgeführt.

Auch auf diesem Screen findet sich einerseits das bereits eingeführte Farbschema wieder und es werden andererseits ebenfalls Semantiken verwendet. Bei den neuen Nachrichten wird jeweils der Benutzername vorgelesen und bei den neuen Matches je nach ausgewähltem Bereich der Filmname, die Icons oder der Text dazwischen.

Am unteren Bildschirmrand ist eine sogenannte Bottom-Navigation-Bar zu sehen. Sie ermöglicht eine kompakte und anschauliche Navigation durch die relevanten Bildschirme. Außerdem zeigt sie an welcher Bildschirm aktuell ausgewählt ist, wobei diese Information wie in Abschnitt 5.8.3 erarbeitet nicht ausschließlich auf einer Farbänderung basieren sollte und deshalb das ausgewählte Icon durch Hinzufügen von Text hervorgehoben wird. Liest der Screenreader die Semantik hiervon, gibt er die Bezeichnung des aktuellen Bildschirms sowie die Anzahl der weiteren Möglichkeiten an.

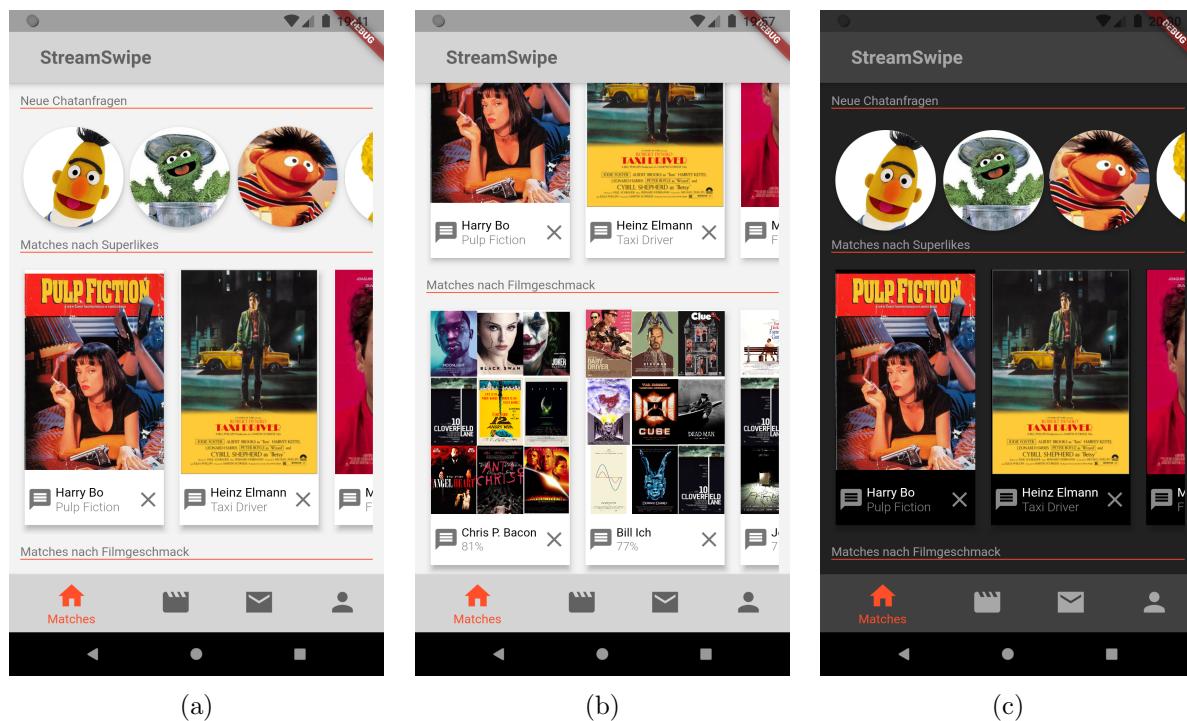


Abbildung 30: Der Home-Screen, der beim Öffnen der App zuerst gezeigt wird und Neuigkeiten wie neue Nachrichten und Matches zusammenfasst. Um den gesamten Inhalt dieser Seite sehen zu können, wird in (a) der obere Abschnitt und in (b) der untere Abschnitt gezeigt. Hat der User in den Systemeinstellungen den dunklen Modus aktiviert, so wird (c) der Home-Screen wie alle anderen Screens angepasst.

### 6.2.3 Swipe-Screen

Auf dem Swipe-Screen (Abbildungen 31) findet die Bewertung der Filme statt. Durch das hier verwendete Matchingsystem mithilfe des Filmgeschmacks unterscheidet sich StreamSwipe von anderen Apps und erhält so einen innovativen, individuellen Charakter, womit diese Seite das Herzstück der App bildet.

Das zuvor eingeführte Farbschema bleibt auch hier erhalten, wie Abbildung 31a zeigt. Eine Überschrift im selben Stil wie bereits aus Abschnitt 6.2.2 bekannt, verdeutlicht durch eine Frage nach welcher Motivation die Filmauswahl getroffen werden soll. Zentral im Bild ist eine Liste von Postern der zu beurteilenden Filme. Wie bereits durch die Datingapp Tinder verbreitet, werden die vier Antwortmöglichkeiten durch eine Swipe-Bewegung in eine der vier Richtungen ausgewählt. Abhängig von der Position des Fingers auf dem Touchscreen bewegt sich das Filmposter innerhalb des Bildschirms, was den Effekt einer frei beweglichen Karte hervorruft. Um klarzustellen welche Swipe-Richtung für welche Entscheidung steht, verfärbt sich der jeweilige Indikator in der unteren Reihe bei Verschiebung des Filmposter. Beide diese Animationen sind in Abbildung 31d zu sehen. Die Indikatoren sind mit Icons versehen, zeigen aber durch Drücken welche Entscheidung sie repräsentieren und in welche Richtung der User dafür swipen muss, wie Abbildung 31c am Beispiel des rechten Indikators zeigt.

Durch Antippen des Filmposters werden weitere Informationen zu dem jeweiligen Film dargestellt, wie in Abbildung 31b zu sehen. Gleichfalls wird durch ein einfaches Antippen wieder zurück zu den Postern gewechselt. Eine Rotations-Animation verdeutlicht die Illusion der Karten.

Alle diese für die Bedienung der App grundlegenden Steuerungen verlangen keine feinmotorischen Eingaben und können problemlos von Personen mit motorischen Einschränkungen genutzt werden. Auch dieser Bildschirm ist vollkommen mit Semantiken ausgestattet. Anstelle des Filmposters wird der Name des Films ausgelesen und für die vier Indikatoren am unteren Rand werden jeweils deren Funktion und durch welche Swipe-Richtung sie erreicht werden vorgelesen. Sämtliche Textfelder können ebenfalls problemlos von einem Screenreader gelesen werden.

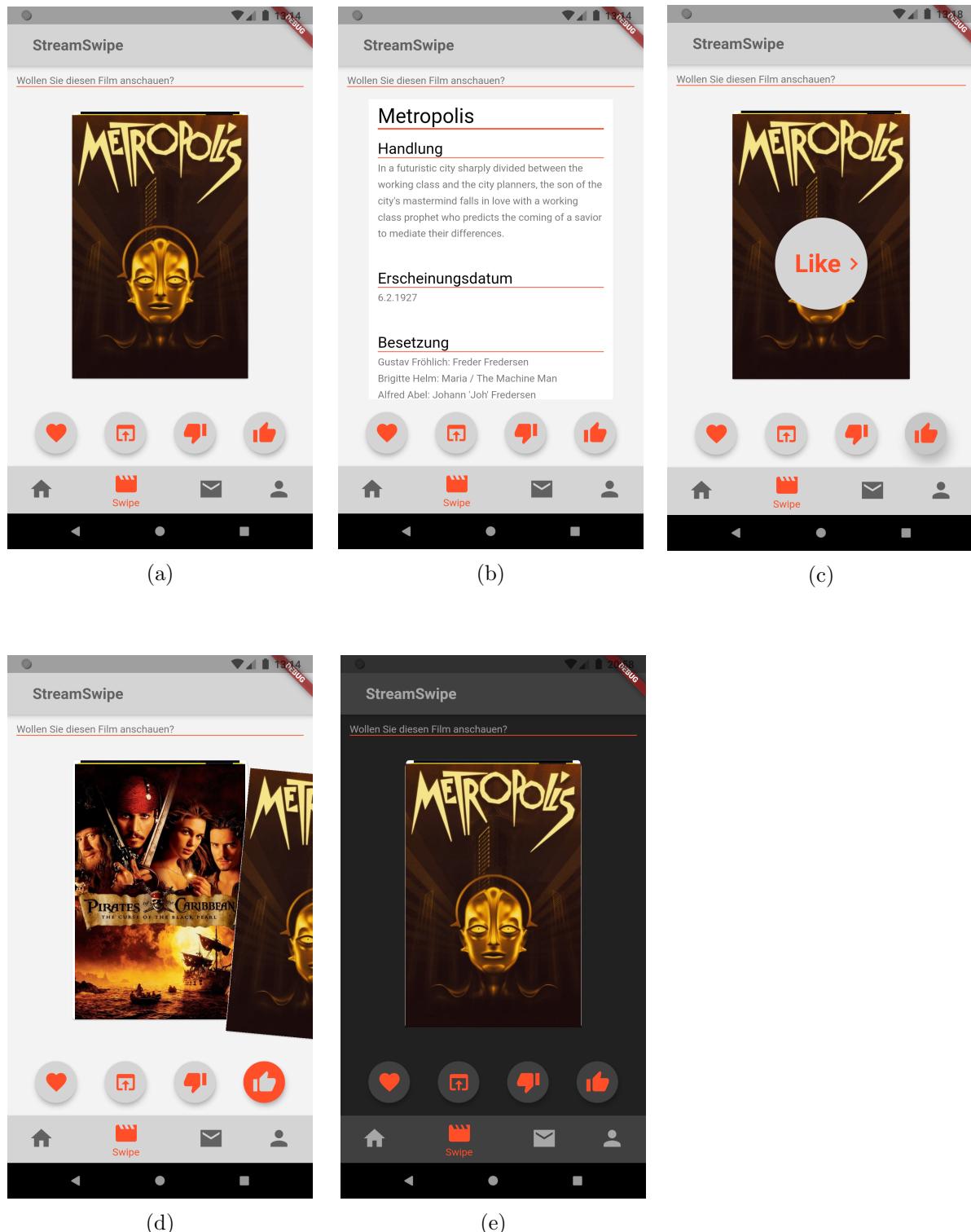


Abbildung 31: Darstellungen und Funktionen des Swipe-Screens mit (a) der Standarddarstellung, (b) weiteren Filminformationen, (c) einer Animation beim Drücken einer der Indikatoren und (d) der Swipe-Animation. Hat der User in den Systemeinstellungen den dunklen Modus aktiviert, so wird (e) der Swipe-Screen wie alle anderen Screens angepasst.

#### 6.2.4 Chat

Die Chatseite ist in eine Liste aus aktiven Chats und eine Warteliste aufgeteilt, siehe 32a und 32b. Um zwischen diesen beiden Listen zu wechseln werden Tabs eingesetzt, wie sie aus Windowsanwendungen bekannt sind. Zwischen diesen Tabs kann entweder gewechselt werden, indem ein anderes Tabfenster angetippt wird, oder der gesamte Bildschirm mit einer Geste zur Seite gewischt wird. Bei jedem Element der Chatliste ist der jeweilige Benutzername und die neueste Nachricht zu sehen, dazu wird falls vorhanden entweder ein Profilbild oder eine einfarbige Fläche mit dem Anfangsbuchstaben des Namens angezeigt. Chat Requests können jeweils durch das Schieben nach links angenommen oder nach rechts ablehnt werden, wie in den Abbildungen 32c, bzw. 32d zu sehen ist. Diese Mechanik wird häufig in Email-Apps zum Löschen oder Verschieben der Mails benutzt.

Durch Antippen eines Matches, öffnet sich der Chatverlauf, welcher in Abbildung 32e zu sehen ist. Die Anordnung der Nachrichten innerhalb des Chatverlaufs ist wie aus anderen Messenger bereits bekannt, aber in den Stilfarben von StreamSwipe. Am oberen Bildschirmrand wird der Profilname des Matches angezeigt und rechts davon befindet sich der Button zu dessen Profilseite, auf der genauere Details über diese Person zu finden sind. Die Profilseiten werden in Kapitel 6.2.5 genauer vorgestellt.

Dem User wird durch das ihm bereits vorgestellte Design und der ausschließlichen Nutzung von bekannter Mechanik ein vertrautes Umfeld geboten. Wie auf jedem Screen passt sich auch hier das Farbschema automatisch an, falls in den Systemeinstellungen des Smartphones das dunkle Design gewählt wurde, wie beispielsweise in Abbildung 32f dargestellt. Neben der Benutzerfreundlichkeit wird auch die Barrierefreiheit beachtet, indem alle Elemente, die nicht bereits aus einem Text bestehen, mit Semantiken ausgestattet werden. Zusätzlich werden keine feinmotorischen Bewegungen zur Navigation durch die Bildschirme benötigt. Bis auf die Eingabe über die Tastatur kann alles über große Flächen oder Wischmechaniken bedient werden. Im Chatverlauf wird die Standardtastatur des Systems verwendet, mit der der Benutzer bereits vertraut ist.

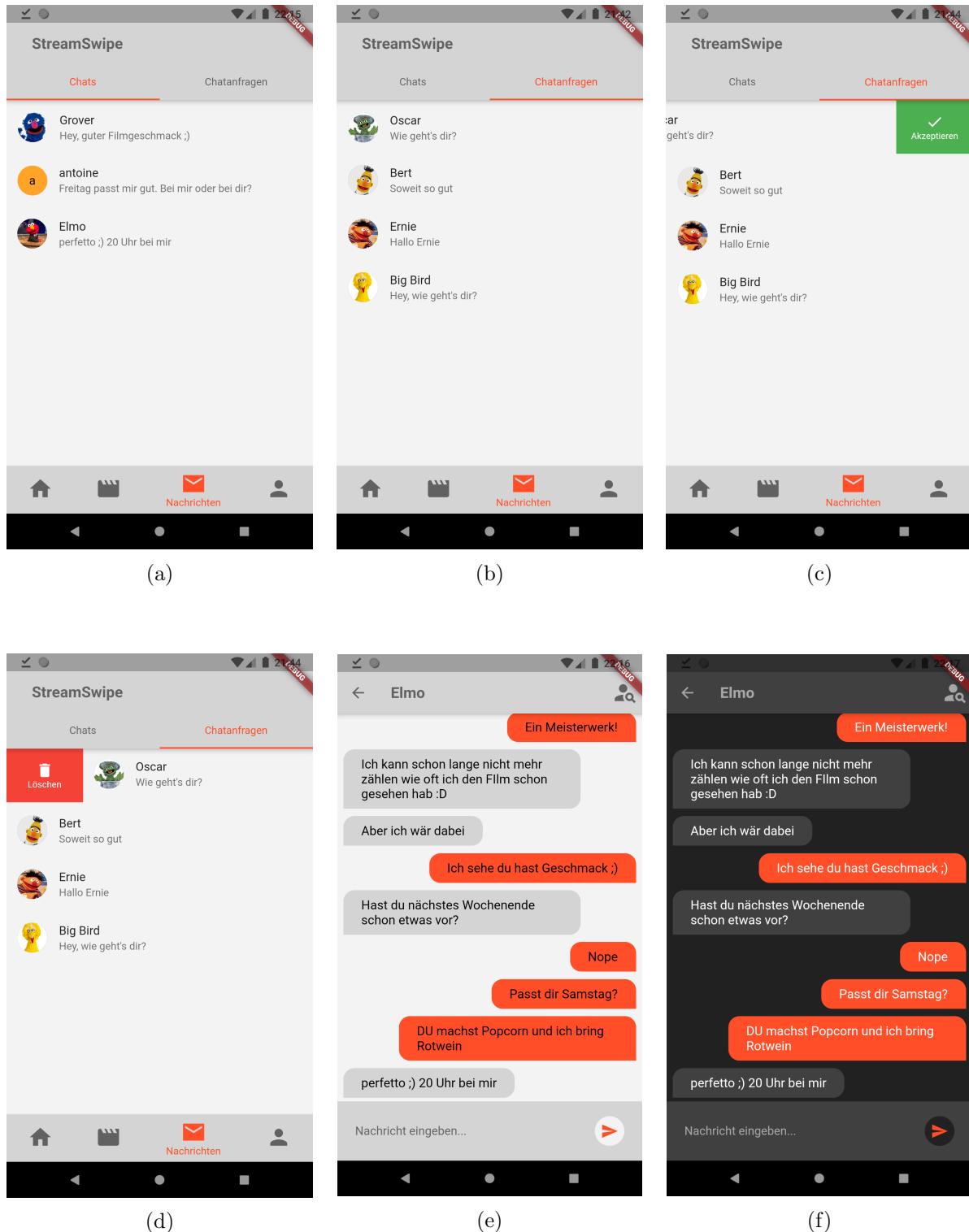


Abbildung 32: Darstellungen und Funktionen der Chat-Screens mit (a) den aktiven Chats, (b) den Chats auf der Warteliste, (c) und (d) angenommene, bzw. abgelehnte Chats auf der Warteliste, sowie (e) einem Chatverlauf im hellen und (f) in dunklen Modus.

### 6.2.5 Benutzerprofil

Auf der Profilseite werden ein Profilbild, ein Hintergrundbild und für das Matching relevante persönliche Informationen dargestellt. Es gibt eine Version, die nur von anderen Nutzern sichtbar

ist, mit denen ein Match stattgefunden hat, und eine Version, die über die Bottom-Navigation-Bar erreichbar werden kann. Die Letztere wird in Abbildung 33 dargestellt und unterscheidet sich von der Version für andere Nutzer darin, dass Profil- und Hintergrundbild bearbeitet werden können.

Das Farbschema und das Design wurden an die bisherigen Seiten angepasst. Um die Oberfläche simpel und selbsterklärend zu halten, wird jede dargestellte Information mit einem passenden Icon und einem Hinweis versehen (siehe Abbildung 33a). Die Icons zum Bearbeiten der Bilder sind wie auch in vielen anderen Apps platziert und designt. Sie öffnen die systemeigene Bildergalerie des Smartphones um den User aus einem bekannten Umfeld Bilder auswählen lassen zu können.

Beim initialen Öffnen einer Profilseite sollen Namen, Profilbild und ein Hintergrundbild ins Auge springen. Sie stellen die ersten Informationen dar, die dem Betrachter wichtig sind, weshalb sie wie in Abbildung 33a deutlich sichtbar ist beim Öffnen mehr als die Hälfte des Bildschirms einnehmen. Anschließend wird der Fokus auf detailliertere Informationen gerichtet. Auf der Profilseite von StreamSwipe wird hierfür heruntergescrollt um den Block mit den Profildaten sehen zu können. Bei dieser Aktion blendet eine Animation das Profilbild aus und verschmälert das Hintergrundbild. Der Benutzername wird ebenfalls aus dem Fokus gezogen, bleibt aber wie in Abbildung 33b zu sehen mit dem verbleibenden Hintergrundbildausschnitt erhalten. Dies hilft dem Betrachter unterbewusst bei dem Fokuswechsel und schafft ein modernes, responsives Feedback bei der User Experience.

Um das durchgängig schlichte Design der App zu erhalten ist der Zugang zu den Einstellungen ausschließlich auf der Profilseite zu finden. Hierfür ist im rechten oberen Bildschirmbereich das repräsentative Icon. Der hierdurch erreichbare Bildschirm (Abbildung 33c) ist gleich aufgebaut wie die Informationeneingabe nachdem ein neuer Account erstellt wurde (Abbildungen 29c und 29d). Die dort angegebenen Informationen können hier wieder angepasst werden.

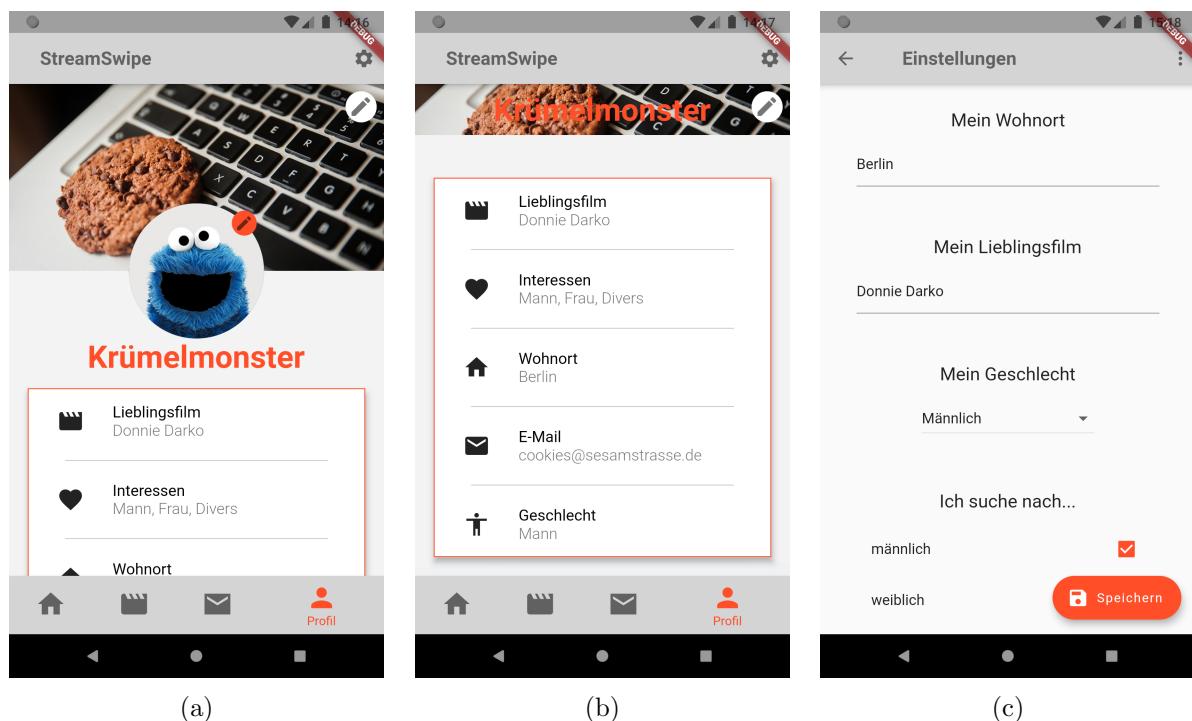


Abbildung 33: Profilseite wie sie für den Nutzer selbst angezeigt wird (a) im normalen Zustand und (b) nach vollständigem Einklappen des Profilkopfes durch eine Animation während dem Herunterscrollen. Mit den von hier aus erreichbaren Einstellungen (c) können die anfänglich gegebenen Profilangaben abgepasst werden.

## 7 Probleme

In der Planungsphase eines Projektes werden oft große Meilensteine gesetzt. Sobald das Thema bekannt ist, werden durch Brainstorming und andere Methoden der Ideenfindung in kurzer Zeit viele Ziele gesteckt und ein in der Theorie fertiges Projekt ausgearbeitet. Eine Begrenzung wird nur durch die Fantasie der Beteiligten gezogen, die jedoch später in der Umsetzung schnell erreicht wird. Aber auch bei realistischen Zielen wird oft eine sehr spezielle Vorstellung angestrebt, die mit den gegebenen oder gewählten Mittel nur bedingt umsetzbar ist. Neben den Arbeitsmitteln zählen aber auch Zeit und Geld zu den einschränkenden Ressourcen.

Wie unzählige Projekte vor uns, konnten wir ebenfalls nicht alle zu Beginn gesteckten Ziele zu unserer vollsten Zufriedenheit abschließen. Auch wenn alle grundlegend wichtigen Kriterien erfüllt sind und funktionieren, gibt es auch nicht erreichte Ziele. Diese bestehen aus erkennbaren und nicht erkennbaren Schwächen in der App. Ersteres sind Unsauberkeiten, die der User bei der Benutzung in manchen Situationen bemerken kann. Hierzu zählen beispielsweise Designfehler, die bei der Programmierung nicht erkannt wurden. Nicht erkennbare Schwächen sind Ideen und Ziele, die im vorgegebenen Rahmen nicht mehr eingebaut werden konnten. Diese fallen nur dem Programmierer auf. In unserem Fall ist das Fehlen mancher Funktionen hauptsächlich auf Zeitmangel zurückzuführen. Bei einem zeitlich begrenzten Bearbeitungsrahmen, während dem noch einige andere Projekte, Vorlesungen, Prüfungen und eine Arbeitsstelle in Vollzeit belegt werden, können nicht alle beabsichtigten Features eingebaut werden. Alle geplanten Erweiterungen wurden jedoch bereits durchdacht und werden nach Abgabe der schriftlichen Arbeit implementiert, aber hierzu mehr in Kapitel 10.

Da StreamSwipe in Deutschland entwickelt und programmiert wurde, liegt es nahe Deutsch als In-App-Sprache zu verwenden. Außer den auf den Oberflächen angezeigten Texten müssen auch Fehlermeldungen, Semantiken und Autovervollständigungen angepasst werden. Wie auf den Screenshots im ganzen Kapitel 6.2 zu sehen ist, wird dies in der App konsequent durchgesetzt. Die Filmdatenbank, aus der die dargestellten Informationen erhalten werden, liefert diese jedoch nur auf Englisch, was an der Handlungsausgabe in Abbildung 31b zu sehen ist. Alle Datenbanken, die in Kapitel 4.4 betrachtet werden, bieten ihre Informationen nur in einer Sprache an. Eine Lösung dieses Problems wird in Kapitel 9 beschrieben.

Bei einer sauberen Implementierung werden die Elemente auf den Bildschirmen in Abhängigkeit der Gesamtgröße des Screens angeordnet, sodass sie sich auf unterschiedlichen Geräten entsprechend anordnen und ihre Größe anpassen können. Man spricht hierbei von responsivem Design, auf welches bei StreamSwipe ebenfalls Wert gelegt wurde. Je nach Verhältnis von Breite und Höhe der Bildschirmgrößen unterschiedlicher Geräte können so in Grenzfällen jedoch ungewollte Verzerrungen auftreten. Das Aussehen einer Benutzeroberfläche kann außerdem auch von dem verwendeten Betriebssystem abhängen, da unterschiedliche Generationen unterschiedliche Standards verwenden. Durch die begrenzte Entwicklungszeit und den limitierten Zugang zu Ressourcen konnte die App nur auf einer kleinen Anzahl von Endgeräten getestet und angepasst werden. Es kann somit leider keine Garantie für eine saubere Darstellung auf älteren Geräten gegeben werden. Längere Testphasen könnten dieses Problem minimieren, jedoch ist der Markt von Smartphones mittlerweile so unübersichtlich groß, dass es nahezu unmöglich ist jede Variation zu testen.

Das wahrscheinlich größte Manko ist das Ausbleiben der Veröffentlichung der App am Abgabetermin der schriftlichen Arbeit. Ursprünglich geplant war eine Veröffentlichung in Google Play für Androidgeräte und im App Store für iPhones, weshalb wie in Kapitel ?? beschrieben Flutter als Framework verwendet wurde. Bei einer Veröffentlichung in Google Play wird die App vor Release einer umfangreichen Überprüfung unterzogen, was voraussichtlich eine Woche und in

manchen Fällen auch länger dauern kann [19]. Für eine Veröffentlichung im App Store ist eine Mitgliedschaft im Apple Developer Programm notwendig, die jährlich \$ 99 kostet [20]. Bei einer App ohne Monetarisierungsprogramm ist dies keine angemessene Investition.

Wie in Kapitel 3.2 bereits erwähnt, werden die Verteilung der Film-IDs und die Auswertung der Filmbewertungen auf einem Server ausgeführt. Dieser Server besteht zur Zeit aus einem Raspberry Pi. Für den Gebrauch während der Entwicklungsphase ist die somit erreichte Rechenkapazität völlig ausreichend, da nur maximal drei Personen gleichzeitig darauf zugegriffen haben. Sollte die App jedoch veröffentlicht werden, werden die Nutzerzahlen unvorhersehbar ansteigen, sodass nicht vorausgesagt werden kann, ab wann der Server in diesem Aufbau überlastet sein wird. Um den Benutzern eine positive User Experience bieten zu können und eventuelle Hardwareschäden an dem Raspberry Pi zu vermeiden, muss mit der Veröffentlichung der App gewartet werden bis eine leistungsstärkere Lösung gefunden wurde.

## 8 Anwendbarkeit

anwendbarkeit: - was will der user? was machen die user wirklich daraus - wie sieht das technisch gesehen aus? - wie sind wir auf dem markt vertreten?

## 9 Ausblick

Die Entwicklung und Programmierung der App war ein wichtiger und großer Schritt, jedoch ist er nicht der letzte. Für die nun kommende Zeit stehen bereits einige Aufgaben fest um das Projekt StreamSwipe aufrechtzuerhalten.

Wie bereits in Kapitel 7 erwähnt, konnten nicht alle gewünschten Features eingebaut werden. Sie waren während der Entwicklungsphase entweder von zu geringer Priorität, oder sind erst im späteren Stadium aufgekommen. Beispielsweise werden die Filmdaten bisher nur auf Englisch erhalten. In einer sonst deutschsprachigen App ist diese Umstellung jedoch schon in Planung. Es existieren deutsche Filmdatenbanken wie beispielsweise OFDb, aus denen die übersetzten Daten erhalten werden können. Diese ist leider nicht so umfangreich wie die bisher genutzte Datenbank, sodass dann von beiden parallel Daten erhalten und abgeglichen werden um die User Experience nicht zu mindern. In diesem Zuge ist auch eine optionale In-App-Sprache geplant, die der Benutzer auswählen kann, da nun die Informationen auf Deutsch und auf Englisch vorliegen. Um die Sprache der App ändern zu können, müssen alle Textelemente durch Variablen ersetzt werden, die in einer zentralen Tabelle beschrieben werden. Für eine andere Sprache werden diese Variablen dann mit den übersetzten Formulierungen überschrieben. Ist die App angepasst und der Datenfluss der beiden Datenbanken synchronisiert, können auch mit weniger Aufwand auch andere Sprachen in das System aufgenommen werden.

Ein Update, bei dem Filmempfehlungen eingeführt werden, ist bereits geplant. Hierbei werden auf Basis der berechneten Filmpräferenz und den Überschneidungen innerhalb eines Matches eine Liste mit Filmvorschlägen generiert. Dieses Feature ist von großer Bedeutung wenn man bedenkt wie viel Zeit in die Suche eines geeigneten Filmes gesteckt wird.

Auch das Ressourcenproblem des Servers kann in absehbarer Zeit behoben werden. Da die Skripte auf dem Raspberry Pi bereits geschrieben sind, ist lediglich eine leistungsstärkere Hardware nötig um einen stabilen Server zu realisieren. Solange die Benutzerzahlen und somit auch die benötigte Leistung überwacht werden, kann frühzeitig ohne Datenverlust oder Performanceeinbußen auf ein größeres System umgestellt werden.

Ist die App dann für die Öffentlichkeit zugänglich, lohnt es sich ein Feedbacksystem zu nutzen, durch welches die User Lob und Kritik äußern können. Oft treten kleine Bugs nur in sehr speziellen Situationen auf, die während der Testphase nicht bedacht oder nicht realisiert werden. Der Wartungsaufwand nach Release einer Software wird oft im Verhältnis zum Entwicklungsaufwand als gering erachtet, sollte aber nicht unterschätzt werden. Werbung spielt einen entscheidenden Faktor in der Kundengewinnung. Mithilfe gezielter Produktplatzierungen, möglichst großflächiger Marketingstrategien und zufriedenen Bestandskunden kann innerhalb von wenigen Wochen ein beachtlicher Kundenstamm aufgebaut werden. Timing ist in diesem Fall sehr wichtig, da die angeworbenen Kunden möglichst gleichzeitig mit der App bekannt gemacht werden sollten. StreamSwipe basiert auf einem möglichst flächendeckenden Kundenkreis, da nur lokal gematcht wird und ein User ohne Matches nicht lange gehalten werden kann. Entsprechend ergibt es Sinn gezielte Werbung in einer lokalen Umgebung einzusetzen und lieber eine hohe Benutzerdichte, als eine große Reichweite aufzubauen. Wird die App in diesem Umfeld genutzt und somit Matches generiert, verbreitet sie sich automatisch auch in umliegende Städte und beginnt so zu wachsen. Mit dem Benutzerradius sollte auch der Radius der geschalteten Werbung sukzessiv vergrößert werden.

## 10 Fazit

fazit: - welche ziele haben wir erreicht - gutes/schlechtes - nicht perfekt (kleine bugs), aber wann ist man wirklich fertig??? Nichtmal Gott ist mit seiner Schöpfung wirklich fertig -

## A Verfasser einzelner Abschnitte

Kapitel / Abschnitt	Verfasser
1. Einleitung	Vincent Schreck
1.1 Motivation	Vincent Schreck
1.2 Methode	Vincent Schreck
2. Theoretische Grundlagen	Leon Gieringer & Robin Meckler
2.1 Netzwerkprotokolle	Robin Meckler
2.2 JavaScript	Robin Meckler
3.3 NodeJS	Robin Meckler
2.4 Representational State Transfer - Application Programming Interface	Robin Meckler
2.5 NoSQL-Datenbank	Robin Meckler
2.6 Firebase	Leon Gieringer
2.7 Anwendungsentwicklung für mobile Endgeräte	Leon Gieringer
2.8 Frameworks zur mobile, plattformübergreifenden Entwicklung	Leon Gieringer
2.9 Recommender Systems	Leon Gieringer
3. Konzept	Vincent Schreck
4. Auswahl geeigneter Technologie	
4.1 Server	
4.2 Datenbank	
4.3 Kommunikationsschnittstelle	
5. Backend-Implementierung	Robin Meckler
5.1 Server	Robin Meckler
5.2 Datenbank	Robin Meckler
5.3 Kommunikationsschnittstelle	Robin Meckler
5.4 Filmdatenbank	Robin Meckler
6. Funktionen/Komponenten	
6.1 Swipe/Aussuchen/Voting	
6.2 Matches/Chat	Leon Gieringer
6.3 Filmvorschläge	
6.4 Gespeicherte Filme	
6.6 Barrierefreiheit	Vincent Schreck
7. Benutzeroberfläche	Vincent Schreck
7.1 Aspekte von Benutzeroberflächen	Vincent Schreck
7.2 Oberflächen von StreamSwipe	Vincent Schreck
9. Probleme	
10. Anwendbarkeit	
11. 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] Facebook Inc. (2021) React Native documentation. [Online] Verfügbar: <https://reactnative.dev/docs/getting-started>, letzter Zugriff: 13. April 2021
- [4] Facebook Inc. (2021) React documentation. [Online] Verfügbar: <https://reactjs.org/docs/getting-started.html>, letzter Zugriff: 13. April 2021
- [5] Flutter (2021) Flutter architectural overview [Online] Verfügbar: <https://flutter.dev/docs/resources/architectural-overview>, letzter Zugriff: 04. März 2021
- [6] Johnson R. E. & Foote B. "Designing Reusable Classes." Journal of ObjectOriented Programming 1, 2 (June/July 1988). Page 22-35.
- [7] Majchrzak TA, Ernsting J, Kuchen H (2015) Achieving business practicability of model-driven crossplatform apps. OJIS 2(2):3–14
- [8] Cisco (2020) Cisco Annual Internet Report (2018–2023) White Paper [Online] Verfügbar: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-white-paper-c11-741490.html>
- [9] Charland, A. & Leroux, B. (2011). Mobile application development: Web vs. native. Communications of the ACM, 54(5):49–53.
- [10] Lachgar, M., & Abdelmounaim, A. (2017). Decision Framework for Mobile Development Methods. International Journal of Advanced Computer Science and Applications, 8.
- [11] Biørn-Hansen, A., Rieger, C., Grønli, TM. et al. (2020) An empirical investigation of performance overhead in cross-platform mobile development frameworks. Empir Software Eng 25, 2997–3040. <https://doi.org/10.1007/s10664-020-09827-6>
- [12] Stahl T, Volter M (2006) Model-driven software development. Wiley, Chichester
- [13] Firebase Inc. (2021) Firebase documentation. [Online] Verfügbar: <https://firebase.google.com/docs>, letzter Zugriff: 25. April 2021
- [14] Meinungsforschungsinstituts Civey (2020): <https://www.presseportal.de/pm/145489/4627304>, letzter Zugriff: 13. Mai 2021
- [15] Splendid Research (2017): <https://www.springerprofessional.de/konsumforschung/marketingstrategie/konsumenten-auf-der-serien-welle/15146374>, letzter Zugriff: 14. Mai 2021
- [16] Statistisches Bundesamt (2020): <https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Behinderte-Menschen/Tabellen/schwerbehinderte-alter-geschlecht-quote.html;jsessionid=885260788D4FFC7F670576B72E5089F4.live741>, letzter Zugriff: 17. April 2021
- [17] Behindertengleichstellungsgesetz (2002): <https://www.gesetze-im-internet.de/bgg/BGG.pdf>, letzter Zugriff: 19. April 2021

- [18] Institut für Demoskopie Allensbach (2019): *Untersuchung zum Sehbewusstsein der Deutschen*, file:///C:/Users/Vincent/AppData/Local/Temp/ZVA\_Brillenstudie\_2019-1.pdf, letzter Zugriff: 11. Mai 2021
- [19] Google Support (2021): *App veröffentlichen*, <https://support.google.com/googleplay/android-developer/answer/9859751?hl=de>, letzter Zugriff: 14. Mai 2021
- [20] Apple Support (2021): *Infos zum Apple Developer Program*, <https://developer.apple.com/de/support/compare-memberships/>, letzter Zugriff: 14. Mai 2021