HAW HAMBURG

BACHELOR THESIS
Maximilian Lutz

Backend-Anwendungen im Zeitalter von Serverless: Ein Einblick in Backend-as-a-Service am Beispiel der Entwicklung einer mobilen Applikation

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Maximilian Lutz

Backend-Anwendungen im Zeitalter von Serverless: Ein Einblick in Backend-as-a-Service am Beispiel der Entwicklung einer mobilen Applikation

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Wirtschaftsinformatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 22. November 2022

Maximilian Lutz

Thema der Arbeit

Backend-Anwendungen im Zeitalter von Serverless: Ein Einblick in Backend-as-a-Service am Beispiel der Entwicklung einer mobilen Applikation

Stichworte

Backend, BaaS, Serverless, Cloud Computing, Anwendung, mobile App, Firebase, AWS, Amplify, Parse, Back4App

Kurzzusammenfassung

Backend as a Service ist ein Teil des Serverless Konzeptes und bietet den Entwicklern die Möglichkeit, standardisierte Dienste zur Entwicklung eines vollständigen Backends zu verwenden. Evender möchte als Start-up die Suche nach Veranstaltungen mittels einer mobilen App neu interpretieren. Diese Arbeit untersucht, inwieweit sich verschiedene Anbieter eines Backends-as-a-Service zur Umsetzung einer Backendanwendung für diesen Anwendungsfall eignen.

Maximilian Lutz

Title of Thesis

Backend-Applications in the age of serverless: An Introduction to Backend-as-a-Service using the example of the implementation of a mobile application

Keywords

Backend, BaaS, Serverless, Cloud Computing, Application, mobile Application, Firebase, AWS, Amplify, Parse, Back4App

Abstract

Backend as a Service is an important part of the Serverless concept and offers the possibility to use standardized services to implement an entire backend. Evender as a young start up wants to reinterpret the search for events with a mobile application. This thesis

examines to what extent different providers of a Backend-as-a-Service suit the needs of Evender for this specific use case.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BaaS Backend-as-a-Service.

FaaS Functions-as-a-Service.

IaaS Infrastructure-as-a-Service.

PaaS Platform-as-a-Service.

SaaS Software-as-a-Service.

1 Einleitung

Das Serverless Konzept ist eine vergleichsweise neue Entwicklung im Bereich des Cloud Computings. Damit wird es den Entwicklern ermöglicht, sich auf das Implementieren einer Anwendung zu konzentrieren, anstatt einen großen Teil der zur Verfügung stehenden Zeit für das Aufsetzen und die Wartung der Infrastruktur aufzuwenden. Auch der Trend der Verwendung von Microservices trägt dazu bei, dass Serverless immer mehr an Beachtung gewinnt, da Serverless als nächste Evolutionsstufe von Microservice Architekturen gesehen wird. Backend-as-a-Service (BaaS) als Teil von Serverless bietet den Entwicklern die Möglichkeit, standardisierte Dienste zur Umsetzung eines Backends zu verwenden, was zu einer geringeren time-to-market führt. So können neue Features oder auch ganze Anwendungen schneller bereitgestellt werden, was BaaS gerade im Bereich der mobilen Apps und Webapps populär macht. Für 2022 wird für den BaaS Markt ein Marktvolumen von 2,4 Milliarden Dollar vorausgesagt und bis 2032 wird das Marktvolumen auf 22 Milliarden Dollar anwachsen. Grund dafür ist die steigende Adaption von Mobilgeräten wie Smartphones, Tablets und Wearables wie Smartwatches [Saha, 2022]. Damit einher geht eine steigende Anzahl an mobilen Apps und in genau diesem Bereich können Entwickler von den Vorteilen der Verwendung eines BaaS profitieren.

1.1 Motivation

Das Team von Evender ist ein junges Gründungsteam, das mit einer mobilen Anwendung das Suchen und Finden jeder Art von Veranstaltungen nachhaltig verändern möchte. Mit Hilfe dieser Applikation soll vorrangig die bisher oft sehr zeitaufwändige Recherche bei verschiedenen Anbietern vereinfacht werden: Events sollen aus Quellystemen wie bspw. Facebook oder Eventbrite in einer App konsolidiert und dem Nutzer in Form eines Swipe Stacks präsentiert werden. Die Entwicklung soll dabei möglichst schnell, einfach, flexibel und kostengünstig gestaltet werden. Auch automatische Skalierbarkeit und ein geringer Aufwand beim Erstellen und Warten von Infrastruktur ist wichtig. All dies kann erreicht

werden, indem man sich statt eines vollständig eigenhändig entwickelten Backends eines BaaS bedient. BaaS ist dabei ein Cloud Service Modell, bei dem Entwickler die meisten Hintergrundaspekte einer Web- oder Mobilanwendung zu einem Anbieter auslagern, sodass hauptsächlich nur das Frontend entwickelt und gewartet werden muss. Dieses Modell wird dabei mit in den Bereich des Serverless Computings eingeordnet. Diese Ausgangslage bildet die Grundlage dieser Arbeit: Für ein Start-up, das eine mobile App entwickeln möchte, stellt sich die Frage nach der Umsetzung dieser Anwendung. In einem schnelllebigen Markt für mobile Apps kann eine kurze Entwicklungszeit von Vorteil sein, genauso wie der geringere Aufwand in Bezug auf die Infrastruktur oder geringere Kosten. Somit kann die Verwendung eines BaaS der richtige Ansatz für diesen Anwendungsfall sein. Auch die Auswahl eines Anbieters oder eines Tools im BaaS Bereich kann aufgrund der Vielzahl an Anbietern sehr aufwändig sein. Jeder Anbieter offeriert unterschiedliche Funktionalitäten, Rahmenbedingungen und Architekturen. Die Eignung verschiedener Anbieter soll nun im Rahmen dieser Arbeit untersucht werden.

1.2 Fragestellung und Methodik

Das Ziel dieser Arbeit ist die Evaluation verschiedener, noch auszuwählender BaaS Anbieter. Die zentrale Forschungsfrage lautet dabei: Inwieweit eignen sich verschiedene BaaS Anbieter zur Entwicklung eines Backends für die mobile Evender App? Auch die Gemeinsamkeiten und Unterschiede der Anbieter sollen so herausgearbeitet werden. Um zu einem Ergebnis zu gelangen wird zuerst die Anwendung an sich erarbeitet. Mittels verschiedener Methoden und Techniken wie Brainstorming, User Storys und Wireframes werden die Anforderungen an die mobile App und resultierend daraus für das Backend erhoben und spezifiziert. Als Ergebnis der Spezifikation wird die Architektur der Backendanwendung konzipiert, die es dann bei den ausgewählten BaaS Anbietern umzusetzen gilt. Zur Auswahl der Anbieter werden Kriterien wie die Auswahl der angebotenen Dienste oder die Möglichkeiten zur Umsetzung aller Anforderungen eingesetzt. Für den Vergleich der Anbieter wird ein eigener Kriterienkatalog erarbeitet: Dieser basiert auf verschiedenen Merkmalen der Softwarequalität aus der ISO Norm 25010 und weiteren für diesen Anwendungsfall relevanten Merkmalen der Backendanwendungen. Das Backend wird bei allen ausgewählten Anbietern implementiert, um Informationen für die verschiedenen Kriterien des Kataloges zu erarbeiten. Der Code dieser implementierten Anwendungen befindet sich im elektronischen Anhang der Arbeit. Abschließend erfolgt der Vergleich in einer Gegenüberstellung der Anbieter in Bezug auf die Kriterien.

1.3 Struktur der Thesis

Zuerst wird im zweiten Kapitel auf die nötigen Grundlagen zum Verständnis dieser Arbeit eingegangen. Es werden die Konzepte des Cloud Computings und von Serverless erklärt, wobei speziell auf das Functions-as-a-Service (FaaS) und das BaaS Konzept eingegangen wird. Zudem werden bereits erste Vor- und Nachteile der Verwendung eines BaaS dargestellt. Im darauffolgenden dritten Kapitel folgt die Analyse und Spezifikation. Hier wird der konkrete Anwendungsfall erläutert und analysiert sowie die Anforderungen an die Anwendung erhoben. Diese Anforderungen werden für das Backend konkretisiert und spezifiziert. Im vierten Kapitel der Konzeption wird das Backend entworfen: Es werden die Komponenten und die Zielarchitektur dargestellt. Außerdem werden die Anbieter für den durchzuführenden Vergleich der BaaS Systeme ausgewählt und es wird ein Kriterienkatalog entwickelt, aus Basis dessen der Vergleich dann letztendlich bewertet werden kann. Danach wird im fünften Kapitel der Vergleich der implementierten Backendanwendungen dargestellt. Es wird auf die Kriterien des Kataloges in Bezug auf die Anwendungen eingegangen, sodass Gemeinsamkeiten und Unterschiede der Anbieter ersichtlich werden. Abschließend wird der Vergleich im Kapitel der Evaluation bewertet. Die Ergebnisse werden zusammengefasst und interpretiert und es wird Bezug genommen auf die oben dargestellte Fragestellung. Dabei wird auch auf Limitationen eingegangen. Im Fazit werden die Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick über Weiterentwicklungsmöglichkeiten gegeben.

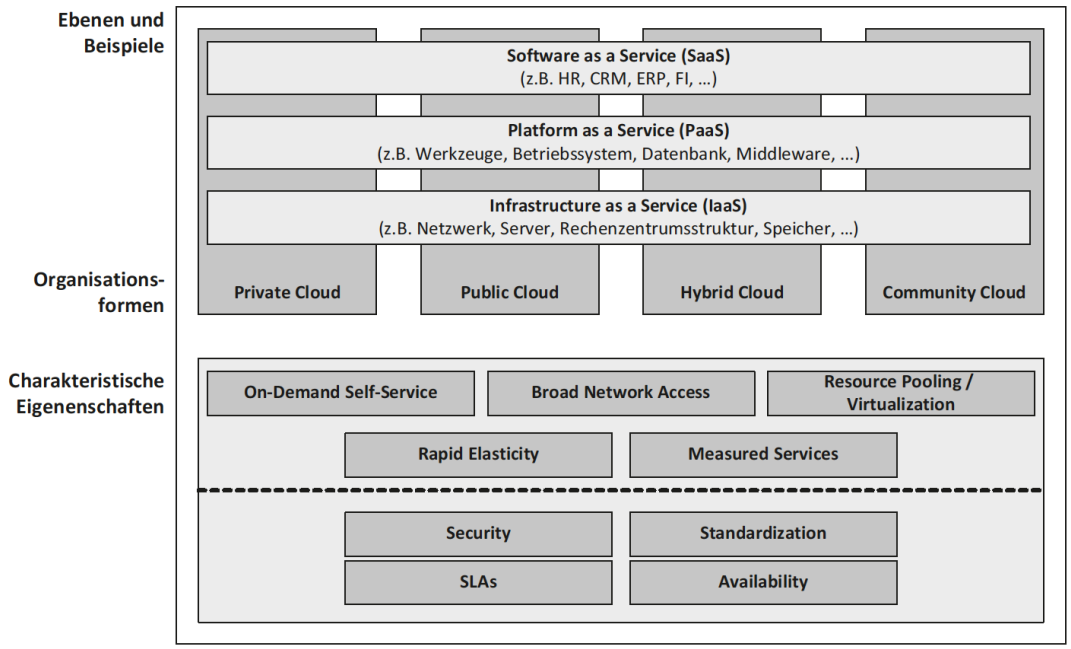
2 Grundlagen

In diesem Kapitel werden die Grundlagen der Thesis dargestellt. Es soll ein grundlegendes Verständnis für die weiteren Teile aufgebaut werden. Der Begriff des Cloud Computings bildet dabei die Grundlage für die weiterführenden behandelten Themen wie Serverless, FaaS und BaaS. Für den Teil des BaaS wird im Besonderen auf die mit der Verwendung einhergehenden Vor- und Nachteile eingegangen.

2.1 Einführung in das Cloud Computing

Die immer weiter zunehmende Digitalisierung der Gesellschaft und Wirtschaft erfordert, dass auch Kapazitäten bezüglich der Rechenleistung und des Speichers in Informationssystemen steigen. Eine Lösung für dieses Problem kann Cloud Comupting sein. Benötigte Ressourcen stehen auf Abruf bereit und Kunden zahlen nur für die Dienstleistungen, die sie auch tatsächlich in Anspruch genommen haben. Somit bietet das Cloud Computing Unternehmen und anderen Akteuren unserer Gesellschaft eine enorme Flexibilität. Der Begriff des Cloud Computings wurde erstmals von Ramnath K. Chellappa 1997 auf einer Konferenz verwendet, seitdem wurde bis heute keine eindeutige Definition festgelegt [Reinheimer, 2018].

Cloud Computing beschreibt dabei ein informationstechnisches Bereitstellungsmodell, welches es ermöglicht, Ressourcen wie bspw. Server oder Anwendungen mittels des Internets als Dienst in Anspruch zu nehmen [Krzmar, 2015]. Das National Institute of Standards and Technology aus den USA identifiziert zur Definition des Cloud Computings vier Bereitstellungsmodelle, drei Servicemodelle und fünf Charakteristiken, die in Abbildung 2.1 dargestellt sind [Mell u. a., 2011]. Weiterhin lassen sich in der Abbildung bereits weitere Eigenschaften identifizieren: Security, Standardization, Service Level Agreements und Availability. Auf diese Punkte wird im weiteren Verlauf der Arbeit noch näher eingegangen.



Quelle: [Reinheimer, 2018]

Abbildung 2.1: Charakteristiken und Service- und Bereitstellungsmodelle des Cloud Computings

Abbildung 2.1 zeigt die notwendigen Eigenschaften des Cloud Comuptings: On-Demand Self Service, Broad Network Access, Resource Pooling, Rapid Elasticity und Measured Services. On-Demand Self Service bedeutet, dass Nutzer eines Dienstes Computing Ressourcen wie zum Beispiel Server oder Speicher je nach Bedarf anfragen können, sodass diese Ressourcen umgehend automatisch vom Anbieter bereitgestellt werden. Diese Ressourcen bzw. die Bereitstellung dieser wird mittels standardisierter Technologien über das Internet erreicht (Broad Network Access). Weiterhin werden die Ressourcen vom Anbieter der Dienste in einem Pool verwaltet, somit können mehrere Kunden gleichzeitig bedient werden und die Ressourcen werden dynamisch je nach Bedarf zugewiesen. Der Kunde besitzt dabei keinerlei Kontrolle darüber, wo genau die angefragten Ressourcen bereitgestellt werden (Resource pooling). Rapid elasticity bedeutet, dass Ressourcen zugewiesen und wieder freigegeben werden können und so je nach Bedarf skaliert werden können. Für den Nutzer eines solchen Cloud Dienstes hat es oft den Anschein, als wäre eine unendliche Menge an Ressourcen verfügbar. Außerdem wird die Nutzung der Ressourcen durchgehend auf Basis verschiedener Metriken kontrolliert und optimiert, sodass

eine hohe Transparenz für den Nutzer gewährleistet ist (Measured service) [Mell u. a., 2011].

Die bereits erwähnten vier Bereitstellungsmodelle sind Private Cloud, Community Cloud, Public Cloud und Hybrid Cloud.

- Public Cloud: Der Begriff Public Cloud beschreibt das Betreiben einer Cloud-Infrastruktur für die Öffentlichkeit, ein unabhängiger Anbieter stellt dem Nutzer verschiedene Dienste in seiner Umgebung zur Verfügung.
- Private Cloud: Im Gegensatz zur Public Cloud steht die private Cloud, bei der eine Cloud-Infrastruktur nicht öffentlich zur Verfügung steht, sondern einem einzelnen Unternehmen gehört und auch nur von diesem genutzt wird. Das Unternehmen besitzt dabei die absolute Kontrolle über die Infrastruktur.
- Hybrid Cloud: Eine hybride Cloud stellt eine Mischung der ersten zwei Bereitstellungsmodelle dar. So wird es einem Unternehmen ermöglicht, verschiedenen Anwendungen auf öffentlicher oder privater Infrastruktur zu betreiben und die Vorteile beider Modelle zu nutzen. Dieses Modell geht mit erhöhter Komplexität in Bezug auf bspw. das Überwachen oder Bereitstellen von Anwendungen einher.
- Community Cloud: In einer Community Cloud wird die bereitgestellte Infrastruktur exklusiv von einer Gruppe an Nutzern geteilt, die ähnliche Interessen aufweisen.

[Stanoevska u. a., 2009], [Mell u. a., 2011]

Außerdem werden in Abbildung 2.1 mehrere Service Modelle unterschieden, die sich bezüglich ihres Abstraktionsgrades einteilen lassen. Das Modell mit der geringsten Abstraktion ist Infrastructure-as-a-Service (IaaS). In diesem Modell wird dem Nutzer nur eine physikalische Infrastruktur bereitgestellt. Diese Infrastruktur kann bspw. aus Speicher, Computing- oder Netzwerkressourcen bestehen. Die Ressourcen können dann mittels Virtualisierung nach Bedarf zugeteilt und bereitgestellt werden [Vaquero u. a., 2008].
Die nächsthöhere Abstraktionsschicht beinhaltet das Platform-as-a-Service (PaaS) Modell. Dieses Modell richtet sich vor allem an Anwendungsentwickler: Der Cloud Anbieter stellt Umgebungen zur Verfügung, in denen entwickelte Software laufen kann. Der Anbieter kümmert sich darum, die entsprechende benötigte Infrastruktur bereitzustellen [Reinheimer, 2018].
Das Modell mit der höchsten Abstraktion heißt Software-as-a-Service (SaaS). Mit diesem Modell wird standardisierte Software bereitgestellt, die sich an Endnutzer richtet. Die

Software ist mittels des Internets erreichbar, somit benötigen Nutzer nur einen Zugang zum Internet und einen Browser um die Software verwenden zu können. Der Anbieter der Software kümmert sich um den Betrieb und die Wartung, aber dadurch, dass die Software standardisiert ist, sind auch die Anpassungsmöglichkeiten eingeschränkt [Buxmann u. a., 2008].

Durch die Verwendung von Dienstleistungen in der Cloud ergeben sich viele Vorteile. Aus finanzieller Sicht ist es nicht mehr notwendig, teure Hard- und Software anzuschaffen, die Kosten basieren auf dem Pay as you go Prinzip: Kunden bezahlen nur für diejenigen Dienstleistungen, die sie auch tatsächlich in Anspruch genommen haben. Auch kleinere Unternehmen, die bspw. keine eigene Infrastruktur besitzen, profitieren so vom Cloud Computing. Auf Seite der Anbieter von Cloud Services führt das Pooling der Ressourcen und das dynamische Zuweisen dieser Ressourcen zu einer besseren Auslastung der Infrastruktur was wiederum zu sinkenden Kosten für den Betrieb führt [Reinheimer, 2018]. Abgesehen von den finanziellen Vorteilen ergeben sich auch operative Vorteile: Die in Anspruch genommenen Ressourcen skalieren automatisch und können vom Kunden je nach Bedarf angepasst werden. Somit können sich Unternehmen verstärkt auf die eigene Strategie konzentrieren indem bspw. Geschäftsprozesse verbessert oder neue Geschäftsbereiche erschlossen werden. Die Verwendung von Cloud Services birgt jedoch auch Risiken. Wenn Anwendungen in eine Cloud migriert werden, schwindet der direkte Einfluss eines Unternehmens auf die eigene Anwendung. Damit einher geht auch ein erhöhter Integrations- und Migrationsaufwand [Reinheimer, 2018].

Die oben genannten Service Modelle bilden die Grundlage für das Verständnis der Dienste im Cloud Bereich. „Der technische Fortschritt ermöglicht hier eine feingranulare Konzeption standardisierter Dienstleistungen, bei denen es von immer geringerer Bedeutung ist, von wem und wo diese Dienstleistungen erbracht werden. Diese Entwicklung wird als Everything as a service (XaaS) bezeichnet.“[Krzmar, 2015] Krzmar verweist hier auf aktuelle Entwicklungen im Cloud Computing Bereich wie bspw. das Serverless Konzept. Dienstleistungen werden zunehmend ausgelagert und Entwickler können diese ohne großen Aufwand in Anspruch nehmen. Im Folgenden wird auf weitere Vertreter dieses Paradigmas aus dem Bereich Serverless genauer eingegangen.

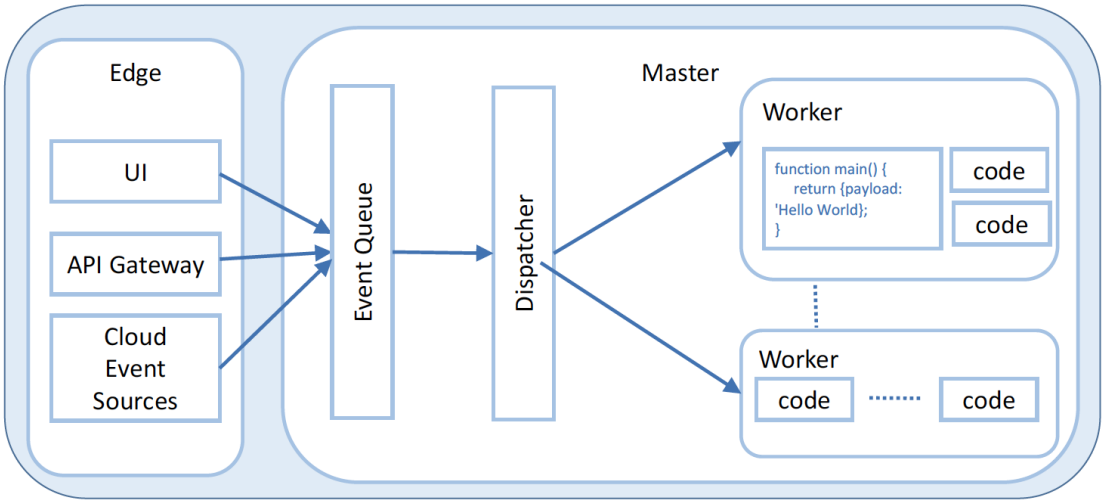
2.2 Serverless

Unter dem Begriff Serverless wird eine Reihe von Techniken und Technologien verstanden, die sich in zwei Bereiche aufteilen: FaaS und BaaS. Somit ist FaaS kein Synonym für Serverless, sondern es geht um die verwalteten Dienste [Friedrichsen, 2022]. Auf die beiden Bereiche wird im Folgenden noch näher eingegangen, es soll nun jedoch zuerst ein Verständnis für das Serverless Konzept vermittelt werden.
Serverless ist ein vergleichsweise neuer Bereich innerhalb des Cloud Computings. Dienste werden über das Internet bereitgestellt und dem Entwickler wird dabei das Meiste in Bezug auf die Infrastruktur und das Verwalten der Infrastruktur abgenommen. Auch die Skalierung der Dienste ist vollständig automatisiert, die Bezahlung erfolgt nach dem Pay as you go Prinzip. Wenn die Dienste nicht aktiv sind fallen auch keine Kosten an. In diesem Zusammenhang fällt oft die Aussage „Run Code, not Servers“[Koschel u. a., 2021], denn dies ist die grundlegende Idee von Serverless: Entwickler können sich auf die Geschäftslogik konzentrieren anstatt Zeit mit dem Warten oder Aufsetzen der Infrastruktur zu verschwenden. Server werden zwar weiterhin benötigt, jedoch wird den Entwicklern durch eine weitere Abstraktionsschicht die Möglichkeit geboten, sich auf die reine Anwendungsentwicklung fokussieren zu können [Koschel u. a., 2021].
Dass das Serverless Paradigma aktuell so viel Beachtung findet liegt auch daran, dass Architekturen von Anwendungen jeder Art in den letzten Jahren zunehmend von Containern und Microservices geprägt waren. Jede weitere Schicht der Abstraktion von der eigentlichen Anwendung führte dazu, dass die Komponenten einer Architektur immer leichtgewichtiger wurden. Der Begriff Serverless wurde 2014 das erste Mal von Amazon auf der AWS re:invent Konferenz geprägt, als Amazon den hauseigenen Lambda Service vorstellte. Weitere Cloud Anbieter folgten mit der Zeit und stellten auf ihrer Seite entsprechende Serverless Dienste vor [Baldini u. a., 2017].

2.2.1 Functions-as-a-Service

FaaS beschreibt den Teil von Serverless, bei dem der Code für die Logik einer Anwendung mittels zustandsloser Funktionen in Containern ausgeführt wird. Diese Funktionen sind meist Event-basiert, sie werden also ausgeführt, sobald ein bestimmtes Event durch die Plattform registriert wird. Das Verwalten der Container und der Ausführung übernimmt dabei der Anbieter der FaaS Plattform, genauso wie das Allokieren der benötigten Ressourcen. Nach Ausführung der Funktion werden die entsprechenden Kapazitäten wieder

freigegeben [Roberts, 2018]. Auf Grund der Zustandslosigkeit der Funktionen wird, wenn ein Zustand gespeichert werden soll, eine Komponente für die Persistenz in der Architektur benötigt.
Die definierten Funktionen einer FaaS Plattform sind so lange inaktiv, bis sie durch ein Event ausgelöst werden. Die Plattform wartet dabei auf bestimmte, für diese Funktion definierte Events. Wenn die Funktion ausgeführt wurde werden die eingenommen Ressourcen wieder freigegeben, alternativ kann eine Funktion jedoch auch von der Plattform *warmgehalten* werden, falls in naher Zukunft ein weiteres Event registriert wird. So verringert sich die Zeit bis zur Ausführung der Funktion [Roberts und Chapin, 2017].
Quellen für solche Events können synchron und asynchron sein. Ein Beispiel für ein synchrones Event sind HTTP Anfragen. Events, die durch einen Objektspeicher ausgelöst werden (bspw. Upload einer Datei) oder auch geplante, wiederkehrende Events sind hingegen asynchron. Abbildung 2.2 zeigt dabei die grundlegende Architektur einer FaaS Plattform. Events aus verschiedenen Quellen wie bspw. eines API Gateways gelangen in eine Event Queue, die mittels eines Dispatchers abgearbeitet wird. Der Dispatcher weist den Events entsprechende Container zu, in denen dann der Code der entsprechenden Funktion ausgeführt wird.



Quelle: [Baldini u. a., 2017]

Abbildung 2.2: Functions-as-a-Service Architektur

2.2.2 Backend-as-a-Service

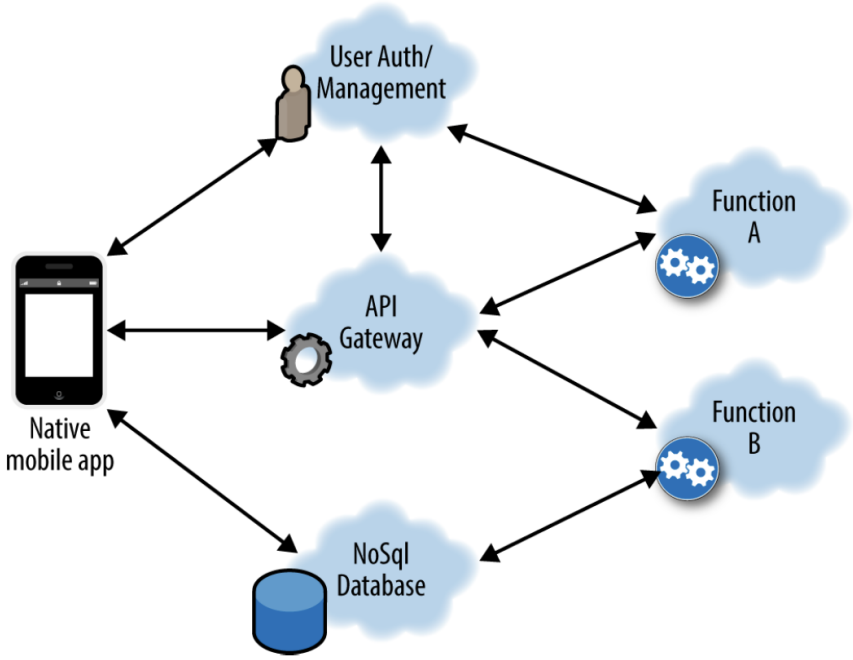
BaaS ist der zweite Teil von Serverless. Hier stehen nicht einzelne, kleinteilige Funktionen im Vordergrund, sondern es geht um Anwendungen, die vorrangig standardisierte Cloud-Dienste nutzen, um Geschäftslogik und den eigenen Zustand zu verwalten [Roberts, 2018]. Dadurch entstehen Anwendungen, die einen großen Teil der Logik in die Clients verlagern. Es wird ein ganzes Ökosystem an Diensten genutzt, um bspw. Daten zu speichern oder Nutzer zu authentifizieren. Auf diese Weise werden die Funktionen eines Backends nicht mehr zentral verwaltet, denn es gibt keine zentrale Komponente, die die Dienste orchestriert. Stattdessen wird mit dezentralen Diensten eine Strategie der Choreographie verfolgt [Roberts, 2018].
BaaS weist dabei viele Ähnlichkeiten mit dem SaaS Konzept auf, jedoch werden hier keine vorgefertigten Anwendungen verwendet sondern es geht darum, die eigene Anwendung in kleinere Komponenten zu zerlegen und diese Komponenten vollständig mittels externer Dienste zu implementieren [Roberts und Chapin, 2017]. Vor allem im Bereich der Entwicklung mobiler Anwendungen oder Webanwendungen ist BaaS populär, deshalb wird auch oft der Begriff Mobile-Backend-as-a-Service (mBaaS) verwendet. Ein Beispiel für eine Auslagerung einer Backend Funktion zu einem standardisierten Dienst ist die in vielen Anwendungen nötige Authentifizierung. Im Normalfall wäre es nötig, im Backend für die Authentifizierung eigene Logik zu implementieren. Da diese Logik in vielen Anwendungen jedoch sehr ähnlich funktioniert bietet es sich an, hierfür einen Dienst eines Cloud Anbieters zu verwenden. Somit entfällt die Notwendigkeit, eigene Logik zu implementieren [Roberts und Chapin, 2017]. Zur Integration der in Anspruch genommenen Dienste in die eigene Anwendung werden oft Software Development Kits vom Anbieter verwendet, die für viele verschiedene Entwicklungsplattformen angeboten werden [Nandyal und Rafi, 2020]. Die Dienste der meisten BaaS Anbieter lassen sich grob in fünf Bereiche einteilen.

- **Authentifizierungs- und Autorisierungsdienste:** Viele Anwendungen benötigen Dienste für eine Nutzerverwaltung. Der BaaS Anbieter stellt Dienste bereit, mittels derer sich ein Nutzer authentifizieren und für bspw. den Zugriff auf Daten auch autorisieren kann. Oft lassen sich dafür auch Identitäten von anderen Identitätsanbietern wie Google oder Facebook verwenden. Benutzt werden weit verbreitete Protokolle wie OAuth oder SAML.
- **Dienste für die Datenhaltung:** Kern der Dienste vieler BaaS Anbieter sind Dienste für die Datenhaltung und -verwaltung. Mit diesen Diensten lassen sich

Daten speichern, Abfragen und synchronisieren, oft unabhängig davon ob ein Client gerade on- oder offline ist.

- **Dienste zur Benutzerinteraktion:** Mittels Push Benachrichtigungen lassen sich alle Nutzer einer Anwendung direkt erreichen, auch Dienste zur Analyse der Nutzer werden angeboten.
- **Dienste für die Entwicklung:** Auch Dienste für die Entwicklung einer Anwendung angeboten, bspw. für das Hosten einer Webanwendung oder auch zum Implementieren eigener Geschäftslogik. Dies erfolgt oft in Form von Cloud Functions, dem elementaren Bestandteil des FaaS Konzeptes. Weiterhin werden wie oben erwähnt für viele verschiedene Plattformen SDKs angeboten, um die Dienste in die eigene Anwendung zu integrieren. Auch Schnittstellen wie REST oder GraphQL sind oft vorhanden.
- **Dienste für die Sicherheit:** Abgesehen von Authentifizierungsdiensten wird auch die Verschlüsselung von Netzwerkanfragen oder auch der gesamten Daten geboten.

[Nandyal und Rafi, 2020]



Quelle: [Roberts und Chapin, 2017]

Abbildung 2.3: Backend-as-a-Service Architektur

Abbildung 2.3 zeigt eine mögliche Architektur eines BaaS. Zugriffe vom Client erfolgen dabei direkt auf Dienste wie die Datenbank oder die Authentifizierung. Zusätzlich kann ein API Gateway verwendet werden, um eine Anfrage aus dem Client zu den entsprechenden Cloud Functions zu routen, mittels der dann die Geschäftslogik implementiert wird. Im nächsten Abschnitt wird auf die mit der Verwendung eines BaaS einhergehenden Vor- und Nachteile eingegangen.

Vor- und Nachteile der Verwendung eines Backend as a Service

Die Verwendung eines BaaS bringt verschiedene Vor- und Nachteile mit sich. Dadurch, dass Server und andere Komponenten nicht mehr selbst verwaltet werden müssen, entstehen geringere operationale Kosten, auch der Verwaltungsaufwand sinkt. Außerdem wird vom Skaleneffekt profitiert: Durch das Verwenden standardisierter Dienste verringern sich auch die Kosten für den Anbieter dieser Dienste was letztendlich an den Kunden weitergegeben wird [Roberts, 2018]. Auch die Entwicklungskosten für Anwendungen lassen sich so verringern, da bei einem BaaS ganze Komponenten des Systems standardisiert bereitgestellt werden. Somit muss für viele Dienste nur eine Integration in die eigene Anwendung erfolgen, ohne diese Dienste selbst implementieren zu müssen. Überdies hinaus werden durch das automatische Skalieren der Dienste die Kosten für das Betreiben einer Anwendung verringert [Roberts und Chapin, 2017].
Die Zeit, die vergeht, bis ein Feature oder Produkt den Markt erreicht, wird durch die Verwendung eines BaaS verringert [Eismann u. a., 2020]. Durch geringere Kosten, weniger Aufwand für die Bereitstellung und kaum Verwaltungsaufwand lassen sich neue Anwendungen oder Features schnell umsetzen [Roberts, 2018].
Weiterhin lässt sich auch das Risiko verringern, das mit dem Betreiben einer Anwendung einhergeht. Server oder Datenbanken können ausfallen, was zu Ausfallzeiten einer Anwendung führt. Dieses Problem wird sogar noch größer, je mehr unterschiedliche Komponenten ein System besitzt. Das Risiko eines Ausfalls gilt zwar auch für die Anbieter eines Dienstes, aber durch Service Level Agreements wird dem Kunden eine gewisse Verfügbarkeit garantiert. Der Anbieter kümmert sich somit um das Beheben von Problemen und Ausfällen [Roberts und Chapin, 2017]. Auch die automatische Skalierung, ein oft erwähnter Vorteil im Serverless Bereich, gilt auch für die Verwendung eines BaaS [Shafiei u. a., 2019].
Die Verwendung eines BaaS birgt jedoch auch Gefahren. Die Kontrolle über Funktionen der eigenen Anwendung wird vollständig an einen externen Dienstleister abgege-

ben, was bspw. dazu führen kann, dass unvorhergesehene Kosten entstehen oder vorher nicht bekannte Limitierungen auftreten. Der Kontrollverlust bringt auch einen Mangel an möglicher Konfiguration mit sich. Auch die Funktionalität einer Anwendung kann eingeschränkt werden, wenn standardisierte Dienste verwendet werden [Roberts, 2018].
Einer der größten Nachteile ist jedoch der entstehende Vendor lock-in. Die Dienste eines Anbieters variieren im Vergleich zu den Diensten eines anderen Anbieters. Somit müssten bei einem Wechsel die entsprechenden Komponenten der Anwendung angepasst werden. Teilweise ergeben sich auch bedingt durch die Architektur Abhängigkeiten von Diensten, die ein anderer Anbieter nicht anbietet. Das Migrieren einer Anwendung ist also in den meisten Fällen äußerst schwierig [Sbarski und Kroonenburg, 2017].
Auch die Sicherheit einer Anwendung kann durch Verwendung eines BaaS leiden, bspw. erfolgt der Zugriff auf eine Datenbank meist direkt vom Client aus ohne eine gesonderte Schicht, die den Zugriff regelt. Dies erfordert eine besondere Aufmerksamkeit beim Entwerfen der Anwendung. Zudem werden Daten bei einem Drittanbieter gespeichert, was wiederum einen Verlust an Kontrolle und Sicherheit bedeutet [Roberts, 2018].
Ein weiterer Punkt kann in manchen Fällen die Performance sein: Viele BaaS Anbieter erlauben die Verwendung von Cloud Functions zur Implementierung eigener Geschäftslogik. Eines der bekanntesten Probleme dabei ist der sogenannte Cold Start. Dabei geht es im FaaS Konzept um die Zeit der Instantiierung des Containers, in dem dann der Code ausgeführt wird. Wenn eine solche Funktion länger nicht ausgeführt wurde so dauert das Starten des Containers länger. Dies kann bei Anwendungen mit kritischer Latenz zu Problemen führen [Baldini u. a., 2017].
Weiterhin wird bei den Nachteilen von Serverless auch das Debugging und Testen erwähnt. Auf Grund der Tatsache, dass das Serverless Konzept vergleichsweise neu ist, sind die Entwicklungstools noch nicht sehr ausgereift, obwohl das Debuggen und Testen elementare Bestandteile der Softwareentwicklung sind [Shafiei u. a., 2019].

3 Analyse und Spezifikation

In diesem Kapitel wird die Analyse und Spezifikation der zu implementierenden Anwendung durchgeführt. Dabei wird zuerst der Anwendungsfall vorgestellt bevor auf die Vorgehensweise für den praktischen Teil eingegangen wird. Danach werden die Anforderungen an die Anwendung erhoben und spezifiziert, wobei das Backend der Anwendung im Fokus steht.

3.1 Vorstellung und Analyse des Anwendungsfalles

Das Team von Evender ist ein junges, diverses Gründungsteam, das mit einer mobilen Anwendung das Suchen und Finden von Veranstaltungen jeglicher Art nachhaltig verändern möchte. Die Idee zu dieser Applikation entstand im Rahmen der Bearbeitung des Praktikumsprojektes in dem Modul Software Engineering und Architektur II. Mit Hilfe der App soll die bisher oft sehr zeitaufwändige Recherche nach Veranstaltungen bei diversen Quellen wie Eventim oder Eventbrite vereinfacht werden. Um einen Überblick über verschiedene Arten von Veranstaltungen zu bekommen ist es nötig, auf mehreren Plattformen danach zu suchen. Beispielsweise gibt es bei Eventim ausschließlich Konzerte und bei Eventbrite größtenteils Partys. Diese Problematik will Evender mit einer mobilen App lösen.
Dazu sollen Veranstaltungen aus möglichst vielen Quellsystemen in der App konsolidiert werden. Veranstaltungen sollen in einer Persistenzkomponente gespeichert und dem Nutzer in der App dann in Form eines Swipe-Stacks präsentiert werden. Der Nutzer kann dann durch Swipen einer Veranstaltung diese zu seinen Favoriten hinzufügen oder verwerfen. Ergänzt werden soll die Kernfunktionalität mit Informationen zu den einzelnen Veranstaltungen (bspw. Ort der Veranstaltung, Eintrittspreise, eine Beschreibung oder der Ticketlink), mit einer Filterfunktion und mit einer interaktiven Karte. Auch ein Ticketkauf über die App ist mittelfristig geplant sowie ein Social Feature, um sehen zu können, für welche Veranstaltungen sich seine Freunde interessieren.

Zur Umsetzung dieser Anwendung kann sich ein BaaS eignen, da Evender keine eigene Infrastruktur für das Betreiben oder entwickeln eines Backends besitzt und eine erste Version der App möglichst kostengünstig umgesetzt werden soll. Auch die verringerte time-to-market ist auf Grund des umkämpften Marktes bei mobilen Apps ein wichtiger Punkt, der für eine Umsetzung mittels eines BaaS spricht. Ob das Konzept eines BaaS in diesem Fall wirklich geeignet ist soll im weiteren Verlauf dieser Arbeit mittels eines Vergleiches verschiedener Anbieter untersucht werden. Zuerst wird auf die Vorgehensweise für den praktischen Teil eingegangen bevor dann die Anforderungen an die App erhoben und spezifiziert werden.

3.2 Vorgehensweise im praktischen Teil

Für den praktischen Teil der Thesis wird das Backend der mobilen Anwendung bei drei BaaS Anbietern implementiert. Auf diese Weise soll untersucht werden, ob sich ein BaaS für Evender eignet, welcher Anbieter die Anforderungen am Besten erfüllt und wo Gemeinsamkeiten und Unterschiede liegen. Dabei wird nach dem Wasserfallmodell vorgegangen. Nach der Analyse und Spezifikation folgt die Konzeption des Backends, welches dann bei allen drei Anbietern implementiert wird. Zugleich erfolgt eine Integration der Backendanwendungen in die mobile App. Der Quellcode der implementierten Cloud Functions und der iOS App befindet sich im elektronischen Anhang der Arbeit. Die resultierenden Backendanwendungen werden in einem Vergleich gegenübergestellt. Für den Vergleich der Anbieter wird im Abschnitt der Konzeption ein Kriterienkatalog entwickelt, auf dessen Basis dann der Vergleich durchgeführt wird.

3.3 Anforderungserhebung

In diesem Abschnitt wird auf die Erhebung der Anforderungen an die mobile Anwendung eingegangen. Dabei wird insbesondere die Methodik erläutert. Die resultierenden Anforderungen an die mobile Applikation befinden sich im Anhang der Arbeit.

"Der Erfolg einer Softwareentwicklung hängt letztlich davon ab, ob das fertiggestellte System den Erwartungen der Beteiligten entspricht"[Broy und Kuhrmann, 2021]. Manfred Broy hebt mit dieser Aussage die Wichtigkeit der Anforderungserhebung und der Spezifikation bei der Durchführung eines Software-Projektes hervor. Alle Beteiligten eines

Projektes haben dabei unterschiedliche Erwartungen an das System. Das Ziel der Analyse ist es also, Anforderungen zu erfassen, zu konsolidieren und zu dokumentieren und das aus der Sicht aller Stakeholder. Funktionale Anforderungen sind diejenigen Anforderungen, die das resultierende System in Bezug auf ihre Nutzbarkeit beschreiben, es stehen die nutzbaren Funktionen an der Systemgrenzen im Mittelpunkt. Nicht-funktionale Anforderungen hingegen legen die Charakteristika des Systems in Bezug auf die Art der Leistungserbringung fest [Broy und Kuhrmann, 2021].
Zur Erfassung der Anforderungen für das in dieser Arbeit umzusetzende System kommen verschiedene Methoden zum Einsatz. User Storys bilden dabei das zentrale Element zur Dokumentation der Anforderungen. Eine User Story ist eine Beschreibung dessen, wie das zu implementierende System eine Aufgabe für einen Nutzer umsetzt. Eine solche Story sollte dabei den INVEST Kriterien (Independent, Negotiable, Valuable, Estimable, Small, Testable) genügen und Akzeptanzkriterien aufweisen, um die Erfüllung der Story belegen zu können [Broy und Kuhrmann, 2021]. Außerdem wird eine Persona verwendet, um einen Einblick in die Denkweise möglicher Nutzer zu erhalten. Personas eignen sich gut als Ergänzung zu User Storys, da die User Storys die Anforderungen zwar aus Anwendersicht beschreiben, aber oft wird der Anwender dabei nicht näher beschrieben. Somit helfen Personas beim Steigern des Verständnisses für hypothetische Anwender [Holt u. a., 2011].
Um nun die Anforderungen an die mobile Anwendung und an das Backend im Speziellen zu erheben wurde ein Workshop mit dem Team von Evender durchgeführt. Dabei war es wichtig, zuerst die Anforderungen an die gesamte Anwendung zu erheben, um daraus dann die Anforderungen an das Backend der Anwendung abzuleiten. Die grundlegende Aufteilung der User Storys erfolgt in funktionale und nicht-funktionale Anforderungen. Die Menge an User Storys aus dem Workshop wird dann weiter verfeinert: Aufgrund einiger Redundanzen werden einige Storys zusammengefasst oder, wenn sie zu groß gefasst sind, gesplittet. Das Zusammenfassen und Aufteilen der Storys erfolgt methodisch nach den oben erwähnten INVEST Kriterien für User Storys. Erst wenn diese Kriterien erfüllt sind, sind die Storys für eine Umsetzung geeignet. Die User Storys wurden dahingehend ergänzt, als dass sie auch für dieses Projekt relevante Qualitätseigenschaften von Software repräsentieren. Um eine Definition of Ready zu erreichen, werden für diese Storys Akzeptanzkriterien ergänzt, um die Erfüllung messbar zu machen. Als Darstellungsform der User Storys wird eine Darstellung als Story Card gewählt: Die Darstellung beinhaltet den Namen der Story, die Beschreibung und die Akzeptanzkriterien. Zu finden ist diese Darstellung der User Storys im Anhang, genauso wie die erstellte Persona und die initiale Menge an User Storys aus dem Workshop. Da im Rahmen dieser Arbeit das Backend

der Anwendung umgesetzt werden soll, werden die Anforderungen an das Backend im Folgenden detaillierter dargestellt.

3.4 Anforderungen an das Backend

Aus den erhobenen Anforderungen an die mobile Applikation an sich lassen sich die Anforderungen speziell an das Backend ableiten. Viele der im Workshop erhobenen Anforderungen beziehen sich nur auf die mobile Applikation, bspw. eine Funktion zum erneuten Anzeigen einer bereits geswipten Veranstaltung. Andere Anforderungen hingegen beziehen sich auf die zu implementierende Geschäftslogik im Backend, diese Anforderungen werden im folgenden Teil dargestellt und in den weiteren Kapiteln der Arbeit umgesetzt.

Funktionale Anforderungen

- **Speichern von Veranstaltungen:** Veranstaltungen müssen aus Quellen in eine Datenbank importiert werden (bspw. Über API oder Scraping), damit diese in einem Swipe Stack präsentiert werden können. Des Weiteren müssen nicht nur die Veranstaltungen und zugehörige Informationen, sondern auch die Bilder, die bei jeder Veranstaltung zu sehen sein sollen, in eine Storage Komponente importiert werden (mit entsprechendem Mapping zu den Veranstaltungen in der Datenbank).
- **Veranstaltungsinformationen:** Veranstaltungen in der Datenbank müssen verschiedene Informationen zugehörig haben: Koordinaten, Ticketlink und alle entsprechenden Attribute für den Filter.
- **Newsletter:** Für einen automatischen Newsletter muss ein Sending Service verwendet werden und eine Umsetzung, bspw. in Form einer Cloud Funktion, im Backend stattfinden.
- **Favoriten:** Favoriten und nicht favorsierte Veranstaltungen müssen beim Swipen in der Datenbank gespeichert werden, zum Einen damit Veranstaltung in der Favoritenliste sichtbar sind und zum Anderen, damit Nutzer geswipte Veranstaltungen nicht noch einmal sehen.

- **Empfehlungen:** Für Empfehlungen muss im Backend eine Recommendation Engine verwendet oder umgesetzt werden.
- **Authentifizierung und Autorisierung:** Nutzer sollen sich mit ihrer E-Mail Adresse und mit ihrem Facebook, Google oder Apple Konto anmelden können um dann auch Zugriff auf ihre Daten zu erhalten.
- **Soziale Kontakte:** Nutzer sollen andere Nutzer mittels eines eindeutigen Namens finden können und sich die Favoriten der gesuchten Person anzeigen lassen können.

Nicht-Funktionale Anforderungen

- **Erreichbarkeit:** Die Anwendung sollte bei allen Anbietern jederzeit erreichbar sein.
- **Latenz:** Die Latenz der Anwendung soll möglichst gering sein, dies betrifft vor allem Datenbankabfragen.
- **Modularität:** Die Architektur der Anwendung, insbesondere der mobilen Applikation, sollte möglichst modular sein und eine geringe Kopplung aufweisen, damit das Evaluieren der verschiedenen Backendanwendungen so wenig Änderungen im Code wie möglich verlangt.
- **Skalierung:** Die Anwendung sollte bei hoher Last automatisch skalieren.
- **Kosten:** Es sollen keine Kosten entstehen, wenn die App nicht aktiv ist.

3.5 Spezifikation

Die erhobenen Anforderungen werden in diesem Abschnitt mittels verschiedener Methoden spezifiziert. Dabei wird zuerst genauer auf die Methodik eingegangen bevor die Spezifikation dargestellt wird.

3.5.1 Methodik

Für die Spezifikation werden Wireframes, Anwendungsfälle, ein Komponentendiagramm und ein Datenmodell erstellt. Hierbei geht es um eine strukturierte Darstellung und Dokumentation der erhobenen Anforderungen und darum, validieren zu können, ob die Anforderungen vollständig und korrekt erhoben worden sind. Mit den Anwendungsfällen lässt sich die Funktionalität des Systems oder eines Teils des Systems in sinnvolle Einheiten gliedern und darstellen. Außerdem lässt sich das System auf diese Weise modularisieren [Brandt-Pook und Kollmeier, 2015]. Überdies hinaus sind Daten ein zentraler Bestandteil der Anwendung. Als Teil der Spezifikation kann es sich deshalb lohnen, die im System vorkommenden Daten mittels eines Datenmodells zu spezifizieren. Außerdem sollen mit einem Komponentendiagramm die möglichen Bestandteile des Systems dargestellt werden, dies lässt erkennen, welche Abhängigkeiten und Beziehungen zwischen den Komponenten existieren [Broy und Kuhrmann, 2021]. Ergänzend werden Wireframes erstellt, um darzustellen, wie die mobile App aussehen könnte. Die Wireframes können dabei als Diskussionsgrundlage dienen, bspw. bei zukünftigen Änderungen. Außerdem eignen sich Wireframes als Ergänzung zu den Anwendungsfällen, um die beschriebenen Funktionen zu visualisieren [Berenbrink u. a., 2013]. Auch die erhobenen User Storys dokumentieren die Anforderungen, sie werden als Story Cards dargestellt und mit Akzeptanzkriterien ergänzt, um die Erfüllung messbar zu machen.

3.5.2 Anwendungsfälle, Datenmodell und Komponentendiagramm

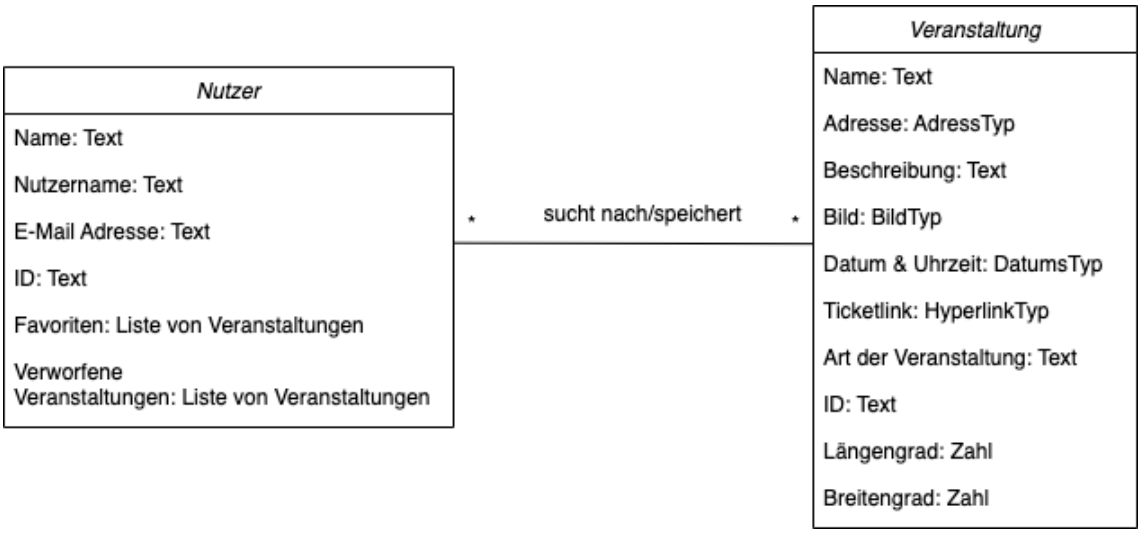
In diesem Abschnitt werden die Anforderungen spezifiziert und mit den gewählten Methoden dargestellt.

Anwendungsfälle

Um einige Funktionen der mobilen App darzustellen, werden Anwendungsfälle verwendet. Die Anwendungsfälle werden dabei aus den Story Cards der User Storys extrahiert und bestehen aus einem Kontext, Vorbedingungen, einem erfolgreichen und einem nicht erfolgreichen Endzustand, den Akteuren, dem Trigger und einer Beschreibung. Für die Anwendungsfälle werden dabei Funktionen der App ausgewählt, die zu den Kernfunktionalitäten gehören. Zu finden sind die Anwendungsfälle im Anhang mit den ergänzenden Wireframes.

Datenmodell

Um die Daten, die bei Verwendung des Systems entstehen, darzustellen, wird ein logisches Datenmodell erstellt. Dies hilft dabei zu verstehen, welche Datentypen es gibt, welche Zugriffe auf diese Daten erfolgen und ermöglicht auch einen ersten Entwurf einer möglichen Architektur der Anwendung. Im Mittelpunkt der Anwendung stehen dabei die Nutzer und die Veranstaltungen, dies sind die elementaren Entitäten des Datenmodells. Mittels verschiedener durch die Anwendung bereitgestellter Funktionen interagieren diese Entitäten miteinander, beispielsweise sucht ein Nutzer in der Hauptansicht nach neuen Veranstaltungen oder er speichert eine dieser Veranstaltungen in seinen Favoriten ab. Die dargestellten Eigenschaften sind dabei diejenigen Eigenschaften, die essentiell sind um das System wie gefordert umsetzen zu können.



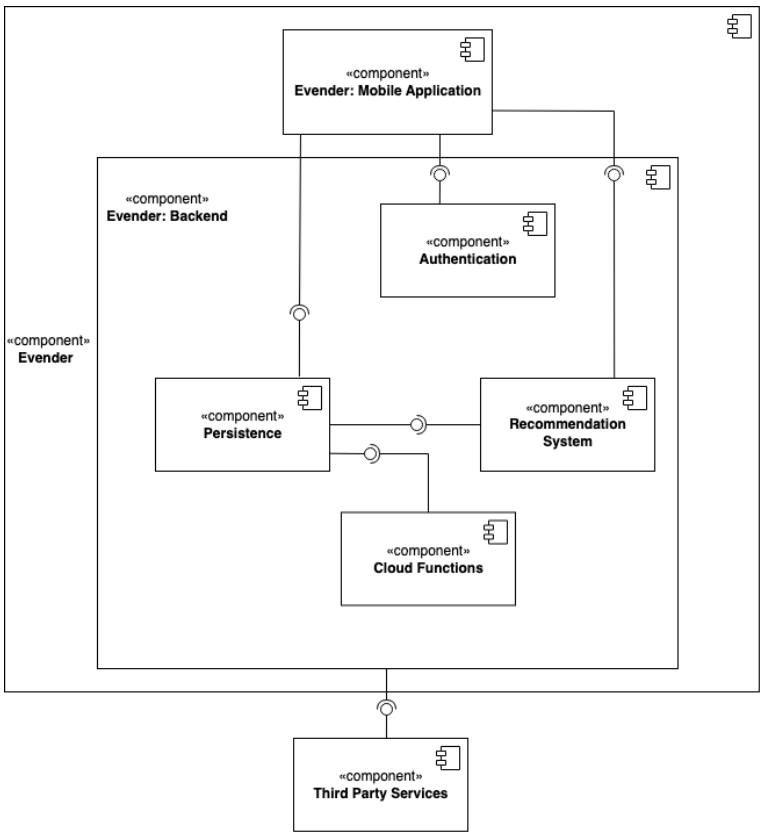
Quelle: Eigene Darstellung

Abbildung 3.1: Logisches Datenmodell

Komponentendiagramm

Aus den bereits spezifizierten Story Cards, den Anwendungsfällen und dem Datenmodell lässt sich ein UML-Komponentendiagramm ableiten, das eine erste Version der möglichen Architektur der Anwendung zeigt. Es wird eine Frontendkomponente geben, die als mobile Applikation realisiert wird und die es dem Nutzer erlaubt, sich anzumelden, Veranstaltungen zu entdecken und zu speichern und auch, diese Veranstaltungen zu teilen. Um Daten zu speichern und die Geschäftslogik zu implementieren, wird aber auch

eine Backendanwendung benötigt, die es im praktischen Teil dieser Arbeit zu entwickeln gilt. Das Backend wird bei mehreren noch auszuwählenden Cloud Anbietern umgesetzt. Dieses Backend benötigt dabei eine Speicherkomponente in Form einer Datenbank und eines Storage, um Nutzer und Veranstaltungen zu speichern. Weiterhin werden auch entsprechende Schnittstellen benötigt, um der mobilen Applikation die Funktionalität bereitzustellen. Auch Geschäftslogik, beispielsweise zum Extrahieren neuer Veranstaltungen aus Quellsystemen, gehört zur Funktionalität der Backendanwendung. Außerdem wird ein Empfehlungssystem benötigt, das es ermöglicht, den Nutzern ihren Interessen entsprechende Vorschläge für Veranstaltungen zu unterbreiten. Im Folgenden wird ein Komponentendiagramm gezeigt, das einen ersten Anhaltspunkt für die zukünftige Architektur gibt.



Quelle: Eigene Darstellung

Abbildung 3.2: Komponentendiagramm

In dem Komponentendiagramm aus Abbildung 3.2 lassen sich alle schon genannten Komponenten erkennen: es gibt eine Frontend-Komponente, die als Client fungiert und mit dem zu entwickelnden Backend kommuniziert. Das Backend übernimmt dabei alle an-

fallenden Aufgaben der Geschäftslogik in Form von Cloud Functions und zusätzliche Aufgaben, wie zum Beispiel das Authentifizieren der Nutzer oder das Speichern von Daten in der Persistenz Komponente. Die gezeigte Architektur gilt es nun auf Grundlage der Anforderungen und der BaaS Prämisse weiter zu verfeinern und auszuarbeiten, sodass am Ende eine Zielarchitektur entsteht, die es bei allen ausgewählten Anbietern umzusetzen gilt.

4 Konzeption

In diesem Kapitel wird die Zielarchitektur für die zu implementierende Backendanwendung entwickelt und es wird der Vergleich der Anbieter konzipiert. Die resultierende Architektur gilt es bei allen BaaS Anbietern umzusetzen. Grundlage für die Architektur bildet dabei das Komponentendiagramm aus dem vorherigen Kapitel sowie Referenzarchitekturen der Anbieter. Da im Rahmen dieser Abschlussarbeit das Backend der Anwendung im Fokus steht, bezieht sich dieses Kapitel ausschließlich auf die Architektur des Backends, wobei aber Schnittstellen zur Frontend-Komponente miteinbezogen werden. Zuerst wird auf die einzelnen Komponenten der Anwendung eingegangen bevor die eigentliche Zielarchitektur erarbeitet wird. Darauf folgt die Auswahl der Anbieter sowie die Entwicklung des Kriterienkataloges, anhand dessen die Implementierungen bei den einzelnen Anbietern verglichen werden können.

4.1 Architektur und Entwurf

In diesem Abschnitt wird auf die Architektur der Anwendung und die einzelnen Komponenten näher eingegangen. Im Fokus steht dabei die Architektur des Backends, vorher wird aber auch das Frontend der Anwendung näher erläutert.

4.1.1 Komponenten der Anwendung

Aus den erhobenen Anforderungen und der Spezifikation lassen sich folgende Komponenten der Anwendung identifizieren.

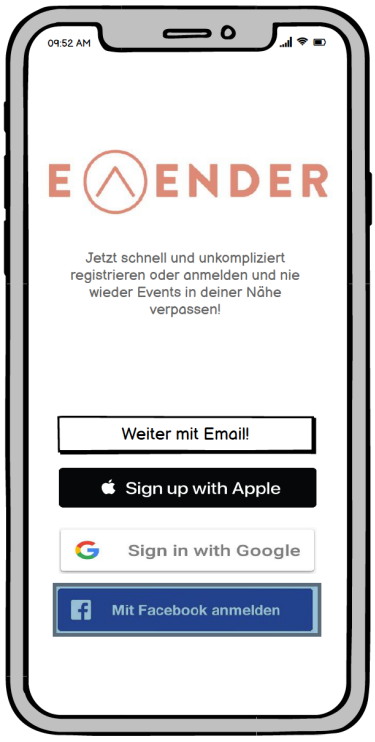
- **Datenbank:** Zum Verwalten der Daten der User und der Veranstaltungen wird eine Datenbank benötigt.

- **Storage:** In der Storage Komponente werden Dateien gespeichert, die innerhalb der Anwendung benötigt werden, bspw. die Bilder der Veranstaltungen.
- **Cloud Functions:** Um benötigte Business Logik zu implementieren stehen bei allen BaaS Anbietern Cloud Functions zur Verfügung. Für diese Cloud Functions stellen die Anbieter lokal oder online eine Entwicklungsumgebung bereit mittels der Code für verschiedene Plattformen geschrieben werden kann. Es wird aller Voraussicht nach eine Funktion zum Laden der Daten der Veranstaltungen in die Datenbank geben und eine Funktion zum Versenden des Newsletters, genauso wie verschiedene Middleware Funktionen.
- **Authentifizierung:** Die Authentifizierung wird mit Diensten der BaaS Anbieter umgesetzt.
- **Frontend:** Mobile App für iOS.
- **Empfehlungssystem:** Eine weitere Anforderung an die Anwendung ist, dass Nutzer ihren Interessen entsprechende Veranstaltungen angezeigt bekommen: Dies lässt sich mit einem Empfehlungssystem lösen, das basierend auf den Interessen des Nutzers oder seiner Interaktion mit Veranstaltungen Vorschläge erstellt.

4.1.2 Frontend

Das Frontend der Anwendung ist eine iOS App für alle gängigen Apple Smartphones. Mittelfristig sollen auch alle anderen Betriebssysteme bedient werden, für eine bestmögliche Performance und Nutzererfahrung wird aber für eine erste Version der Anwendung eine native iOS App entwickelt. Eine hybride App, d.h. eine Entwicklung nur einer einzigen App für mehrere Plattformen (Beispiel: Instagram) kommt nicht in Frage, da native Apps viele Vorteile mit sich bringen, wie bspw. einen besseren Zugriff auf gerätespezifische Funktionen. Die Performance ist aber der wichtigste Grund. Die Anforderungen von Nutzern an eine mobile App sind weitaus umfangreicher geworden: Auf Grund der Vielzahl an existierenden Apps kann mangelnde Performance im Vergleich zu anderen Apps ein Grund sein, eine App direkt nach dem Ausprobieren wieder zu deinstallieren [Janson, 2017]. Die App wird auf macOS 12.3.1 mittels Xcode 13.1 entwickelt, die Programmiersprache der Anwendung ist Swift 5.7. Swift ist eine kompilierte Programmiersprache, die von Apple entwickelt wird und zum Programmieren von Apps für iOS, Mac, Apple TV und Apple Watch gedacht ist [Apple, 2022]. Sie ist seit 2014 verfügbar und seit 2015 Open

source. Die erstellten Wireframes (im Anhang dieser Arbeit) zeigen, wie die resultierende App aussehen könnte. In Abbildung 4.1 ist der Bildschirm zur Authentifizierung zu sehen. Für jeden untersuchten BaaS Anbieter wird ein eigenes Projekt in Xcode erstellt und implementiert. Die mobile App basiert dabei auf dem Model View Control Pattern, dies erlaubt eine saubere Trennung zwischen der Logik, die für das Backend in die App integriert werden muss und der Präsentationsschicht der Anwendung. So kann bei der Entwicklung das Backend der App leicht substituiert werden und es wird eine höhere Vergleichbarkeit gewährleistet.

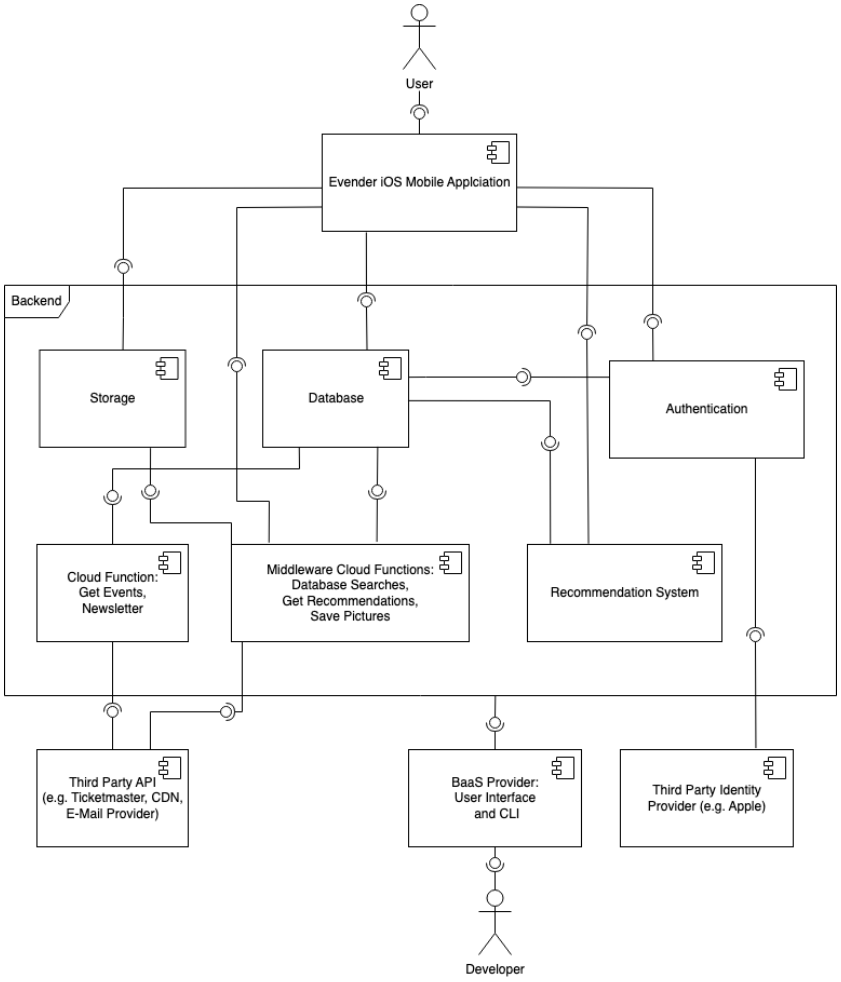


Quelle: Eigene Darstellung

Abbildung 4.1: Wireframe der Ansicht zur Anmeldung

4.1.3 Backend

Das Backend der Anwendung gilt es im Rahmen dieser Arbeit bei den ausgewählten Anbietern zu implementieren. Aus den Komponenten der Anwendung lässt sich eine erste Zielarchitektur ableiten. Das Backend beinhaltet alle im obigen Abschnitt dargestellten Komponenten.



Quelle: Eigene Darstellung

Abbildung 4.2: Zielarchitektur

Das Frontend wird in der Architektur als Black Box dargestellt, da das Backend im Fokus steht. Abbildung 4.2 zeigt die Zielarchitektur des Backends. Das Backend besteht aus den oben erwähnten Komponenten: Einer Datenbank, einer Speicherkomponente, einer Authentifizierungskomponente, verschiedenen Cloud Functions und einem Empfehlungssystem. Der Nutzer interagiert dabei ausschließlich mit dem Frontend. Für den Entwickler stellt der entsprechende BaaS Anbieter Zugriffsmöglichkeiten zur Verfügung, dies kann eine CLI sein oder auch ein Web Interface. Weiterhin wird das Backend Dienste und Schnittstellen von Drittanbietern in Anspruch nehmen müssen, bspw. zum Laden der Daten der Veranstaltungen oder auch der Identitäten von anderen Identitätsanbietern wie Google oder Facebook. Details zu den einzelnen Komponenten lassen sich an

dieser Stelle noch nicht darstellen, da jeder Anbieter unterschiedliche Dienste bereitstellt und die Komponenten unterschiedlich implementiert. Somit können sich die Komponenten hinsichtlich ihrer Eigenschaften deutlich unterscheiden, auf diese Tatsache wird in der Evaluation näher eingegangen. Details zu den einzelnen Komponenten werden im Kapitel des Vergleiches dargestellt.

4.2 Konzeption des Vergleiches

In diesem Abschnitt wird das Konzept des Anbieter Vergleiches erarbeitet. Zunächst wird auf die Auswahl der Anbieter eingegangen bevor ein Katalog an Kriterien erstellt wird, auf Basis dessen dann abschließend der Vergleich durchgeführt wird.

4.2.1 Auswahl der Provider

Für den Vergleich werden die verschiedenen Anbieter abhängig von mehreren Faktoren ausgewählt. Wichtigste Faktoren bei der Entscheidung, welche Anbieter im Vergleich untersucht werden, sind die eigenen Fähigkeiten, die Anforderungen an die Anwendung und auch die Möglichkeit, kostenlos in die Entwicklung einzusteigen. Für ein BaaS gibt es eine ganze Reihe von Anbietern, dazu gehören bspw. Firebase von Google, Parse, das mittels eines Stacks unterschiedlicher Technologien ein BaaS implementiert, jedoch eigenes Hosting erfordert, AWS mit seinem Amplify Dienst oder auch Backendless und Kinvey. Für den Vergleich sind Google Firebase, AWS Amplify und Back4App gewählt worden. Googles Firebase stellt dabei auf Grund der großen Community und der Popularität einen guten Ausgangspunkt für den Vergleich dar. Außerdem bietet Firebase viele verschiedene Dienste und Features an, deshalb lohnt sich hier der Vergleich mit anderen Anbietern. Weiterhin wurden zur Umsetzung des Backends AWS Amplify und Back4App ausgewählt. Amplify wurde 2017 von AWS vorgestellt und bietet genau wie Firebase alle nötigen Dienste an, die zur Umsetzung des Backends der Anwendung notwendig sind. Interessant bei Amplify ist vor allem die Integration mit allen anderen von AWS angebotenen Diensten, da AWS der meist verwendete Cloud Provider ist und auch bei den angebotenen Diensten regelmäßig große Innovationsfreude zeigt [Zhang, 2022]. Als dritter zu untersuchender Anbieter wurde Back4App gewählt. Interessant ist hier vor allem die Tatsache, dass Back4App das Open source Framework Parse als darunterliegenden Technologie Stack verwendet, sodass zumindest theoretisch die Gefahr eines Vendor

lock-ins geringer ist. Neben den drei erwähnten Anbietern gibt es noch viele weitere, die sich jedoch nur bedingt zur Umsetzung der Anwendung eignen, weil zum Beispiel für die Umsetzung essentielle Dienste wie die Cloud Functions nicht angeboten werden oder die Verwendung nicht kostenfrei ist.

Google Firebase

Firebase an sich existiert seit 2011 und wurde 2014 von Google aufgekauft [javaTpoint]. Firebase ist eine Development Plattform für Web- oder mobile Apps die von Google betrieben wird. Die Plattform bietet dabei verschiedene Lösungen und Produkte an, die sich in drei verschiedene Kategorien gliedern lassen: Build, Release and Monitor und Engage. Mittels der Dienste und Produkte in der Build Kategorie wird die eigentliche App Entwicklung bedient: Es wird ein vollständig verwaltetes Backend bereitgestellt. Zu den Diensten in dieser Kategorie gehören beispielsweise Storage, Database und Cloud Functions. Der Release and Monitor Bereich stellt Dienste zur Verfügung, die sich mit der Performanz und Stabilität der Applikation beschäftigen. Auch Google Analytics als Analysetool fällt in diesen Bereich. Des Weiteren ist ein Testbereich vorhanden, mit dem es möglich ist, neue Versionen der Applikation auf virtuellen Geräten zu testen. Die Firebase Engage Kategorie stellt Dienste bereit, um das User Engagement zu erhöhen, beispielsweise mittels A/B Testing oder Messaging Kampagnen. Firebase Projekte werden mittels der Google Cloud Plattform betrieben, sodass eine nahtlose Integration mit dessen Diensten möglich ist [Firebase, 2022j].

AWS Amplify

AWS Amplify bietet, ähnlich wie Firebase, eine große Auswahl an Diensten an, um ein Backend für eine mobil oder Web Anwendung zu konzipieren. Amplify bietet dabei dem Entwickler die Möglichkeit, die volle Bandbreite der AWS Dienste nutzen zu können. Mit Amplify lassen sich viele Anwendungsfälle abdecken: Es ist möglich, mittels der CLI und angebotenen Libraries ganze Backends zu konzipieren, außerdem kann gänzlich ohne Code mit Amplify Studio eine Fullstack Anwendung erstellt werden. Überdies hinaus lassen sich auch Anwendungen in Amplify hosten. Per Amplify CLI lassen sich Backend Dienste ohne die Integration in eine Anwendung erstellen, dies kann z.B. von Vorteil sein, wenn eine Anwendung schon besteht und bspw. nur das Backend migriert werden soll. Die Dienste von AWS Amplify umfassen unter anderem Authentifizierung, Storage,

eine GraphQL API, DataStore, Analytics und ML Modelle. Die nahtlose Integration mit allen anderen AWS Diensten stellt sicher, dass das schon breite Angebot an Diensten jederzeit erweiterbar ist. Der Entwickler hat somit jederzeit Zugriff auf alle anderen AWS Dienste.

Back4App

Back4App ist ein weiterer BaaS Anbieter. Die Plattform basiert dabei auf dem Open Soruce Framework Parse. Zu den Features, die Back4App anbietet, gehören eine Datenbank, Cloud Functions, GraphQL APIs, ein File Storage, Authentifizierung und Push Benachrichtigungen. Parse wurde 2013 von Facebook aufgekauft und dazu benutzt, Anwendungen des Unternehmens und Drittanbieteranwendungen auf Facebook zu unterstützen. 2016 hat Facebook Parse als Open Source Framework veröffentlicht [Clark, 2022]. Parse lässt sich auch ohne Back4App verwenden. Es ist möglich, die einzelnen jeweils benötigten Komponenten selber auf einem Server zu installieren und laufen zu lassen. Das Hosting muss dann aber vom Entwickler übernommen werden. Dies bietet jedoch die Flexibilität, auch nur die Komponenten zu betreiben und zu verwenden, die auch wirklich benötigt werden. Für die meisten Programmiersprachen und Plattformen bietet Parse entsprechende SDKs für die Integration in die eigene Anwendung an.

4.2.2 Entwicklung des Kriterienkataloges

In diesem Abschnitt werden die Kriterien entwickelt, anhand derer im weiteren Verlauf der Arbeit der Vergleich durchgeführt wird. Die Kriterien basieren dabei zu einem großen Teil auf dem ISO 25010 Standard, in dem ein Modell für die Qualität von Software festgelegt ist. Das Modell beinhaltet acht Merkmale mit jeweils mehreren untergeordneten Merkmalen [ISO/IEC, 2011]. Von diesen Merkmalen werden diejenigen in den Kriterienkatalog aufgenommen, die sich für eine Überprüfung eines BaaS im Rahmen des Vergleiches eignen. Außerdem wird im Kriterienkatalog auf weitere Charakteristiken wie bspw. die entstehenden Kosten eingegangen, die besonders für die Entwicklung von Anwendungen in der Cloud interessant sind. Für alle Kriterien, die sich quantifizieren lassen, werden im Vergleich entsprechende Messungen vorgenommen.

Überblick

- **Umsetzung der Anforderungen:** Hier soll untersucht werden, ob die erhobenen Anforderungen bei den Anbietern umgesetzt werden konnten.
- **Architektur:** Bei diesem Kriterium wird analysiert, ob die resultierende Architektur Unterschiede zur entwickelten Zielarchitektur aufweist.
- **Kosten:** Mittels einer beispielhaften Kostenrechnung werden die Preise für die Verwendung der BaaS Anbieter verglichen.
- **Regionen:** Hier soll verglichen werden, in welchen Regionen die Anbieter aktiv sind und ihre Dienste anbieten.

Dienste

- **Angebot und Vielfalt:** Es wird untersucht, welche Dienste der Anbieter zur Umsetzung einer Anwendung bietet. Dabei wird nicht nur auf verwendete sondern auch auf andere angebotene Dienste eingegangen.
- **Integration:** Bei diesem Kriterium wird die Integration des Backends in das Frontend analysiert. Dabei wird vor allem auf die Software Development Kits der Anbieter an sich und deren Verwendung eingegangen.
- **Persistenz:** Dieses Kriterium dient der Untersuchung der Persistenzkomponente des Backends, dabei wird besonders auf die Eigenschaften des angebotenen Dienstes eingegangen.
- **Authentifizierung:** Mit diesem Kriterium wird die Umsetzung der Authentifizierung und Autorisierung bei den Anbietern analysiert.
- **Cloud Functions:** Bei den Cloud Functions wird untersucht, welche Entwicklungs- und Laufzeitumgebungen angeboten werden und wie die Plattformen skalieren.
- **Storage:** Hier wird auf die Speicherkomponenten der Anbieter eingegangen, im Besonderen auf die Art des Speichers und wie der Zugriff erfolgt.

- **Empfehlungssystem:** Mittels dieses Kriteriums wird die Einbindung eines Empfehlungssystems untersucht. Dabei wird auf die Frage eingegangen, welche Möglichkeiten der Anbieter für die Erstellung ein solches System bietet und wie die genaue Umsetzung erfolgt.

Leistungseffizienz

- **Zeitverhalten:** Hier wird das Zeitverhalten der Dienste der Anbieter untersucht. Dabei wird vor allem auf das Zeitverhalten der Anfragen an das Backend, also des Storage und der Datenbank eingegangen.

Benutzbarkeit

- **Dokumentation:** Zur Umsetzung einer Anwendung bei einem BaaS Anbieter stellt die zur Verfügung gestellte Dokumentation eine wichtige Grundlage dar. Es wird untersucht, wie detailliert und benutzerfreundlich die Dokumentationen sind.
- **Bedienbarkeit:** Bei der Bedienbarkeit wird untersucht, welche Entwicklerschnittstellen vom Anbieter bereitgestellt werden und wie sich diese bedienen lassen.

Verlässlichkeit

- **Verfügbarkeit:** Es wird genauer auf die Verfügbarkeit der einzelnen Dienste eines Anbieters eingegangen, insbesondere in Bezug auf die dem Entwickler gebotenen Service Level Agreements.
- **Skalierbarkeit:** In diesem Abschnitt wird die Fähigkeit zur Skalierung des Anbieters dargestellt, da dies ein grundlegender Bestandteil des Serverless Paradigmas ist.
- **Disaster Recovery und Backup:** Mittels dieses Kriteriums wird erarbeitet, inwiefern der Anbieter den Kunden bei Ausfällen unterstützt bzw. welche Backup Möglichkeiten geboten werden.

Wartbarkeit

- **Analysierbarkeit und Testbarkeit:** In diesem Abschnitt wird die Analysierbarkeit und Testbarkeit der Anbieter verglichen, indem auf die gebotenen Monitoring und Debugging Möglichkeiten eingegangen wird.

Sicherheit

- **Sicherheit:** Mittels dieses Kriteriums werden die vom Anbieter gebotenen Sicherheitsmaßnahmen verglichen und evaluiert. Dabei steht der Zugriff auf die Daten innerhalb der Anwendung im Vordergrund.

5 Vergleich

In diesem Kapitel werden die implementierten Backendanwendungen bei den entsprechenden Anbietern in Bezug auf den Kriterienkatalog dargestellt. Die Struktur entspricht dabei der des Kriterienkataloges. Am Ende jedes Abschnittes werden die Unterschiede und Gemeinsamkeiten der Anbieter in Bezug auf die jeweiligen Kriterien hervorgehoben. Außerdem wird im Rahmen des Vergleiches ein Punktesystem eingeführt, um die Leistung der Anbieter in Bezug auf die Kriterien übersichtlich darzustellen. Dabei werden zwischen einem und drei Punkte vergeben, wobei drei die Höchstpunktzahl darstellt. Diese Punkte befinden sich in der Übersicht am Ende jedes Abschnittes. In der Evaluation wird auf die verteilten Punkte näher eingegangen.

5.1 Überblick

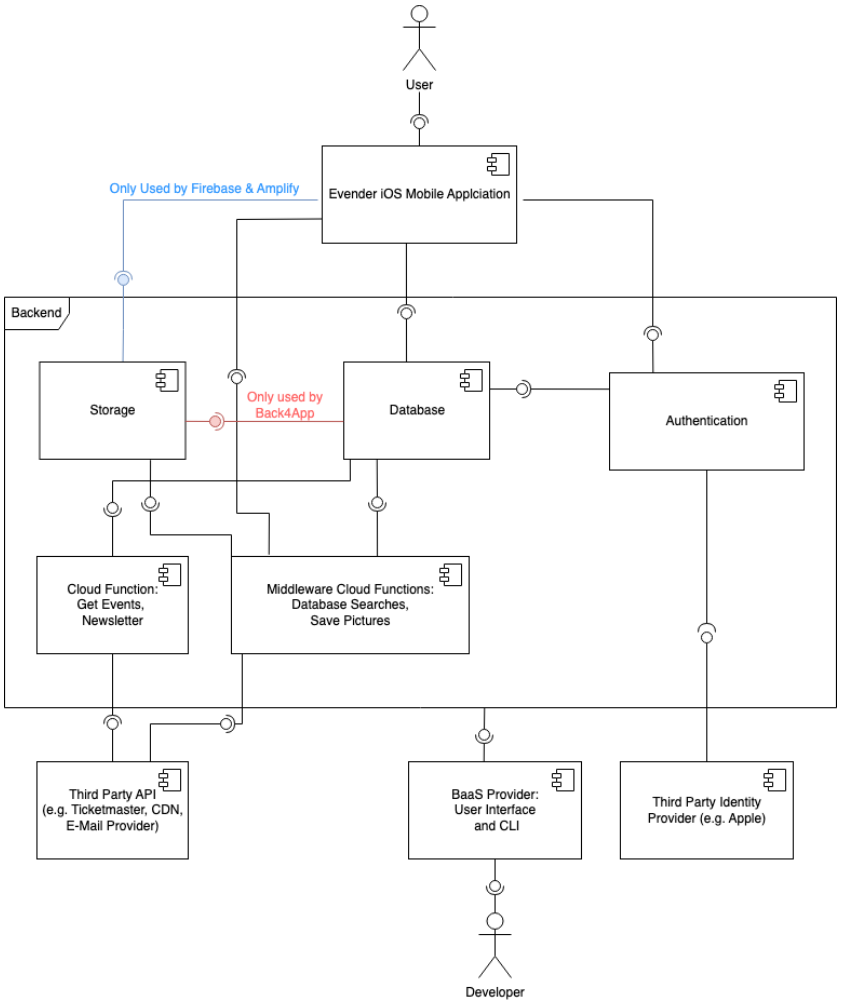
In diesem Abschnitt werden die implementierten Backendanwendungen in Bezug auf die folgenden Kriterien untersucht: Umsetzung der Anforderungen, Architektur, Kosten und die Regionen, in denen der Anbieter aktiv ist. Dies soll einen ersten Eindruck der jeweiligen Implementierungen und der Anbieter vermitteln.

5.1.1 Umsetzung der Anforderungen

Die in der Analyse und Spezifikation erhobenen funktionalen Anforderungen an das Backend konnten bei allen untersuchten Anbietern erfolgreich umgesetzt werden, bis auf das System zur Generierung der Empfehlungen. Google Firebase und AWS Amplify bieten diesbezüglich zwar jeweils einen Dienst an, der es ermöglicht, Empfehlungen auf einfache Weise in die Backendanwendung zu integrieren, aber auf Grund der nicht vorhandenen Daten zum Trainieren eines Modells konnte diese Anforderung nicht erfüllt und umgesetzt werden. Back4App hingegen bietet keine Möglichkeit zur Umsetzung eines Empfehlungssystems. Bei diesem Anbieter müsste ein solches System außerhalb des BaaS umgesetzt

werden. Die benötigten Daten zum Trainieren eines Modells lassen sich jedoch bei allen Anbietern mittels eines Analytics Dienstes sammeln. Auf die nicht-funktionalen Anforderungen aus der Analyse und Spezifikation wird im Abschnitt der Verlässlichkeit und der Kosten näher eingegangen.

5.1.2 Architektur



Quelle: Eigene Darstellung

Abbildung 5.1: Resultierende Architektur

Die resultierenden Architekturen der Implementierung bei den verschiedenen Anbietern unterscheiden sich nur in Details von der Zielarchitektur aus dem Entwurf. Die Unterschiede sind in Abbildung 5.1 dargestellt. Das Empfehlungssystem fehlt in der resultie-

renden Architektur, da eine Umsetzung im Rahmen dieser Arbeit nicht möglich war. Außerdem wird für das Versenden des Newsletters keine Drittanbieter Sendeplattform verwendet, sondern ein Node.js Modul. Allerdings wird ein Zugriff auf ein E-Mail Postfach benötigt. Weiterhin besitzt Back4App eine Besonderheit in Bezug auf die Storage Komponente: Diese ist direkt mit der Datenbank verknüpft, Dateien die nicht einem Objekt in der Datenbank zugeordnet sind, sind nicht auffindbar. Es ist kein direkter Zugriff auf den Storage möglich, sondern es muss mittels der Datenbank auf Dateien zugegriffen werden. Auf diese Tatsache wird im Abschnitt der Dienste noch näher eingegangen.

5.1.3 Kosten

Zum Schätzen der Kosten, die das Betreiben des Backends der mobilen Anwendung bei einem Anbieter verursacht, wird eine theoretische Kostenrechnung erstellt. Dabei werden die Kosten auf Basis der verwendeten Dienste und auf einer theoretischen Nutzerzahl und Nutzung berechnet. In Xcode lassen sich beim Verwenden der Anwendung die Netzwerkdaten verfolgen, also bspw. die Menge an gesendeten und empfangenen Bytes. Dies wird in Verbindung mit dem Instruments Entwicklungstool genutzt, um ein mögliches Nutzungsprofil eines Nutzers zu erstellen. Mittels Instruments wird die Anazhl der Netzwerkanfragen überwacht, die bei vielen Cloud Diensten eine Rolle spielen. Das resultierende Nutzungsprofil sieht wie folgt aus: 10.000 monatlich aktive Nutzer, bei 20 Minuten Verwendung der App pro Woche ergeben sich 200 einzelne API Anfragen an das Backend, bspw. an die Datenbank. Dabei werden 50 MB an Daten gesendet und empfangen. Die 50 MB große transferierte Datenmenge besteht dabei zu einem großen Teil aus den Bildern aus dem Storage, von den 50 MB pro Woche und Nutzer besteht nur ein MB aus Daten aus der Datenbank. Außerdem wird die aktuell implementierte API für Ticketmaster ein Mal täglich nach neuen Veranstaltungen durchsucht. Dabei werden durchschnittlich 20 neue Veranstaltungen gefunden. Die Speicherung von Veranstaltungen und Nutzern in der Datenbank wird dabei vernachlässigt, weil diese Daten keine nennenswerten Größen erreichen. Das Speichern der Bilder wird jedoch Auswirkungen auf die Kosten haben, deshalb wird pro Veranstaltung und pro Bild eine Größe von durchschnittlich 100 KB angenommen. Außerdem wird geschätzt, dass sich knapp ein Drittel der Nutzer dafür entscheidet, den Newsletter zu erhalten. Somit ergeben sich folgende Gesamtzahlen für die Nutzung des Backends in einem Monat:

- Anzahl monatlicher Anfragen: 8.000.000 für Nutzeranfragen, für die Funktionen 600 Anfragen um Veranstaltungen in die Datenbank hinzuzufügen, dazu 600 Anfragen

an den Storage und für den Newsletter zusätzliche 3.000 Anfragen pro Woche, also 12.000 im Monat, somit ergeben sich insgesamt 8.013.200 Anfragen pro Monat.

- Übertragene Datenmenge: 2.000.000 MB, entspricht 2.000 GB, davon 1960 GB für den Storage und 40 GB für die Datenbank.
- Storage für Bilder: 60 MB kommen im Monat hinzu, aktuell gespeichert sind 128 MB.
- Persistenz: Bei 50 Nutzern und 1200 Veranstaltungen aktuell 2 MB, also bei 10.000 Nutzern und 5.000 Veranstaltungen 24 MB. Der verwendete Speicherplatz lässt sich also vernachlässigen.
- Cloud Functions: 34 Aufrufe im Monat, bei einer Laufzeit von geschätzten 3 Minuten (180 Sekunden) ergeben sich bei einem Speicher von 128 MB 783,36 GB/Sekunden im Monat.

Resultierende Kosten der Anbieter

Bei Firebase gestaltet sich die Berechnung der Kosten relativ einfach: Es gibt nur zwei Kostenpläne, einen Free Plan und den Blaze Plan, bei dem die Kosten nach dm Pay as you go Prinzip berechnet werden. Da die Cloud Functios im Free Plan nicht verfügbar waren, befindet sich das implementierte Backend im Blaze Plan [Firebase]. Die Kosten richten sich hier also nach den benötigten Kapazitäten für die einzelnen Dienste. Die Kosten werden mittels des Firebase Kostenrechners ermittelt. Die einzigen Dienste, die Kosten verursachen sind die Realtime Database (30 US Dollar pro Monat) und der Storage (116 US Dollar pro Monat), da hier relativ große Datenmengen übertragen werden müssen. Insgesamt ergeben sich so 146 US Dollar pro Monat.
Das Amplify Framework von AWS verursacht keinerlei Kosten, die eigentlich verwendeten Dienste jedoch schon. Somit gelten für das Evender Backend die gleichen Preise wie bei Verwendung der Dienste ohne Amplify [AWS, 2022l]. Die Kosten lassen sich bei Amplify mittels eines Kostenrechners berechnen. Dabei muss die voraussichtliche Nutzung der Dienste angegeben werden, auf Basis dessen AWS dann eine Schätzung erstellt. AWS besitzt jedoch die Besonderheit, dass bei einigen Services wie der DynamoDB Vorabzahlungen geleistet werden müssen. Dies ist dann der Fall, wenn bspw. dedizierte Instanzen einer DynamoDB bereitgestellt werden, bei der on-demand Nutzung entstehen nur monatliche Kosten [AWS, 2022a]. Die Dienste, die Kosten verursachen sind AWS AppSync

(32,45 US Dollar pro Monat), die DynamoDB (15,25 US Dollar pro Monat) und der S3 Storage (3,66 US Dollar pro Monat). Insgesamt ergeben sich so 51,36 US Dollar im Monat für die Verwendung von AWS Amplify.
Die Kosten für Back4App lassen sich auf Grund des Back4App Kostenmodells leicht berechnen. Es gibt vier verschiedene Pläne zur Nutzung von Back4App: Den Free Plan, den MVP Plan, den Pay as you go Plan und den Dedicated Plan. Weiterhin gibt es die Möglichkeit, in einen personalisierten Business Plan einzutreten [Back4App, 2022b]. Diese Kostenpläne unterscheiden sich dabei vor allem, was die Kapazitäten angeht. Wenn diese Kapazitäten überschritten werden, so kommen auf die Betreiber des Backends zusätzliche Kosten hinzu. Es kämen für das oben beschriebene Nutzungsprofil entweder der Pay as you go Plan oder der Dedicated Plan in Frage. Da die Anzahl der Requests das Limit im Pay as you go Plan überschreiten, kämen hier weitere Kosten hinzu. Die Kosten des Pay as you Go Plans liegen bei 100 US Dollar pro Monat plus 2 US Dollar für je 100.000 weitere Anfragen, also 32 US Dollar monatlich zusätzlich. Das Data Storage Maximum wird nicht überschritten. Das Data Transfer Limit hingegen schon, hier kämen 100 US Dollar pro Monat zusätzlich hinzu. Auch die File Storage Kapazität wird nicht überschritten. Insgesamt ergeben sich also 232 US Dollar und somit ist dieser Plan immer noch günstiger als der Dedicated Plan (500 US Dollar pro Monat) [Back4App, 2022b].

5.1.4 Regionen

Einige Firebase Dienste wie die Realtime Database, Cloud Functions, der Storage und der Firestore müssen beim Erstellen mit einer Region konfiguriert werden, in der die Dienste bereitgestellt werden sollen. Die Realtime Database kann dabei nur in den USA, Europa und Südostasien bereitgestellt werden. Für die anderen Dienste sind die Einstellungen der Region vielfältiger. Beim Erstellen einer neuen Firebase Anwendung muss außerdem eine Standardregion ausgewählt werden, in der dann die Cloud Functions, der Firestore und der Storage bereitgestellt werden [Firebase, 2022h]. Statt einer einzelnen Region lässt sich auch eine multiregionale Einstellung wählen: Diese besteht dabei aus einer Sammlung von regionalen Zonen und ist verfügbar für Europa und Nordamerika. Es werden in jeder Region Replikate der Datensets mittels mehrere Verfügarbkeitszonen gespeichert, die entweder ein vollständiges Read-Write Replikat sind oder ein Witness-Replikat das beim Replizieren hilft. Durch diese Replikation können auch dann noch Daten bereitgestellt werden wenn eine der Regionen ausfallen sollte [Firebase, 2022h]. Wenn eine regionale Konfiguration gewählt wird, so sind die Kosten und die Latenz nied-

riger.
Beim Konfigurieren von Amplify wird wie bei Firebase auch eine Standardregion ausgewählt. In dieser ausgewählten Region werden dann alle Dienste, die eingerichtet werden, bereitgestellt [AWS, 2022d]. Eine Region ist dabei ein physischer Ort, an dem AWS Rechenzentren betreibt. Innerhalb dieser Regionen wiederum gibt es verschiedene Availability Zones. Innerhalb einer Region werden die verwendeten Dienste, sofern vom Entwickler gewünscht, auch repliziert, sodass bei Verlust einer Zone der Betrieb der eigenen Anwendung weiterhin gesichert ist [AWS, 2022n]. Dafür kann auf den AWS Replication Dienst zurückgegriffen werden. Dieser Dienst ist jedoch nicht für alle Ressourcen verfügbar [AWS, 2022k]. Die Replikation gewährleistet nicht nur bessere Ausfallsicherheit sondern erhöht auch die Fähigkeit zur Skalierung [AWS, 2022n].
Back4App bietet dem Entwickler keine große Auswahl bezüglich einer Region. Die angebotenen Regionen werden auch nicht näher konkretisiert. Im Free Plan werden alle Dienste in den USA bereitgestellt. Für den MVP und Pay as you go Plan sind zusätzliche Regionen verfügbar: Europa, Asien, der mittlere Osten und Australien. Wo genau sich die dazugehörigen Rechenzentren befinden wird hingegen nicht klar. Erst im Dedicated Plan, bei dem für die Anwendung eine eigene Infrastruktur bereitgestellt wird, lassen sich die Regionen feingranularer auswählen, da auf die verfügbaren Regionen eines Cloud Anbieters zurückgegriffen wird. Im Free Plan lassen sich also keine Regionen auswählen, genauso wenig wie Verfügbarkeitszonen oder eine Replikation. Ob dies in einem der anderen Kostenpläne der Fall ist lässt sich auf Grund der Umsetzung der Anwendung im Free Plan nicht einschätzen, da keine Informationen dazu gefunden werden konnten.

5.1.5 Zusammenfassung

Tabelle 5.1: Zusammenfassung des Überblicks

	Firebase	**Amplify**	**Back4App**
Umsetzung der Anforderungen	Erfolgreich, bis auf Empfehlungssystem, die Möglichkeit dafür wäre aber gegeben	Erfolgreich, bis auf das Empfehlungssystem, die Möglichkeit dafür wäre aber gegeben	Erfolgreich, bis auf das Empfehlungssystem, keine Möglichkeit zur Umsetzung gegeben
Punkte	. . .	. . .	. .
Architektur	Nur geringfügige Abweichungen von der Zielarchitektur	Nur geringfügige Abweichungen von der Zielarchitektur	Nur geringfügige Abweichungen von der Zielarchitektur. Besonderheit: Storage Komponente in Datenbank integriert
Punkte	. . .	. . .	. . .
Kosten	151 US Dollar pro Monat	51,36 US Dollar pro Monat	232 US Dollar pro Monat
Punkte	. .	. . .	.
Regionen	Realtime Database nur in drei Regionen verfügbar, andere Dienste hingegen auf der ganzen Welt, multiregionale Bereitstellung bietet Replikation, innerhalb einer Region mehrere Zonen	Viele verschiedene Regionen und Zonen verfügbar, Replikation muss aktiviert werden und erhöht die Kosten.	Eingeschränkte Auswahl der Regionen, nur im teuersten Plan lässt sich dies detaillierter konfigurieren, keine Möglichkeit zur Replikation, außer im teuersten Kostenplan
Punkte	. . .	. .	.

5.2 Dienste

In diesem Abschnitt wird auf die Dienste der Anbieter genauer eingegangen. Dabei werden die implementierten Dienste dargestellt und es wird das Angebot und die Vielfalt der Dienste, die der Anbieter bereitstellt, untersucht. Weiterhin wird die Integration dieser Dienste in das Frontend dargestellt.

5.2.1 Angebot und Vielfalt

Firebase bietet dem Entwickler eine große Auswahl verschiedener Dienste und Produkte, die in die eigene Anwendung integriert werden können. Diese Dienste werden dabei in drei verschiedene Kategorien eingeteilt: Build, Release and Monitor und Engage. In der Build Kategorie sind dabei die Dienste verordnet, die das eigentliche Implementieren einer Backendanwendung betreffen. In der Realease and Monitor Kategorie stellt Firebase Dienste zur Verfügung, die sich mit der Performance und Stabilität der Anwendung beschäftigen. Außerdem können mit Hilfe dieser Dienste auch neue Versionen einer Anwendung bereitgestellt werden. Hier ist bspw. auch Google Analytics zu finden, dass bei der Analyse der Nutzer und des Nutzungsverhaltens hilft. In der Engage Kategorie

sind diejenigen Dienste, die dabei helfen, das User Engagement zu erhöhen, bspw. mittels In App Messaging und A/B Testing [Firebase, 2022j]. Der Build Bereich beinhaltet dabei alle nötigen Dienste für ein Backend, dazu gehören unter anderem verschiedene Datenbanken, Authentifizierung, Cloud Functions, Storage und Hosting. Auf einige dieser Dienste wird im Folgenden noch näher eingegangen. Überdies hinaus bietet Firebase auch integrierte Erweiterungen an, bspw. für den automatisierten E-Mail Versand. Da bei Erstellung einer Anwendung in Firebase auch gleichzeitig ein Projekt in der Google Cloud erstellt wird, ist eine einfache Integration mit anderen Google Cloud Ressourcen möglich.
Auch Amplify bietet eine große Auswahl an Diensten an. Enthalten sind dabei unter anderem eine Datenbank, ein Data Storage, Cloud Functions, Authentifizierung und Analytics Dienste. Amplify trennt die einzelnen Dienste jedoch nicht nach Funktionalität wie es Firebase tut. Funktionen wie bspw. ein A/B Testing oder In App messaging fehlen jedoch, Amplify konzentriert sich auf den Kern der Funktionen, die ein Backend benötigt. Amplify bietet jedoch eine weitere interessante Funktion: Mit Amplify Studio kann ohne Code mittels eines visuellen Editors eine Fullstack Anwendung implementiert werden. Dabei können Backend Ressourcen ganz ohne Code generiert werden und beim Frontend wird eine Integration mit Figma geboten. Da Amplify ein AWS Service ist, ist auch hier eine einfache Integration mit anderen AWS Ressourcen möglich. [AWS, 2022f].
Back4App hingegen bietet eine eingeschränkte Auswahl an Diensten. Beim Erstellen einer neuen Anwendung in Back4App wird automatisch eine mongoDB bereitgestellt, die Daten einer Anwendung lassen sich jedoch auch in einem Blockchain Data Storage speichern. Weiterhin werden Cloud Functions und Web Hosting geboten, eine API für die Datenbank und weitere Funktionen wie Webhooks, Analytics und In App messaging. Weitere Funktionen sind nicht vorhanden. Dadurch, dass Back4App auf dem Open Source Framework Parse Server basiert, lassen sich aber benötigte Erweiterungen des Parse Frameworks direkt auf dem Server installieren.

5.2.2 Integration

Für alle untersuchten Anbieter muss in die mobile App ein vom Anbieter bereitgestelltes SDK integriert werden, um die Kommunikation mit dem Backend zu ermöglichen. Weiterhin müssen bei Firebase und Amplify jeweils Konfigurationsdateien in das Xcode Projekt integriert werden. Bei Firebase ist das eine .plist Datei, dies ist ein Xcode eigenes Format und steht für Information Property List. In dieser Datei ist bspw. die App-ID der

Firebase Anwendung zu finden, die generiert wird, wenn bei Firebase eine Anwendung erstellt wird. Dabei wird auch automatisch die .plist Datei erzeugt. Ohne diese Datei ist keine Verbindung zum Backend möglich. Um für Firebase das SDK zu importieren kann der Swift Package Manager verwendet werden, danach können einzelnen Libraries des SDKs verwendet werden. Damit die Verbindung zwischen der iOS App und Firebase gelingt muss im Haupteinstiegspunkt der App Code zur Initialisierung von Firebase hinzugefügt werden.
Für AWS Amplify sieht die Integration des Cloud Anbieters in die mobile Anwendung ähnlich aus. Hier gibt es jedoch kein Web Interface um die Anwendung zu erstellen, sondern es muss mit der Amplify CLI gearbeitet werden. Bei der Initialisierung von Amplify im Verzeichnis der mobilen Anwendung werden Konfigurationsdateien erstellt, die auch hier in das Xcode Projekt importiert werden müssen. Diese Dateien enthalten die Konfiguration für die erstellten Dienste, wenn die Backendkonfiguration geändert wird, werden diese Dateien automatisch angepasst. Die Amplify CLI erstellt, sobald ein Dienst eingerichtet wird, einen CloudFormation Stack. Mit diesem wird dann der eigentliche Dienst in AWS bereitgestellt. CloudFormation ist ein AWS eigener Infrastrucutre as Code Dienst. Mittels des Swift Package Managers lässt sich das Amplify SDK in das Projekt integrieren. Danach muss auch für dieses Projekt im Haupteinstiegspunkt der mobilen Anwendung eine Initialisierung von Amplify stattfinden. Im Gegensatz zu Firebase müssen hier jedoch auch alle Dienste, die der Entwickler verwenden möchten, initialisiert werden.
Für die Verwendung von Back4App muss das ParseSwift SDK in die App integriert werden. Dieses kann mittels des Swift Package Managers importiert werden. Nach dem Erstellen einer Anwendung im Back4App Web Interface sind in den Einstellungen eine Application ID und ein User Key zu finden, die im Haupteinstiegspunkt der mobilen Anwendung mittels Initialisierungscode übergeben werden müssen, damit eine Verbindung zum Backend hergestellt werden kann. Die Initialisierung einzelner Dienste ist hier nicht erforderlich.

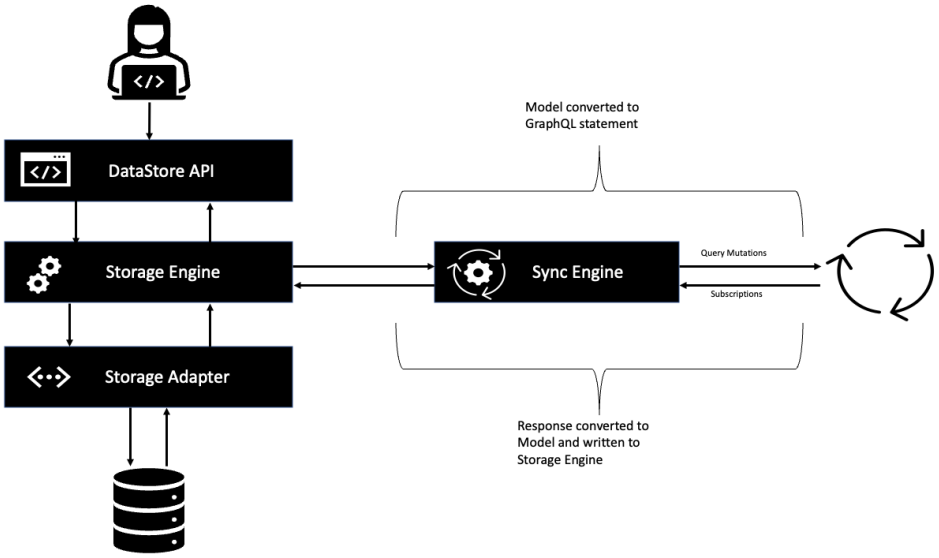
5.2.3 Persistenz

Firebase bietet für die Persistenz der Backendanwendung zwei verschiedene Datenbanktypen an: die Realtime Database und den Firestore. Sie sind jeweils für unterschiedliche Anwendungsfälle geeignet, unterstützen jedoch beide eine Echtzeit-Daten-Synchronisation. Die Realtime Database ist die erste von Firebase entwickelte Datenbank, die eine geringe

Latenz bietet. Der Firestore ist vergleichsweise neu und bietet ein anderes Datenmodell als die Realtime Database, reichhaltigere Abfragen und umfangreichere Skalierung [Firebase, 2022c]. Die Realtime Database speichert Daten als großen JSON Baum, so lassen sich gut strukturierte Daten mit geringer Komplexität einfach speichern. Der Firestore hingegen speichert Daten als Sammlung von Dokumenten, so lassen sich komplexe Daten einfacher organisieren, da weniger Normalisierung und Glättung erforderlich ist. Ein Vorteil der Realtime Database ist, dass der Verbindungsstatus eines Clients aufgezeichnet wird. So können sofort Updates der Daten bereitgestellt werden wenn ein Client wieder online kommt. Auch die Abfragen der beiden Datenbanken unterscheiden sich: Bei der Realtime Database kann entweder sortiert oder gefiltert werden. Abfragen geben immer den gesamten Teilbaum zurück, nach dem gesucht wird, inklusive aller enthaltenen Knoten. Der Firestore bietet hingegen auch komplexere Abfragen mit kombinierter Sortier- und Filterfunktion. Die Abfragen liefern jedoch keine Untersammlungen, sondern nur die Dokumente in einer bestimmten Sammlung. Der Firestore bietet außerdem erweiterte Schreiboperationen wie bspw. Arrayoperationen an. Die Realtime Database ist eine regionale Lösung und nur in einer bestimmten Region verfügbar. Die Idee dahinter ist eine niedrige Latenz. Der Firestore hingegen ist multiregional, so wird globale Skalierung und Verfügbarkeit garantiert. Die Skalierung der Realtime Database erfordert hingegen Sharding. Beide Datenbanken bieten eine Sprache zum Erstellen von Regeln, die den Zugriff zur Datenbank sichert, darauf wird im Abschnitt der Sicherheit noch näher eingegangen [Firebase, 2022c]. Firebase bietet also für die Persistenz gleich zwei potente Lösungen an, die sich in ihren Anwendungsfällen unterscheiden. Beide Datenbanken lassen sich jedoch leicht mittels des Firebase SDKs aus dem Client heraus direkt erreichen, sodass die Verwendung bspw. eines API Gateways überflüssig ist. Auch eine REST Schnittstelle wird angeboten, um die Datenbanken auch aus anderen Diensten, auch außerhalb von Firebase, zu erreichen. Für die Implementierung der mobilen App wird die Realtime Database verwendet, da die Daten nicht tief verschachtelt sind und die Synchronisation und das Offline Verhalten wichtig sind. Eingerichtet wird die Realtime Database über das Web Interface von Firebase, danach kann die Datenbank mittels einer Referenz im Code in der mobilen App verwendet werden. Mittels des Oberserver Patterns lassen sich die Daten in der Datenbank im Client laufend aktuell halten. Wenn ein Client keine Verbindung mehr zum Internet haben sollte, funktioniert die Datenhaltung weiterhin: Es wird lokal eine Version der Daten verwendet und bei erneuter Verbindung werden die Daten synchronisiert [Firebase, 2022n].

Amplify bietet zum Speichern von Daten in einer Datenbank den Datastore und eine GraphQL API an. Beide basieren auf verschiedenen AWS Diensten: AWS AppSync und auf der NoSQL Datenbank von AWS, der DynamoDB. App Sync übernimmt die Kommunikation zwischen Client und Datenbank mittels GraphQL. Der Datastore bietet außerdem einen on-device Speicher an, der alle Operationen on- und offline handhaben kann. Bei Bedarf wird dieser Speicher mittels AppSync mit der Datenbank synchronisiert. Das Datenmodell basiert dabei auch auf GraphQL. Die DynamoDB speichert die Daten dann als Schlüssel-Wert-Paare. Dadurch, dass die Kommunikation zwischen Client und Datenbank verwaltet wird, wird es dem Entwickler ermöglicht, sich auf das Modellieren der Daten zu fokussieren anstatt auf das Einrichten und Konfigurieren der Datenbank [AWS, 2022g]. Um den Datastore verwenden zu können, muss zwingend ein GraphQL Schema der Daten modelliert werden, da wie oben erwähnt die gesamte Kommunikation auf dieser Abfragesprache basiert. In diesem Schema lassen sich nicht nur einfach Datentypen beschreiben sondern auch Beziehungen zwischen zwei Modellen erstellen. Sobald in der eigentlichen mobilen Anwendung ein Datenschema erstellt wurde, wird automatisch von Amplify ein Modell erzeugt und es können Daten gelesen und geschrieben werden. Dabei werden auch Dateien erstellt, die eine API Referenz für die GraphQL Schnittstelle von AppSync beinhalten. Das Erstellen eigener Abfragen bspw. für die Cloud Functions wird so erleichtert. Die Amplify CLI stellt dann automatisch AppSync und für jedes erstellte Modell im Schema eine Tabelle in der DynamoDB bereit. Zur Laufzeit werden werden die Modelle von von einer Storage Engine verwaltet. Wenn der DataStore registriert, dass Informationen bezüglich einer Synchronisation mit App Sync vorliegen, so wird zusätzlich eine Sync Engine gestartet, die sich mit der Storage Engine verbindet, um Updates von den Modellen zu erhalten. Diese Updates werden dann zur Laufzeit in GraphQL Statements übersetzt und an die Datenbank übertragen. Daten lassen sich aber nicht nur mittels des Data Sorage abfragen und schreiben, sondern es kann auch direkt der AppSync Dienst mittels GraphQL angesprochen werden [AWS, 2022g]. Um Konflikte aufzulösen, die zwangsweise entstehen wenn viele Clients gleichzeitig auf die Daten zugreifen, bietet der DataStore verschiedene Möglichkeiten an. Standardmäßig eingestellt ist der Automerge, bei dem Informationen jeder GraphQL Abfrage genutzt werden, um das Update mit den aktuellen Daten in der Datenbank zu vergleichen. Abbildung 5.2 zeigt die oben beschriebene Funktionsweise des DataStores mit der Storage Engine, dem Storage Adapter und der Sync Engine.

Innerhalb der mobilen Anwendung lässt sich der DataStore mittels der DataStore Library des Amplify SDK verwenden. Die Abfragen sind beliebig komplex und kombinierbar:



Quelle: Eigene Darstellung in Anlehnung an: [AWS, 2022g]

Abbildung 5.2: Data Store Architektur

Abfragen können Filter und eine Sortierung beinhalten. Auf die Sicherheit der Datenbank wird im weiteren Verlaufe dieses Kapitels noch näher eingegangen.

Back4App stellt mit dem Erstellen einer neuen Anwendung im Web Interface automatisch eine mongoDB bereit. Diese Datenbank enthält außerdem schon zwei Klassen: eine User Klasse und eine Role Klasse. Die mongoDB ist eine dokumentenorientierte NoSQL Datenbank, Back4App lässt sich jedoch auch mit einer relationalen PostgreSQL Datenbank verwenden. Die Daten der Datenbank werden jedoch im Web Interface von Back4App optisch wie ein Spreadsheet angezeigt und lassen sich auf relationale Weise abfragen. Durch Kombination von einzelnen Abfragen lässt sich bspw. ein Join verschiedener Klassen erzeugen [Back4App, 2022j]. Die Datenbank bietet keine automatische Echtzeitdatenverarbeitung, dies lässt sich jedoch mittels der von Back4App angebotenen Live Queries umgehen. Grundlage sind dabei Websockets, die Clients benachrichtigen, sobald Änderungen in der Datenbank auftreten. Clients können auch nur bei den Klassen, bei denen es gewünscht ist, eine Subscription vornehmen. Diese Funktion muss jedoch gesondert aktiviert werden (auch für einzelne Tabellen) und erfordert eine zusätzliche Authentifizierung mittels Telefonnummer [Back4App, 2022a]. Die Daten, die mittels des Parse SDK von einem Client in der Datenbank gespeichert werden, basieren auf dem *Parse.Object*

des Parse Frameworks. Jedes Parse Objekt beinhaltet einen Key-Value Speicher von JSON kompatiblen Daten für die Felder dieses Objektes. Die Daten sind bei Back4App schemalos, das heißt, es muss nicht im Vorhinein festgelegt werden, wie die Daten strukturiert sind. Auch Beziehungen zwischen einzelnen Parse Objekten werden unterstützt [Back4App, 2022i]. Back4App bietet für das Erstellen von Abfragen einen angenehmen Service: Back4App erstellt für jede Klasse, die in der Datenbank erstellt wird, automatisch eine API Referenz. In dieser sind vorgefertigte Abfragen, bspw. für das Erstellen eines neuen Objektes, zu finden. Die Abfragen für Daten in der Datenbank sind dabei sehr umfangreich, Filter und Sortierungen lassen sich einfach miteinander kombinieren.

5.2.4 Authentifizierung

Um einen User in die mobile Anwendung einzuloggen, wird ein Dienst zur Authentifizierung benötigt. Dieser Dienst benötigt Credentials vom Nutzer, bspw. eine Kombination aus einer E-Mail Adresse und einem Passwort oder einen Token eines anderen Identitätsanbieters. Der Dienst verifiziert diese dann und gewährt dem Nutzer Zugriff auf entsprechende Daten und Dienste.

Firebase stellt zum Authentifizieren von Nutzern den Authentication Dienst bereit. Dieser Dienst stellt eine Ende-zu-Ende Lösung zum Behandeln von Identitäten mittels Backend Diensten und SDKs bereit und unterstützt E-Mail und Passwort Authentifizierung, Telefon Verifizierung und einen Login mittels Google, Facebook, Twitter, Apple und anderen Identitätsanbietern. Außerdem bietet dieser Dienst ein schon vorgefertigtes User Interface, das in die eigene Anwendung integriert, erweitert und angepasst werden kann. Dabei wird der gesamte Authentication Flow von Firebase bereitgestellt [Firebase, 2022f]. Mittels des Firebase SDKs wurde für die mobile App jedoch ein selbst erstellter Login Flow implementiert. Im Firebase Web Interface muss der entsprechende Identitätsanbieter aktiviert werden und danach kann mit dem FirebaseAuth Package aus dem SDK die Authentifizierung erstellt werden. Für die Implementierung des Facebook Logins hingegen ist ein Facebook Developer Account nötig. Innerhalb dieses Accounts muss eine App erstellt werden, die dann die für die Verbindung zu Firebase benötigten Informationen bereitstellt. Weiterhin wird das Facebook iOS SDK zur Implementierung benötigt und es muss die Konfigurationsdatei des Xcode Projekts angepasst werden. Nachdem sich ein Nutzer über Facebook eingeloggt hat muss im Code noch ein expliziter Login bei Firebase erfolgen, dies gelingt mittels des Tokens der nach dem Facebook Login bereitsteht. Auch für den Login mittels Google wird ein SDK benötigt. Für weitere Funktionalitäten

kann überdies hinaus eine Integration mit der Google Cloud Identity Plattform erfolgen. Dieses Upgrade fügt einige neue Features hinzu: Multi Factor Authentifizierung, Blocking Functions (Ausführung von selbst erstelltem Code beim Login), SAML und OpenIDC Provider, User Activity Logging, Multi Tenency, Enterprise Support und weitergehende SLAs, was aber auch zu erhöhten Kosten führt [Firebase, 2022f].

Ampify verwendet als Authentifizierungsdienst AWS Cognito. Cognito ist ein Verzeichnisdienst: Dieser kümmert sich um die Registrierung neuer Nutzer, um das Einloggen und auch um die Account Wiederherstellung. Amplify interagiert über eine Schnittstelle mit Cognito, um Nutzerinformationen oder auch föderierte Identitäten von Drittnabietern wie Goole in Nutzer Pools zu speichern. Dabei wird auch der Zugang zu anderen Ressourcen in AWS geregelt. Bei der Konfiguration lässt sich festlegen, auf welche Ressourcen zugegriffen werden soll und die Amplify CLI erstellt dabei automatisch die entsprechenden Zugriffsrichtlinien [AWS, 2022c]. Grundsätzlich erlaubt AWS dabei zwei Möglichkeiten zur Authentifizierung: Mittels eines erstellten Tokens oder mittels einer signierten URL. Amplify verwendet zum Authentifizieren der Nutzer einen Token, die signierten URLs werden hingegen verwendet, um den Zugriff der Dienste untereinander zu authentifizieren und zu autorisieren. Zum Einrichten des Authentifizierungsdienstes muss der Dienst mit der Amplify CLI zum Backend hinzugefügt werden, währenddessen lässt er sich umfassend konfigurieren, bspw. für die Einrichtung eines Identitätsanbieters wie Facebook. Innerhalb der Anwendung funktioniert die Authentifizierung auf die gleiche Weise wie bei den anderen Anbietern: Mittels des Amplify SDKs und der Auth Library des SDKs lässt sich der Login, die Registrierung oder auch der Logout umsetzen. Auch bei Amplify wird für den Facebook Login das Facebook SDK benötigt, genauso wie ein Entwickler Account und eine erstelle App im Facebook Developer Bereich. Dabei wird von Facebook eine App ID und ein Secret erstellt, die beim Einrichten des Authentifizierungsdienstes für Facebook in der Amplify CLI übergeben werden müssen. Weiterhin wird für den Google Login ein Account bzw. ein Projekt bei der Google Cloud Platform benötigt. Dort muss ein Client erstellt werden, dessen Informationen wiederum für die Amplify CLI benötigt werden. Auch bei Amplify lässt sich der Flow der Authentifizierung anpassen und bietet ein vorgefertigtes User Interface für die Authentifizierung an. Zusätzlich lässt sich eine Multi Faktor Authentifizierung aktivieren.

Auch Back4App bietet die Möglichkeit, Nutzer der Anwendung zu authentifizieren. Bei der Erstellung einer Anwendung in Back4App wird automatisch eine Datenbank erstellt, die eine User Klasse enthält. Diese wird auch verwendet, um Nutzer zu authentifizieren. Beim Login oder bei der Registrierung eines Nutzers wird außerdem eine Session Klasse

erzeugt, die alle aktuell eingeloggten Nutzer enthält und deren aktuelle Sitzung widerspiegelt. Grundlage des Authentifizierungs Flows sind also Nutzer Objekte des Parse Frameworks. Es muss keinerlei Konfiguration erfolgen, sondern es kann direkt in der mobilen Anwendung mit dem Implementieren der Funktionen für Registrierung oder Login begonnen werden. In der mobilen Anwendung wird dann zur Authentifizierung die *Parse.User* Klasse des Parse SDKs verwendet, mittels dieser Klasse lassen sich dann direkt Sign In oder Sign Up Funktionen aufrufen. Diese Klasse implementiert jede Funktionalität, die für die Authentifizierung von Nutzern benötigt wird [Back4App, 2022n]. Für das Einloggen und Registrieren mittels einer Facebook Identität wird auch für Back4App das Facebook iOS SDK benötigt. Auch das Einrichten einer App innerhalb eines Facebook Entwickler Accounts ist analog zu den anderen Anbietern. Die Informationen der App im Facebook Account werden wiederum in die Konfigurationsdatei des Xcdoe Projekts integriert [Back4App, 2022l]. Für den Google Login wird das Google SignIn SDK verwendet. Wie für die Authentifizierung bei Amplify muss in einem Google Cloud Account dafür ein Projekt und ein OAuth Client erstellt werden, die Informationen dieses Clients werden dann in die Konfigurationsdatei der mobilen App integriert. Bei beiden Möglichkeiten zur Authentifizierung mit einem Identitätsanbieter muss gesondert nach dem Login beim Identitätsanbieter ein Login mittels des Parse SDKs erfolgen, dafür wird von den Identitätsanbietern nach erfolgreicher Anmeldung ein Token bereitgestellt.

5.2.5 Cloud Functions

Cloud Functions sind der elementare Bestandteil einer Serverless Architektur, nicht nur bei einer FaaS Plattform sondern auch in einem BaaS. Sie ermöglichen es den Entwicklern, eigene Geschäftslogik abseits von den standardisierten Services eines Anbieters zu implementieren. Im Rahmen der umgesetzten Anwendungen dieser Arbeit wurden Cloud Functions erstellt, die Uploads in den Storage umsetzten, das Speichern von Daten in der Datenbank, das Senden eines Newsletters und auch das Abfragen einer API für die Daten zu den Veranstaltungen.

Quellsysteme

Als erstes Quellsystem zum Extrahieren von Veranstaltungen war ursprünglich Eventbrite ausgewählt worden, die entsprechende API zum Abfragen von Veranstaltungen ist

aber inzwischen inaktiv, deshalb wird ersatzweise die Ticketmaster API verwendet. Mittels eines API Keys, der beim Erstellen eines Developer Accounts bereitgestellt wird, lassen sich Abfragen an die Ticketmaster API senden. In der paginierten Antwort sind dann Veranstaltungen mit dazugehörigen Informationen enthalten. Die API bietet dabei einiges an Funktionalität: Es lassen sich Veranstaltungen suchen und mit verschiedenen Parametern filtern, es lassen sich Details zu einzelnen Veranstaltungen oder Veranstaltungsorten abfragen oder auch Vorschläge generieren. Dabei ist die API jedoch auch limitiert: Pro Tag werden höchstens 5.000 Abfragen unterstützt. Außerdem werden pro Abfrage höchstens 1.000 Elemente zurückgegeben.

Implementierung der wichtigsten Cloud Functions

Die Funktion, mit der Daten zu Veranstaltungen in der Persistenzkomponente gespeichert werden, ist die zentrale Cloud Function des Backends. Dabei wird mittels eines Axios Clients eine HTTP Anfrage an die Ticketmaster API gesendet. Das Resultat der Anfrage ist ein JSON Array, das es zu analysieren gilt, um die Daten der Veranstaltungen zu erhalten. Die einzelnen erhaltenen Veranstaltungen werden dann mittels des entsprechenden SDKs in der Datenbank des Backends gespeichert. Zusätzlich wird mit einer Hilfsfunktion das Bild zu der Veranstaltung aus dem Content Delivery Network von Ticketmaster heruntergeladen und in der Storage Komponente gespeichert. Alle implementierten Funktionen bei den Anbietern ähneln sich stark. Um die Funktionen zu triggern wird eine HTTP Anfrage verwendet. Eine weitere wichtige Funktion des Backends betrifft das Versenden des Newsletters. Hier unterscheiden sich die Anbieter jedoch, was das Auslösen der Funktionen angeht. Bei Firebase wird dafür der Firestore verwendet: Beim Erstellen eines neuen Dokumentes mit entsprechenden Feldern für den Betreff und den Text wird automatisch der Versand des Newsletters an diejenigen Nutzer ausgelöst, die den Newsletter abonniert haben. Bei Amplify wird dafür eine Tabelle in der DynamoDB verwendet und bei Back4App eine Mail Klasse in der mongoDB. Die Daten des auslösenden Events beim Erstellen eines neuen Objektes, egal ob im Firestore, in der DynamoDB oder in der mongoDB, werden an die Funktion übergeben. Mit einer Datenbankabfrage wird eine Liste der Nutzer erstellt, die den Newsletter abonniert haben und mittels des Nodemailer Paketes werden die Daten des auslösenden Events dann zur Erstellung einer E-Mail verwendet.

Vergleich der Plattformen der Anbieter

Die Cloud Functions von Firebase lassen sich nicht im kostenlosen Plan von Firebase verwenden, es muss zwingend vom Spark in den Blaze Plan gewechselt werden. Die Funktionen werden dann nach dem Pay as you go Prinzip abgerechnet. Der Cloud Function Dienst ist bei Firebase genau wie bei allen anderen Anbietern ein Serverless Framework, das es ermöglicht, Funktionen als Reaktion auf vorher festgelegte Events auszuführen. Solche Events können dabei bspw. HTTP Anfragen oder Updates in einer Datenbank sein. Der Code einer Funktion wird in der Google Cloud gespeichert und läuft dann bei Bedarf in einer containerisierten Umgebung. Das Bereitstellen einer Funktion muss bei Firebase zwingend mittels der Firebase CLI erfolgen, ein Web Interface gibt es für diesen Dienst nicht. Erhöht sich die Last, also die Anzahl der getriggerten Funktionen, so skaliert Google automatisch. Jede Funktion läuft dabei aber in einer eigenen isolierten Umgebung. Der Lebenszyklus einer Funktion gestaltet sich folgendermaßen: Eine Funktion wird geschrieben und es wird ein Trigger ausgewählt. Danach wird die Funktion als zip Datei in einem Storage Bucket gespeichert. Zusätzlich wird ein Repository erstellt, das alle Images der geschriebenen Funktionen mit dem zugehörigen Trigger enthält. Wenn ein Event Provider ein Event registriert wird die Ausführung des Images angestoßen: Dabei wird ein Snapshot der Daten des Events an die Funktion übertragen. Beim Updaten oder Löschen einer Funktionen werden alte Versionen aus Bucket und Registry gelöscht [Firebase, 2022d]. Um mit anderen Firebase Diensten zu interagieren, wird für die Functions das Firebase Admin SDK angeboten. Mit diesem lässt sich auf andere Dienste im Firebase Kosmos zugreifen. Die Cloud Functions werden in der Standard Region des Firebase Projektes ausgeführt. Die Funktionen lassen sich jedoch nur in Typescript oder Javascript schreiben, andere Sprachen und Laufzeitumgebungen werden nur im Cloud Functions Bereich der Google Cloud unterstützt. Für die Cloud Functions existieren weiterhin einige Limiterungen: Die Laufzeit einer Funktion lässt sich auf höchstens 9 Minuten setzen, der maximale Speicher bertägt 8 GB (entspricht einer 4,8 GHZ CPU). Bei Überschreitung einer dieser beiden Grenzen bricht die Funktion ab. Weiterhin können pro Region höchstens 1.000 Funktionen bereitgestellt werden, eine Funktion darf als zip Datei 100 MB nicht überschreiten und eine Anfrage, Antwort oder ein Event darf 10 MB nicht überschreiten [Firebase, 2022l]. Das Skalieren einer Funktion geschieht automatisch, dabei werden folgende Faktoren zur Evaluation einbezogen: Die Zeit der Ausführung der Funktion, die Zeit der Initialisierung bei einem Cold Start, die Limits der Lese- und Schreiboperationen, die Fehlerrate einer Funktion und Umweltfaktoren, wie die regionale Last, und die Kapazitäten in einem Datenzentrum. Wenn eine Funkti-

on die oben genannten Grenzen erreicht, ist die Ressource nicht verfügbar sein, bis alle eingegangenen Aufrufe der Funktion abgearbeitet wurden. Firebase bietet zwar kein Web Interface zum Bearbeiten einer Cloud Function an, aber in der Firebase Konsole lassen sich die Funktionen überwachen. Dort sind die Nutzung und die Logs einer jeden Funktion einsehbar. Auf das Monitoring wird im Folgenden noch näher eingegangen, genauso wie auf das Debuggen der Cloud Functions. Um das Cold Start Problem einer Function im Rahmen zu halten bietet Firebase die Möglichkeit, das Verhalten der Skalierung anzupassen. Normalerweise skaliert eine Funktion nach den oben beschriebenen Parametern und dies kann dazu führen, dass keine aktiven Instanzen einer Funktion vorhanden sind. Dies führt dann zu einem Cold Start bei einem erneuten Aufruf. Um dies zu verhindern, kann eine minimale Anzahl an Container Instanzen angegeben werden, die *warmgehalten* werden. Auch eine maximale Anzahl lässt sich hier spezifizieren [Firebase, 2022k].

Auch für das mit Amplify implementierte Backend sind Cloud Functions ein wichtiger Bestandteil. Amplify Functions verwendet AWS Lambda im Hintergrund. AWS Lambda bietet eine größere Auswahl an Sprachen und Laufzeitumgebungen an als die anderen Anbieter: Es wird Python, Java, Type-/Javascript, C# und Go angeboten. Auch Lambda Funktionen werden durch bestimmte Events getriggert. Die für die Ausführung der Funktion benötigten Ressourcen werden dabei vom Dienst selbst verwaltet. Trigger können dabei alle Arten von Events sein, von HTTP Anfragen bis hin zu Events einer Datenbank und die Arten der Trigger sind weitaus umfangreicher als bei Firebase oder Back4App und es können sogar Trigger für das Monitoring eingerichtet werden. Für HTTP Trigger wird jedoch zwingend ein API Gateway benötigt, was zusätzliche Kosten verursacht. AWS Lambda bietet weiterhin die Möglichkeit zur Nutzung von Lambda Layers. Mit den Lambda Layers lassen sich genutzte Bibliotheken und verwendeter Code als zip Datei verpacken, sodass diese Elemente dann von anderen Funktionen einfach wiederverwendet werden können. So wird vermieden, dass redundanter Code entsteht oder Bibliotheken erneut hochgeladen werden. Der Code lässt sich entweder als zip Datei hochladen oder es kann direkt ein Container Image bereitgestellt werden [AWS, 2022h]. Auch AWS Lambda bietet automatische Skalierung an. Grundlage für die Anzahl der laufenden Container Instanzen ist dabei die Anzahl der Anfragen. Um einen Cold Start zu verhindern bietet Lambda die Möglichkeit, *Provisioned Concurrency* zu aktivieren. Dadurch hält Lambda die Funktionen initialisiert und bereit und dabei wird im zweistelligen Millisekunden Bereich auf ein eintreffendes Event reagiert [AWS, 2022h]. Die Kosten werden nach dem Pay as you go Prinzip kalkuliert, wenn keine Instanzen einer Funktion laufen so entstehen auch keine Kosten. Funktionen können bei Amplify mittels der CLI erstellt werden,

der Code kann jedoch nach dem Bereitstellen einer Funktion auch in der AWS Konsole verändert werden. Hier lassen sich auch die Laufzeit und der Speicher einer Funktion anpassen. Secrets könenn wie bei Firebase über die CLI hinzugefügt werden. Außerdem können auch Umgebungsvariablen bereitgestellt werden. Auch lässt sich beim Einrichten einer Lambda Funktion mit der Amplify CLI festlegen, auf welche Ressourcen die Funktion zugreifen darf. So wird sichergestellt, dass keine unnötige Sicherheitslücken entstehen. Lambda besitzt keine Limitierungen bezüglich der maximalen Anzahl an Funktionen oder die Anzahl der Anfragen. Das Maximum der Ausführungszeit beträgt 15 Minuten und der Speicher der Container Instanz beträgt maximal zehn GB. Lambda lässt jedoch zu jeder Zeit von jeder Funktion höchstens 1.000 Instanzen parallel laufen.

Back4App bietet dem Entwickler genau wie AWS Amplify und Firebase die Möglichkeit, Geschäftslogik mit Cloud Functions umzusetzen. Dabei laufen die Funktionen als Reaktion auf Events in einer verwalteten Umgebung. Back4App bietet jedoch als einzige Sprache Javascript an. Dndere Sprachen und Laufzeitumgebungen werden nicht unterstützt. Der eigentliche Code wird wie bei den anderen Anbietern in einem Storage Bucket gespeichert und dann bei Bedarf ausgeführt. Die Cloud Functions lassen sich direkt in der Web Konsole von Back4App bearbeiten. Der Code Editor bietet dem Entwickler jedoch keine Unterstützung was bspw. die Syntax des Codes angeht. Somit entstehen beim Bearbeiten in der Konsole leicht Fehler, die erst beim Ausführen der Funktion auffallen. Die Funktionen lassen sich nach Bearbeitung einfach mittels eines Buttons bereitstellen und sind sofort verfügbar. Außerdem lassen sich Dateien auch lokal erstellen und hinterher mittels der Konsole hochladen, sie müssen also nicht zwingend in der Web Konsole an sich geschrieben werden. Die Funktionen können auf mehrere Arten getriggert werden: Durch Aufrufe aus einer Anwendung, mittels einer REST API oder auch durch ein Update eines Datensatzes in der Datenbank [Back4App, 2022e]. Die Funktionen teilen sich in zwei verschiedene Arten: Zum Einen gibt es generische Funktionen, die auf verschiedene Arten von Events wie HTTP Anfragen reagieren und zum Anderen gibt es Trigger Funktionen, die nur auf Events reagieren, die innerhalb des Parse Servers auftreten. Back4App bietet außerdem sogenannte Validatoren an. Damit können die Daten eines Events, das als Trigger einer Funktion dient, vor dem Ausführen validiert werden [Back4App, 2022o]. Weitere Informationen bezüglich eventueller Limitierungen oder des Cold Start Verhaltens ließen sich nicht finden. Da nur Javascript unterstützt wird, bietet Back4App für die Laufzeitumgebung Node.js schon bereits vorinstallierte Module an, die sich so direkt verwenden lassen.

5.2.6 Storage

In diesem Abschnitt wird auf die angebotenen Dienste der Anbieter zur Umsetzung der Storage Komponente eingegangen. Mittels eines Storage lassen sich die in einer Anwendung verwendeten Dateien speichern und abrufen.

Der Cloud Storage Dienst von Firebase besteht aus einem simplen Objektspeicher, der Objekte in einem Bucket der Google Cloud speichert. Somit sind die gespeicherten Dateien in Firebase und auch in der Google Cloud verfügbar. Dies bietet die Möglichkeit, Google Cloud Storage APIs zu verwenden, die das verarbeiten der Dateien ermöglicht. Die Skalierung findet automatisch statt. Der Zugriff kann mittels des Authentifizierungsdienstes gesichert werden, aber auch der Storage Service bietet wie die Datenbanken eine deklarative Regelsprache zum Definieren der Zugriffsrichtlinien [Firebase, 2022e]. Dateien lassen sich mittels des Firebase SDKs up- oder downloaden und über eine Referenz des Buckets mittels des Firebase Admin SDKs in einer Cloud Function verwenden. Der Storage Dienst bietet auch eine Integration für die Cloud Functions, bspw. kann der Upload eines neuen Objektes eine Funktion triggern.

Das Amplify Storage Modul bietet den Entwicklern und Nutzern gleichermaßen eine einfache Möglichkeit, Dateien zu speichern. Um den Storage von Amplify zu verwenden muss zwingend der Authentifizierungsdienst eingerichtet sein, dies garantiert, dass nur berechtigte Nutzer Zugriff auf den Storage bzw. auf die entsprechenden Dateien erhalten. Die Storage Kategorie von Amplify bietet zwei verschiedene Optionen: Es kann ein Bucket im Simple Storage Service (S3) von AWS erstellt werden oder es wird eine Schnittstelle zur DynamoDB erstellt mittels der dann Daten gespeichert werden können, die unabhängig vom eigentlichen Datenmodell sein sollen. Auch bereits existierende Buckets können in die Storage Komponente integriert werden. S3 speichert Daten dabei als Objekte in einem Container Bucket. Eine einzige Datei besteht aus den eigentlichen Daten und den Metadaten der Datei [AWS, 2022p]. Beim Erstellen des Storage mit der Amplify CLI werden auch automatisch Richtlinien für den Zugriff auf diese Komponente erstellt. Auch Trigger für Lambda Funktionen lassen sich in Bezug auf die Storage Komponente erstellen. Auf den Storage kann in der mobilen Anwendung mittels des Amplify SDKs zugegriffen werden. Für den Zugriff aus einer Lambda Funktion heraus wird das AWS SDK benötigt.

Die Storage Komponente von Back4App unterscheidet sich von den Konzepten von Firebase und Amplify. Mittels des Parse SDKs lässt sich ein Objekt des *Parse.File* Typs

erstellen, das dann mit einer Funktion in der Storage Komponente gespeichert wird. Die Storage Komponente wird dabei von Back4App selbst nicht näher beschrieben. Die zu speichernde Datei muss zwingend mit einem bestehenden Objekt der Datenbank assoziiert werden, ansonsten ist diese Datei durch eine Abfrage nicht mehr auffindbar und wird zu einer *orphan File*. Somit sind die gespeicherten Objekte direkt in der Datenbank zu sehen, einen eigenen Dienst für den Storage gibt es nicht [Back4App, 2022k]. Um Dateien aus der Storage Komponente zu laden genügt es jedoch nicht, das entsprechende Objekt der Datenbank zu laden. Die Daten der Datei sind in dem Ergebnis einer Abfrage nicht enthalten. Es muss zwingend ein weiterer Schritt erfolgen: In der Abfrage eines Objektes der Datenbank sind die Metadaten für die Datei enthalten, diese beinhalten auch die URL der Datei im eigentlichen Storage von Back4App. Mittels dieser URL kann die Datei dann abgerufen werden [Back4App, 2022k].

5.2.7 Empfehlungssystem

Bei keinem der Anbieter ließ sich ein System zum Generieren von Empfehlungen umsetzen, was daran liegt, dass bei allen Providern Daten des Verhaltens der Nutzer benötigt werden. Zu Beginn der Arbeit bestand die Hoffnung, dass Empfehlungen nicht nur auf Basis des Nutzerverhaltens sondern auch Item-basiert, also basierend auf den Veranstaltungen, generiert werden können. Die Anbieter geben dem Entwickler jedoch nur die Möglichkeit, Empfehlungen aus Basis des Nutzerverhaltens zu generieren. Deshalb wird im Folgenden näher auf die angebotenen Möglichkeiten des jeweiligen Anbieters zur Generierung von Empfehlungen eingegangen.

Firebase bietet in der Build Kategorie auch einen Dienst für Machine Learning an. Dieser ermöglicht es, selbst erstellte oder auch vorgefertigte Machine Learning Modelle, wie bspw. Bilderkennung, in die eigene Anwendung einzubinden. Empfehlungen werden mit einem entsprechenden Modell direkt auf dem Client erzeugt, was durch Einbinden des Modells in den Client Code geschieht. Firebase bietet also generell die Möglichkeit, Empfehlungen zu generieren. Um dies umzusetzen muss Google Analytics in die Anwendung integriert werden, um Daten der Nutzer bezüglich ihres Verhaltens zu sammeln. Die so entstehenden Analytics Daten lassen sich automatisch in Google Big Query importieren, den Big Data Service der Google Cloud. Mit den Daten lässt sich nun mittels eines Notebook Services wie Google Colab und mit Python ein Machine Learning Modell erzeugen, welches als Datei in Firebase importiert werden kann. In der mobilen Anwendung lässt sich das Modell wiederum importieren und kann mit entsprechendem Input (Verhalten

des Nutzers) Empfehlungen erzeugen [Firebase, 2022a].
Auch Amplify bietet einen Dienst zur Einbindung von Machine Learning in die eigene Anwendung an. Dieser Dienst heißt Predictions und bietet eine Integration mit schon bestehenden Machine learning Modellen wie bspw. Texterkennung. Für das Generieren von Empfehlungen muss jedoch Amazon Personalize verwendet werden, das nicht Teil von AWS Amplify ist. Empfehlungen werden entweder auf Basis von Analytics Daten, die als Events in der mobilen Anwendung getrackt werden, erzeugt oder mittels historischer Daten [AWS, 2022m]. Da Evender über keine historischen Daten verfügt, wird also auch bei Amplify Analytics verwendet, um das Nutzerverhalten zu tracken und Daten zu generieren. AWS übernimmt bei ausreichender Größe des Datensets das Trainieren des Modells, somit entfällt das eigenständige Erstellen des Modells wie bei Firebase. Für den Personalize Service existiert jedoch kein SDK für Swift, somit müssen in einer iOS Anwendung die Empfehlungen mit einer selbst implementierten API abgefragt werden [AWS, 2022b].
Back4App hingegen bietet keine integrierte Lösung für Machine Learning an. Mittels des Analytics Dienstes von Back4App lässt sich jedoch zumindest das Verhalten der Nutzer tracken und speichern. Mit selbst erstellten Events können interessante Interaktionen in der mobilen Anwendung an den Analytics Dienst gesendet werden [Back4App, 2022m]. Die so gesammelten Daten lassen sich exportieren und müssen mit einem anderen Dienst verarbeitet werden, Empfehlungen müssten über eine API bereitgestellt werden.

5.2.8 Zusammenfassung

Tabelle 5.2: Zusammenfassung der Dienste

	Firebase	**Amplify**	**Back4App**
Angebot Vielfalt	Große Auswahl an Diensten, integrierte Erweiterungen, Integration mit der Google Cloud	Integration mit AWS, mittelgroße Auswahl an Diensten, Amplify Studio zur Erstellung ganzer Fullstack Anwendungen	Wenige angebotene Dienste, aber durch das Open Source Parse Framework einfache Erweiterung möglich
Punkte	. . .	. .	.
Integration	Einfache Integration mittels SDK, wenig Konfigurationsaufwand	Einfache Integration, höherer Konfigurationsaufwand beim Erstellen der Dienste	Sehr einfache Integration der Dienste in die mobile App, Dienste müssen nicht gesondert eingerichtet werden
Punkte	. . .	. .	. . .
Persistenz	Verschiedene Datenbanktypen zur Auswahl, Daten in Echtzeit verfügbar und Offline Synchronisation, Firestore bietet automatische Skalierung	Ausgereiftes API Modell mit App Sync und GraphQL, Data store unterstützt Offline Synchronisation, Modellierung der Daten im Vordergrund, automatisch generierte API Referenz	NoSQL und relationale Datenbank zur Auswahl, Live Queries für Echtzeit Datenbankabfragen, automatisch generierte API Referenz
Punkte	. . .	. . .	. .
Authentifizierung	Vorgefertigtes User Interfaces, MFA, Integration mit Google Cloud Identity	Vorgefertigtes User Interface für mehrere Plattformen, Autorisierung mit automatisch erstellten Richtlinien, MFA	Datenbank wird zur Authentifizierung genutzt, keine Einrichtung nötig, MFA
Punkte	. . .	. . .	. . .
Cloud Functions	Nur im teureren Blaze Plan verfügbar, automatische Skalierung, nur Type- und Javascript, Warmhalten von Funktionsinstanzen	Automatische Skalierung, diverse Laufzeitumgebungen, große Auswahl an Triggern, Warmhalten von Funktionsinstanzen, Online Code-Editor	Nur Javascript verfügbar, keine Skalierung im kostenlosen Plan, kein Warmhalten von Funktionsinstanzen, zu einfacher Online Code-Editor
Punkte	. .	. . .	.
Storage	Objektspeicher, Integration mit der Google Cloud, automatische Skalierung	Objektspeicher, automatische Skalierung, unbegrenzter Speicherplatz	Integriert in die Datenbank, Dateien ohne Referenz nicht mehr auffindbar
Punkte	. . .	. . .	. .
Empfehlungssystem	Machine Learning integriert, selbst erstellte Modelle sind möglich, Analytics Dienst zum Sammeln von Daten	Integriertes Empfehlungssystem, auch Integration von Machine Learning Modellen möglich, Analytics Dienst zum Sammeln von Daten	Kein integriertes Empfehlungssystem oder Machine Learning
Punkte	. .	. . .	.

5.3 Leistungseffizienz

Für den Abschnitt der Leistungseffizienz werden Messungen vorgenommen: Mittels der Instruments Anwendung, eines in Xcode integrierten Monitoring Tools, lassen sich Metriken einer iOS Anwendung erheben. Dabei kann das Ausführen beliebiger Code Passagen gemessen werden. Für das Messen der Abfragezeiten wird bewusst auf das Monitoring der BaaS Anbieter verzichtet. Zum Einen bietet Back4App kein umfangreiches Monitoring der Datenbank an, zum Anderen stellt sich die Frage der Vergleichbarkeit: Die Round

Trip Time für eine Abfrage kann bei verschiedenen Anbietern auf Grund unterschiedlicher Monitoring Methoden variieren. Deshalb wird hier ein unabhängiges Tool eingesetzt, das sich zwar auf die Code Ausführung an sich bezieht, aber dafür bei jedem Anbieter exakt gleich eingesetzt werden kann. Gemessen wird die Dauer der jeweiligen Operation bei 5.000 Veranstaltungen in der Datenbank und die Ausführung jeder Funktion wird mehrmals (100 Mal) gemessen, um die Auswirkungen von Messfehlern und eventuellen Ausreißern möglichst gering zu halten. Getestet werden folgende Operationen: Erstellen eines neuen Nutzers, Updaten der Favoritenliste eines Nutzers, Datenbanksuche mittels des Nutzernamens, Lesen der Favoriten eines Nutzers, Suchen einer Veranstaltung in der Datenbank und das Laden eines Bildes aus dem Storage. Somit werden alle für die Leistungseffizienz relevanten Komponenten des Backends untersucht. Die Leistung der implementierten Cloud Functions findet hier bewusst keine Beachtung: Auf Grund der unterschiedlichen Umsetzung der Funktionen bei jedem Anbieter ist keine Vergleichbarkeit gegeben. Auf diese Tatsache wird im Rahmen der Evaluation noch näher eingegangen.

Tabelle 5.3: Messung der Laufzeiten verschiedener Funktionen

	Firebase	**Amplify**	**Back4App**
Nutzererstellung	Minimum: 1,26 s Average 1,35 s Maximum: 1,47 s	Minimum: 115,03 ms Average: 144,70 ms, Maximum: 183,68 ms	Minimum: 227,74 ms Average: 277,43 ms, Maximum: 489,64 ms
Update der Favoriten eines Nutzers	Minimum: 26,47 ms Average: 52,61 ms Maximum: 289,99 ms	Minimum: 7,86 ms Average: 13,46 ms Maximum: 19,30ms	Minimum: 281,10 ms Average: 374,15 ms Maximum: 819, 35 ms
Nutzersuche	Minimum: 42,95 ms Average: 76,58 ms Maximum: 516,22 ms	Minimum: 2,68 ms Average: 6,48 ms Maximum: 19,30 ms	Minimum: 132,70 ms Average: 165,06 ms Maximum: 520,09 ms
Laden der Favoriten	Minimum: 4,55 ms Average: 63,90 ms Maximum: 275,26 ms	Minimum: 0,611 ms Average: 2,05 ms Maximum: 51,42 ms	Minimum: 135,10 ms Average: 171,92 ms Maximum: 429,78 ms
Suche nach einer Veranstaltung	Minimum: 1,83 ms Average: 281, 26 ms Maximum: 614,79 ms	Minimum: 0,31046ms Average: 0,43850 ms Maximum: 1,6 ms	Minimum: 129.61 ms Average: 229,10 ms Maximum: 607,78 ms
Laden eines Bildes aus dem Storage	Minimum: 238,54 ms Average: 1,21 s Maximum: 3,55 s	Minimum: 234,52 ms Average: 388,80 ms Maximum: 630,93 ms	Minimum: 25,11 ms Average: 83,91 ms Maximum: 378,53 ms
Punkte	. .	. . .	. .

Bei Betrachtung der Messung der Ausführungszeiten verschiedener Datenbank- und Storage Operationen zeigt sich, dass Back4App in den meisten Fällen am langsamsten ist. Die Unterschiede sind dabei in meisten Fällen nicht sehr groß, außer im Bereich der Nutzererstellung mit dem Authentifizierungsdienst und dem Laden eines Bildes aus dem Storage. Hier ist Firebase in beiden Fällen äußerst langsam, sowie beim Laden des Bildes auch AWS Amplify. Bei allen anderen untersuchten Operationen zeigt AWS Amplify hingegen eine äußerst hohe Performanz mit äußerst niedrigen Laufzeiten. Im Kapitel der Evaluation wird näher auf die Gründe dafür eingegangen.

5.4 Benutzbarkeit

Im Abschnitt der Benutzbarkeit werden die angebotenen Dokumentationen der Anbieter in Bezug auf die Dienste und die Bedienbarkeit des BaaS untersucht.

5.4.1 Dokumentation

In diesem Abschnitt wird auf die Dokumentationen der Anbieter eingegangen. Eine detaillierte Dokumentation ist die Grundlage dafür, die Dienste und Fähigkeiten eines Anbieters bei der Umsetzung eines Systems zu verstehen.

Die Dokumentation von Firebase ist ausführlich und vollständig unabhängig von der Google Cloud Dokumentation. Für jeden der oben erwähnten Service Bereich (Build, Realease and Monitor, Engage) wird eine eigene Dokumentation angeboten, die jeden enthaltenen Dienst beschreibt. Dabei wird auf die Fähigkeiten und Anwendungsfälle der Dienste eingegangen sowie grundlegende Konzepte erklärt. Außerdem sind kleinere Tutorials enthalten, die die Einbindung des Dienste in die eigene Anwendung zeigen. Es wird auf jedes unterstützte SDK eingegangen genauso wie auf die Sicherheit, die Verwendung, das Monitoring und die Performanz. Für jede unterstütze Plattform existiert außerdem ein Einstiegsguide, der zeigt, wie Firebase in die eigene Plattform integriert werden kann. Überdies hinaus bietet Firebase auch sogenannte Codelabs an: Diese Tutorials sind umfangreicher als die Guides der Dokumentation und enthalten oft das Erstellen einer ganzen Anwendung mit mehreren Komponenten [Firebase, 2022g]. Außerdem ist es möglich, sich für jedes SDK die entsprechende API Referenz anzeigen zu lassen. Auch Beispielprojekte sind in der Dokumentation vorhanden.

Amplify bietet unabhängig von der eigentlichen AWS Dokumentation eine eigene Dokumentation an. Diese wird in folgende Bereiche eingeteilt: Getting Started, Libraries, CLI, Studio, Hosting, Guides und API Reference. Der CLI und der Libraries Bereich sind die elementaren Teile der Dokumentation. Hier wird auf die Verwendung der verschiedenen angebotenen SDKs eingeangen genauso wie auf die korrekte Verwendung der Amplify CLI, da Amplify kein Web Interface zum Einrichten der Dienste bietet. Im Libraries Bereich werden alle Dienste beschrieben, die im Backend verwendet werden können. Code Beispiele werden direkt mit eingebunden. Die Libraries sind jedoch nach Plattformen getrennt: Es gibt sie nur für Javascript, Flutter, iOS, React Native und Android. Andere Plattformen werden nicht bedient. Die Dokumentation für das iOS SDK ist dabei merklich weniger umfangreich und es fehlt bspw. die Dokumentation für den Predictions Dienst im Vergleich zur Dokumentation von JavaScript. Somit muss, um einen Überblick über alle Dienste zu erlangen, zwischen den Plattformen gewechselt werden. Im CLI Bereich der Dokumentation wird genauer auf die nötige Konfiguration der Dienste mittels der CLI eingegangen. Die Dokumentation zu den Cloud Functions bspw. ist nur hier zu finden. Die Dokumentation beinhaltet zwar Codebeispiele, was jedoch fehlt ist eine grundlegende Erklärung der Funktionsweise der Dienste. Dies wird an vielen Stellen vernachlässigt. So müssen Entwickler auch auf die Dokumentation der entsprechenden Dienste von AWS zurückgreifen. Im Guides Bereich sind hingegen viele Beispiele zu finden, aber der Umfang variiert auch hier abhängig von der Plattform.

Back4App besitzt im Vergleich zu den anderen untersuchten Anbietern keine ausführliche Dokumentation. Die Dokumentation besteht größtenteils aus Guides, die nicht mehr als Beispiele zeigen für die Verwendung der Dienste. Die Einteilung richtet sich dabei nach der Plattform aber auch nach angebotenen Funktionen. Die Sortierung ist an dieser Stelle nicht konsistent. Weiterhin werden für die verschiedenen Plattform auch jeweils unterschiedliche Guides angeboten. Ein Einstiegsguide für die Verwendung des Storage ließ sich nur im Bereich von React-Native finden, im Bereich der anderen Plattformen fehlt ein solcher Guide. Die Funktionsweise der Dienste wird zu keinem Zeitpunkt erklärt. Es wird jedoch eine API Referenz angeboten, die je nach Daten und Klassen in der Datenbank automatisch angepasst wird, sodass jederzeit die Verwendung der API der Datenbank klar ist.

5.4.2 Bedienbarkeit

Bei der Bedienbarkeit wird darauf eingegangen, welche Möglichkeiten zur Bedienung die einzelnen Provider bieten und wie gut ein Backend damit umzusetzen ist.

Firebase bietet dem Entwickler mehrere Optionen zum Umsetzen und Konfigurieren einer Backendanwendung. Zum Einen gibt es ein Web Interface. Hier können die meisten der angebotenen Dienste eingerichtet werden, wie bspw. die Authentifizierung oder die Datenbank. Alle Dienste aus allen drei angebotenen Bereichen lassen sich zu einer erstellten Anwendung hinzufügen, einrichten und im Hinblick auf die Nutzung oder der Daten auch überwachen. Zum Anderen bietet Firebase eine CLI an, welche die vorhandene Funktionalität erweitert. Sie ermöglicht es, mit einer Reihe von Tools eine Anwendung zu verwalten und bereitzustellen. Die CLI lässt sich allerdings erst einrichten, sobald ein Firebase Projekt eingerichtet wurde. Die Verwendung des Web Interfaces ist also zwingend notwendig. Zum Bereitstellen der Cloud Functions muss hingegen zwingend die CLI verwendet werden, auch für den lokalen Emulator wird die CLI benötigt.

Amplify hingegen bietet für die Entwicklung eines Backends nur eine CLI an. Das existierende Web Interface beschränkt sich darauf, die bereitgestellten Dienste anzuzeigen, also die aktuelle Konfiguration des Backends. Gesondert aktiviert werden muss hingegen Amplify Studio. Damit lassen sich vollständige Fullstack Anwendung nahezu gänzlich ohne Code implementieren, Ampify Studio wurde im Rahmen dieser Arbeit jedoch nicht verwendet, deshalb wird sich im Folgenden auf die CLI fokussiert. Die CLI, die Amplify anbietet, wird zum Erstellen, Verändern und Bereitstellen aller Dienste verwendet und bietet noch eine Reihe weiterer Tools an. Mittels eines Befehls lassen sich Dienste zum Backend hinzufügen, die dann noch konfiguriert werden müssen. Dies geschieht innerhalb des Workflows in der CLI. Hier können bspw. die Zugriffsrichtlinien für die Daten festgelegt oder die Identitätsanbieter für den Social Login konfiguriert werden. Wenn die Konfiguration abgeschlossen ist erstellt die Amplify CLI einen Cloud Formation Stack mit dem dann die Dienste entsprechend der Konfiguration bereitgestellt werden. Auch nicht zu Amplify gehörende AWS Ressourcen lassen sich so zum Backend hinzufügen. Auch das lokale Mocking und Testing, auf das im weiteren Verlauf noch genauer eingegangen wird, wird mit der CLI umgesetzt.

Bei Back4App steht das Web Interface im Vordergrund. Nach dem Erstellen einer Anwendung in Back4App wird der Entwickler direkt auf das Dashboard der Anwendung weitergeleitet. Hier wird dem Entwickler ein Überblick über alle Dienste geboten, ohne

diese zuerst einrichten zu müssen. Die Datenbank ist direkt einsehbar genauso wie der Cloud Functions Dienst. Da Back4App auf dem Parse Server Framework basiert, sind die Dienste bereits verfügbar und können nach dem Importieren des SDK sofort verwendet werden. Mittels der CLI können allerdings auch neue Anwendungen erstellt, Dateien für Cloud Functions bereitgestellt oder das Hosting konfiguriert werden [Back4App, 2022h]. Mit der CLI lässt sich zum Debuggen und Testen auch ein lokaler Parse Server starten. Die Funktionalität ist jedoch begrenzt und beschränkt sich auf allgemeine Einstellungen wie die Parse Server Version und das Erstellen einer Anwendung oder das Bereitstellen Cloud Function. Für alle anderen Dienste bietet die CLI keine Schnittstelle an.

5.4.3 Zusammenfassung

Tabelle 5.4: Zusammenfassung der Benutzbarkeit

	Firebase	**Amplify**	**Back4App**
Dokumentation	Ausführliche Dokumentation für jede Plattform, viele Tutorials und Guides, umfangreiche Codelabs, API Referenz	Umfangreiche Dokumentation mit eingebetteten Tutorials, je nach Plattform unterschiedlich ausführliche Dokumentation, Teilweise muss auf AWS Dokumentation zurückgegriffen werden, API Referenz wird automatisch erstellt	Keine ausführliche Dokumentation die nur aus Beispielen besteht, je nach Plattform werden unterschiedliche Technologien behandelt, teilweise unübersichtlich, API Referenz wird automatisch erstellt
Punkte	. . .	. .	.
Bedienbarkeit	Übersichtliches und einfach zu bedienendes Web Interface, einfach zu verwendende CLI	Einrichtung der Dienste ausschließlich mit der CLI, umfangreiche Konfiguration mit der CLI möglich, Web Interface für Amplify Studio	Web Interface zum Überwachen der Dienste und Schreiben der Cloud Functions, CLI nur wenig Funktionalität, übersichtliches Dashboard
Punkte	. . .	. . .	. .

5.5 Verlässlichkeit

Im Abschnitt zu der Verlässlichkeit wird im Rahmen der nicht-funktionalen Anforderungen an das Backend untersucht, wie die Cloud Anbieter Verfügbarkeit, Skalierbarkeit und Disaster Recovery umsetzen.

5.5.1 Verfügbarkeit

Die Verfügbarkeit eines Systems ist insofern gewährleistet, als dass das System zu möglichst jeder Zeit reagiert, funktioniert und erreichbar ist. Dabei wird die Verfügbarkeit

oft als Metrik mittels einer Prozentzahl angegeben, um die Verlässlichkeit eines Systems messbar zu machen. Es soll untersucht werden, wie die Verfügbarkeit der einzelnen Anbieter in Bezug auf die Service Level Agreements angegeben wird.

Firebase setzt folgende Verfügbarkeiten für die einzelnen Dienste fest: Für die Realtime Database 99,95 %, für den regionalen Storage 99,9 % , für den Authentifizierungsdienst wurde kein SLA angegeben, für die Cloud Functions 99,95 % und für den regionalen Firestore 99,99 %.
Amplify als Framework und AWS Dienst an sich bietet keinerlei SLAs bezüglich der Verfügbarkeit, es muss auf die eigentlichen AWS Dienste geschaut werden. Cognito bietet 99,9 %, die DynamoDB zwischen 99,99 und 99,999 %, der simple Storage Service (S3) 99,9 %, Lambda bietet 99,95 % und AppSync 99,95 %. Berechnet werden diese Zahlen dabei auf Basis der Zeit, in der die Services verfügbar waren dividiert durch durch die Gesamtzeit, immer in Bezug auf einen Monat. Sollten die SLAs nicht eingehalten werden können, bietet AWS einen Kostenausgleich für die Verwendung der Dienste.
Back4App selbst bietet keinerlei SLAs, da der Parse Server aber in AWS gehostet wird gelten die SLA des Elastic Compute Cloud Services von AWS: 99,99 %.

Insgesamt lässt sich also feststellen, dass alle Anbieter die Anforderungen in Bezug auf hohe Verfügbarkeit erfüllen. Sollten diese Verfügbarkeiten bei AWS nicht erreicht werden, so bietet AWS dem Kunden sogar eine Entschädigung in Form von verringerten Kosten. Für

5.5.2 Skalierbarkeit

Um die Skalierbarkeit der Cloud Anbieter beurteilen zu können wird untersucht, inwieweit der Anbieter selbstständig Ressourcen anpasst, um die Verfügbarkeit und ausreichende Skalierung des Systems zu gewährleisten. Alle untersuchten Anbieter ermöglichen bezüglich der meisten Dienste automatisches Skalieren an, insbesondere in Bezug auf die Verwendung der Cloud Functions. Bei diesen Diensten werden je nach Anzahl der ankommenden Requests und der Laufzeit bei Bedarf mehr Instanzen einer Funktion gestartet.

Die Realtime Database von Firebase besitzt ein Limit von 200.000 gleichzeitigen Nutzern. Wenn dieses überschritten wird, soll insofern skaliert werden, als dass vom Entwickler selbst neue Datenbankinstanzen gestartet werden können [Firebase, 2022o]. Der Firestore hingegen besitzt keine harten Limits, einzig die Performance leidet bei überschreiten

der angegebenen Grenzen. Der Firestore skaliert dabei automatisch [Firebase, 2022q]. Die Cloud Funcions bei Firebase können höchstens 3.000 gleichzeitige Instanzen einer Funktion verarbeiten und dieses Limit kann auch nicht erhöht werden [Firebase, 2022m]. Die bei AWS Amplify verwendeten Dienste besitzen keinerlei Limitierungen bei der Skalierung und erledigen dies automatisch. Im AWS Simple Storage Service (S3) lässt sich bspw. eine unbegrenzte Menge an Objekten speichern und auch der eingenommene Speicherplatz ist unbegrenzt. Auch die DynamoDB verwendet Auto Scaling: Es lassen sich sogar angepasste Grenzwerte für die Anzahl der verwendeten Lese- oder Schreibkapazität festlegen. Wenn diese über- oder unterschritten werden so skaliert AWS die Datenbank automatisch. Dabei kommen entweder neue Instanzen der hinzu oder fallen weg [Yoder und Shriver, 2022]. Für die Lambda Funktionen existiert ein Soft limit: Es können 1.000 Instanzen der Funtkion gleichzeitig laufen, aber dieses Limit lässt sich auf knapp 100.000 gleichzeitig laufende Instanzen erhöhen [AWS, 2022j].
Back4App bietet leider keine Informationen in Bezug auf die Skalierbarkeit an, die genaue Infrastruktur der Plattform ist leider nicht bekannt, außer, dass ein Parse Server in der Elastic Compute Cloud von AWS bereitgestellt wird. Die Skalierung kann für Back4App also nicht abschließend eingeschätzt werden, aber da jeder angebotene Kostenplan eigene Limits für bspw. die Anzahl der Anfragen pro Monat enthält kann davon ausgegangen werden, dass in einem teureren Kostenplan vertikal skaliert wird und die Größe der Instanz angepasst wird. Im Dedicated Plan hingegen scheint auch horizontale Skalierung enthalten zu sein, da dort jegliches Limit entfällt.

5.5.3 Disaster Recovery und Backup

Für Systeme, die mit Daten arbeiten, sind regelmäßige Backups essenziell. Deshalb wird analysiert, ob die untersuchten Anbieter automatische Backups anbieten und was im Falle eines Systemausfalls geschieht.

Firebase bietet für einen Ausfall der Dienste verschiedene Lösungen an. Für die Realtime Database werden regelmäßige und automatische Backups angeboten, ein Mal alle 24 Stunden wird ein Backup der Daten erzeugt, das für 30 Tage verfügbar ist. Dies ist jedoch nur im Blaze Kostenplan von Firebase möglich, im kostenlosen Plan fehlt diese Option. Die Backups werden in einem Storage Bucket der Google Cloud gespeichert. Auch manuelle Backups lassen sich so erzeugen. Aus Backups lässt sich dann eine Datenbank vollständig wiederherstellen [Firebase, 2022b]. Für den Firestore sind keine automatischen Backups nötig: Ob regional oder multiregional bereitgestellt, der Firestore wird

innerhalb verschiedener Zonen repliziert und kann somit Systemausfälle ganzer Zonen oder Regionen verarbeiten.
AWS bietet für die DynamoDB auch die Möglichkeit automatischer Backups an. Mit der Point-in-time-Reocvery werden kontinuierlich Backups erstellt und für einen Zeitraum von 35 Tagen gespeichert. Auch manuelle Backups sind möglich. Mittels eines Backups lässt sich dann eine DynamoDB wieder vollständig herstellen [AWS, 2022o].
Back4App hat eine automatische Backup Policy für die Datenbank: Jede Stunde wird ein Backup gemacht, das sieben Tage behalten wird und jeden Tag wird ein Backup gemacht, das für 30 Tage gespeichert wird. Manuell können die Daten der Datenbank auch heruntergeladen und gespeichert werden [Back4App, 2022p].

5.5.4 Zusammenfassung

Tabelle 5.5: Zusammenfassung der Verlässlichkeit

	Firebase	**Amplify**	**Back4App**
Verfügbarkeit	Hohe Verfügbarkeit der Dienste (mindestens 99,9 %)	Für Amplify keine Verfügbarkeit angegeben, für die verwendeten Dienste hohe Verfügbarkeit (mindestens 99,9 %)	Keine SLAs angegeben, durch Hosten des Parse Servers in AWS gelten die SLAs des AWS EC2 Dienstes (99,9 %)
Punkte	. . .	. . .	. . .
Skalierbarkeit	Automatische Skalierung der Cloud Functions, des Firestores und des Storage, Realtime Database skaliert nur bis 200.000 Nutzer	Alle Dienste skalieren automatisch, dabei keine Limitierungen, Anpassung der Skalierung möglich	Keine Informationen zur Skalierung, durch Hosten des Parse Servers in AWS EC2 wahrscheinlich keine automatische Skalierung
Punkte	. .	. . .	.
Disaster Recovery Backup	Tägliche Backups die 30 Tage gespeichert werden, vollständige Wiederherstellung aus Backup möglich, Firestore ist durch Replikation vor Ausfällen geschützt	Optionale automatische Backups, vollständige Wiederherstellung aus Backups möglich, Backups werden 35 Tage gespeichert, Optionale Replikation	Automatische Backups, werden für 30 Tage gespeichert, Datenbank lässt sich vollständig wiederherstellen, keine Replikation möglich
Punkte	. . .	. . .	. .

5.6 Wartbarkeit

Im Abschnitt der Wartbarkeit wird vor allem auf die Analysierbarkeit der Anwendungen eingegangen. Dabei soll untersucht werden, wie umfangreich das Monitoring der Anbieter ausfällt und inwieweit sich eine zu implementierende Anwendung debuggen lässt.

5.6.1 Monitoring

Die Möglichkeiten für das Überwachen der implementierten Anwendungen gestalten sich bei allen Anbietern unterschiedlich, im Rahmen dieser Arbeit ist es daher wichtig, das

Monitoring der einzelnen Anbieter zu betrachten, um die Analysierbarkeit einschätzen zu können.

Google bietet für Firebase ein sehr feingranulares Monitoring an, das sich in verschiedene Bereiche gliedert. Zum Einen ist es möglich, im Release and Monitor Bereich des Web Interfaces detaillierte Einblicke in die Performance der mobilen App zu bekommen. Mittels eines in die Anwendung zu integrierenden SDKs ist es außerdem möglich, Daten zur Performanz der App zu sammeln und konsolidiert darzustellen. Dabei werden bspw. die Startzeit der App und verschiedene Netzwerkanfragen getrackt. Weiterhin ist es möglich, selbst erstellte Metriken für interessante Code Passagen zu generieren. Zum Anderen gibt es auch für die verwendeten Dienste ein ausführliches Monitoring. Jeder Dienst besitzt einen eigenen Monitoring Bereich, in dem grundlegende Metriken wie die Menge der gespeicherten Daten oder Anzahl der Anfragen festgehalten werden. Dort sind auch die Obergrenzen für die kostenlose Verwendung des Dienstes einsehbar. Überdies hinaus ist es durch die Integration von Firebase in die Google Cloud möglich, das Monitoring der Google Cloud zu verwenden. Dies ist weitaus umfangreicher, als das was der Entwickler in der Firebase Konsole einsehen kann. Für jeden Dienst lassen sich diverse Metriken darstellen und sogar eigene Dashboards erstellen. Allein für die Realtime Database sind 18 verschiedene Metriken verfügbar. Die Cloud Functions lassen sich hingegen nur im Monitoring Bereich der Google Cloud analysieren. Firebase bietet hier keine eigene Monitoring Lösung an, abgesehen von der Anzahl der Aufrufe. Die Nutzung des Authentifizierungsdienstes hingegen kann nur im Firebase Web Interface eingesehen werden. Hier besteht keine Verbindung zur Google Cloud.

Amplify bietet als AWS Dienst kein dediziertes Monitoring an. Dafür werden Metriken aller für Amplify verwendeten Dienste, bspw. für den Simple Storage Service, an AWS Cloudwatch übertragen und dort als Diagramm dargestellt. CloudWatch ist der Monitoring Dienst von AWS, in dem zentralisiert alle bei AWS verfügbaren Dienste überwacht werden können. In CloudWatch sind zusätzlich zu den Metriken auch Logs der Dienste einsehbar, was bspw. beim Analysieren der Ausführungen der Lambda Funktionen hilft. Die Logs sind dabei nach Gruppen sortiert und richten sich nach dem Namen der Lambda Funktion. Für die Logs lassen sich auch in einem Editor Queries erstellen, sodass hier sehr detaillierte Einblicke möglich sind. Typische Metriken für einen Dienst wie die DynamoDB sind z.B. HTTP 4XX oder 5XX Fehler, die Latenz oder auch die übertragene Datenmenge. Diese Metriken lassen sich wiederum in verschiedenen Dimensionen ansehen. Es ist möglich, sie bspw. nur auf eine bestimmte Amplify Anwendung zu beschränken oder auch für den ganzen AWS Account darzustellen. Auch Alarme lassen sich

hier einstellen um benachrichtigt zu werden, wenn ein Dienst nicht so funktioniert wie erwartet. Dashboards lassen sich hier ebenso erstellen, um gewünschte Metriken zu kombinieren und gesammelt darzustellen. Dies kann auch automatisiert erfolgen. Im Explorer der Metriken sind für eine bessere Übersichtlichkeit nur die Metriken dargestellt, die zu Diensten gehören, die aktuell verwendet werden. Die Metriken lassen sich hier beliebig kombinieren und aggregieren und bieten so einen ausführlichen Einblick in die Dienste.

Auch Back4App als BaaS Anbieter eröffnet dem Entwickler einige Möglichkeiten, eine erstellte Anwendung zu überwachen. Im Web Interface von Back4App gibt es einen Analytics Abschnitt. Hier gibt es verschiedene Möglichkeiten zum Darstellen von Metriken: Im Explorer lassen sich in einem Diagramm Metriken bezüglich der Nutzer auswählen, so lassen sich eigene Diagramme erstellen und beliebig kombinieren. Es können bspw. gleichzeitig die Anzahl der täglichen Nutzer und der Neuinstallationen dargestellt werden oder die Anzahl der API Anfragen [Back4App, 2022c]. Diese Anfragen lassen sich auch speichern, um später einfacher auf diese Metriken zugreifen zu können. Der Umfang der auszuwählenden Metriken ist jedoch stark begrenzt. Außer der Anzahl der API Anfragen sind hier keine anderen auf die Dienste bezogenen Metriken zu finden. Im Performance Bereich von Analytics lassen sich diese API Anfragen noch genauer analysieren: Hier werden die Gesamtzahl der Anfragen sowie das Limit und die Anzahl der erfüllten und verworfenen Anfragen dargestellt. Im Abschnitt der Slow Requests werden diejenigen API Anfragen dargestellt, die eine besonders schlechte Performanz hatten (Verglichen mit einem automatisch erstellten Richtwert). Dies konnte jedoch nicht getestet werden, da keine solche Anfrage in der implementierten Anwendung vorhanden war. Abgesehen von diesen drei Möglichkeiten bietet Back4App keine weiteren Funktionen zum Überwachen der Dienste. Die Cloud Functions bspw. werden vollständig außer Acht gelassen, hier existieren einzig die Logs des Parse Servers. Diese loggen jedoch jede Aktivität des Servers und lassen sich nicht filtern. Auch Metriken zum belegten Speicher der Datenbank oder des Storage fehlen, das Monitoring scheint sich ausschließlich auf die API der Datenbank und die Nutzer zu beschränken.

5.6.2 Debugging

Das Debugging ist ein weiter wichtiger Punkt, um die Analysierbarkeit der Dienste eines BaaS Anbieters einordnen zu können. Im Folgenden wird auf die wichtigsten Debugging Möglichkeiten der einzelnen Anbieter eingegangen.

Firebase bietet zum Debuggen der meisten angebotenen Dienste eine lokale Emulator Suite an. Die Emulator Suite besteht dabei aus einzelnen Emulatoren für die jeweiligen Dienste, die je nach Bedarf verwendet werden können. Dies eignet sich zum Testen der Funktionalität oder auch der erstellten Sicherheitsregeln, Komponenten können unter realen Bedingungen getestet werden. Zum Verwenden der Emulatoren wird die CLI benötigt, beim Ausführen wird auch lokal ein Web Interface bereitgestellt.
Amplify bietet ähnlich wie Firebase eine lokale Mocking und Testing Funktion für die meisten erstellten Dienste des Backends an. Dies funktioniert wie auch das Bereitstellen oder Konfigurieren der Dienste mittels der Amplify CLI. Es kann bspw. die API des Backends gemockt werden: Auf diese Weise wird lokal ein GraphQL Endpunkt bereitgestellt der sich genauso verwenden lässt wie eine API einer Produktionsumgebung. Der einzige Dienst, der sich so nicht testen lässt, ist die Authentifizierung.
Back4App bietet keine direkte Möglichkeit, Dienste der Anwendung zu testen und zu debuggen, bevor sie bereitgestellt werden. Da Back4App jedoch auf dem Parse Framework basiert und der zugrunde liegende Server, auf dem die Backendanwendung läuft, ein Parse Server ist, gibt es für den Entwickler die Möglichkeit, lokal auf dem eigenen Rechner das Parse Framework zu installieren und die Dienste so zu testen [Back4App, 2022g]. Der lokale Parse Server lässt sich mit einer Anwendung in Back4App verbinden, sodass bspw. bereits erstelle Cloud Functions lokal verwendbar sind [Back4App, 2022d].

5.6.3 Zusammenfassung

Tabelle 5.6: Zusammenfassung der Wartbarkeit

	Firebase	**Amplify**	**Back4App**
Monitoring	Umfangreiches Monitoring der Dienste und auch des Nutzerverhaltens, Integration mit dem Monitoring der Google Cloud, in der Google Cloud lassen sich Dashboards erstellen	Kein Monitoring bei Amplify an sich möglich, eingerichtete Dienste lassen sich nur mit AWS CloudWatch überwachen, damit jedoch äußerst umfangreiches Monitoring und auch Logging möglich, automatisch erstellte Dashboards	Überwachen des Nutzerverhaltens möglich, Monitoring der Dienste nicht sehr umfangreich, es wird nur die Anzahl an API Anfragen dargestellt, einzelne Dienste hingegen lassen sich überhaupt nicht überwachen
Punkte	. . .	. . .	.
Debugging	Ausführliches Debugging mittels eines lokalen Emulators möglich, ausgiebiges Testen der Dienste möglich	Lokales Mocking und Testing für die API und den DataStore sowie den Storage und die Lambda Funktionen möglich	Einzelne Dienste lassen sich nicht testen, lokal muss ein Parse Server eingerichtet werden, dieser lässt sich jedoch mit der Online Umgebung verbinden zum Testen der eingerichteten Dienste, Testen der API im Web Interface möglich
Punkte	. . .	. .	. .

5.7 Sicherheit

In diesem Kapitel werden die Anbieter in Bezug auf die Sicherheit der einzelnen Dienste untersucht. Dabei steht vor allem die Sicherheit der Daten und der Zugriff darauf im Vordergrund.

Firebase bietet zum Regulieren der Sicherheit der Dienste eine deklarative Regelsprache. So lassen sich für die Realtime Database, den Storage und den Firestore Regeln für den Zugriff auf die Daten festlegen. Für die Realtime Database wird JSON zur Festlegung der Regeln verwendet. Für den Firestore und den Storage wurde die Sprache auf Grund der möglichen Komplexität der Daten weiter verfeinert und basiert nicht auf JSON [Firebase, 2022i]. Die Regeln lassen sich mittels der Firebase CLI oder in der Konsole von Firebase erstellen und auch testen. Mit diesen Regeln lassen sich einzelne Pfade der Datenbank, des Storage oder des Firestore sichern. Jede Anfrage an einen der erwähnten Dienste wird mittels der Sicherheitsregeln überprüft. Ergänzend kann der Authentifizierungsdienst verwendet werden, um Zugang zu Daten nur autorisierten Nutzern zu ermöglichen. Mit den Regeln können auch die erlaubten Operationen für einen Pfad einer Datenbank oder des

Storage festgelegt werden [Firebase, 2022p]. Wenn keine Regeln festgelegt wurde, so wird automatisch jede Anfrage abgelehnt.

Auch Amplify bietet Mechanismen zur Sicherung der Daten an. Grundlage dafür ist AWS IAM, der Identity and Access Management Dienst von AWS. Dieser Dienst hilft dabei, Zugänge zu Ressourcen in der Cloud zu administrieren. Dabei werden Authentifizierung und Autorisierung ermöglicht. Zum Festlegen des Zugriffs auf Ressourcen werden im IAM Dienst sogenannte Policys verwendet. Dies sind JSON Dokumente, die regeln, welche Aktionen für welche Ressourcen von welchen Prinzipalen durchgeführt werden dürfen. Die Policys können dabei Ressourcen- oder Identitäsbasiert sein [AWS, 2022i]. Bei der Konfiguration der Dienste in der CLI wird erfragt, welche Ressourcen und Identitäten Zugriff auf den Dienst bekommen sollen. Mit dem CloudFormation Dienst werden diese Policys dann erzeugt und an die Ressourcen angehängt. Weiterhin gibt es die Möglichkeit, einzelne Modelle des GraphQL Schemas mit eigenen Sicherheitsregeln zu versehen. Dies stellt nicht nur sicher, dass niemand unberechtigtes auf die Daten des Dienstes zugreift, sondern auch, dass Nutzer auch andere Daten außer ihren eigenen sehen können. Auch der Authentifizierungsdienst Cognito kann so in die Sicherheit der Datenbank eingebunden werden [AWS, 2022e]. Wie bei Firebase lassen sich die jeweils erlaubten Operationen genau festlegen und auch der Zugriff auf Feld Ebene kann so geregelt werden. Auch für die anderen Komponenten des Backends werden beim Erstellen über die CLI Zugriffsrichtlinien erstellt. Beim Konfigurieren einer Lambda Funktion lässt sich bspw. festlegen, ob diese Funktion Zugriff auf die GraphQL API benötigt.

Back4App bietet als grundlegendes Sicherheitsfeature Access Control Lists für die Datenbank an. Dies sind Regeln, die beim Erstellen einer Klasse in der Datenbank gesetzt werden. Mit diesen Regeln lässt sich festlegen, wer auf welche Daten zugreifen kann, aber die Einstellungsmöglichkeiten sind eingeschränkt. Der Zugriff lässt sich nur auf öffentlich, private, eine bestimmte erstellte Rolle oder den Eigentümer bezüglich der Daten beschränken. Es lassen sich jedoch Lese- und Schreiboperationen unterscheiden. Die Prüfung von Bedingungen, wie bspw. um zu überprüfen, ob ein Nutzer eingeloggt ist, fehlt vollständig. Diese ACLs lassen sich auf Objekt Level und Class Level einrichten. Für weitere Sicherheitsmaßnahmen wird das Erstellen von Cloud Functions empfohlen, die entsprechende Zugriffe regeln [Back4App, 2022f]. Back4App bietet außerdem an, eine mit Back4App erstellte Anwendung auf General Data Protection Regulation (GDPR) Konformität zu überprüfen. Dies hilft dem Entwickler dabei, eine Anwendung zu veröffentlichen, die alle Datenschutzanforderungen, die in der EU gelten, erfüllt.

Tabelle 5.7: Zusammenfassung der Sicherheitsaspekte der Anbieter

	Firebase	**Amplify**	**Back4App**
Sicherheit	Daten lassen sich mittels feingranularer Regeln sichern, Autorisierung mittels der Kombination aus Authentifizierungsdienst und erstellten Regeln	Automatisch erstellte Zugriffsrichtlinien mit AWS IAM, auch für die Kommunikation der Dienste untereinander, Regeln zur Sicherung der Daten in der Persistenz	Access Control Lists zur Sicherung des Datenzugriffs, nur eingeschränkte Möglichkeiten bei der Granularität, Angebot zur Überprüfung der GDPR Konformität
Punkte	. .	. . .	.

6 Evaluation

In diesem Kapitel wird der Vergleich der untersuchten Anbieter aus dem vorherigen Kapitel evaluiert. Die Ergebnisse werden in Bezug auf die Fragestellung zusammengefasst und interpretiert und es wird eine Übersicht über alle Anbieter mittels des eingeführten Punktesystems gegeben. Weiterhin wird auf die praktische Umsetzung eingegangen und es werden Limitationen der Forschung aufgezeigt.

Das Serverless Paradigma allgemein und BaaS im Besonderen ist immer noch eine vergleichsweise neue Entwicklung im Bereich des Cloud Computings und die Verwendung bringt viele Vor-, aber auch Nachteile mit sich. Sinkende operationale Kosten, Betriebskosten und Entwicklungskosten, ein geringerer Verwaltungsaufwand und die verkürzte Time-to-market sind nur einige der Vorteile, die die Verwendung eines BaaS bietet. Aber genau dies sind die Vorteile, die das Team von Evender davon überzeugten, ein BaaS zu verwenden anstatt das Backend für die mobile App eigenhändig zu implementieren. Damit einher geht jedoch auch das Risiko für die Entstehung unvorhergesehener Kosten oder eines Vendor Lock-ins. Als grundlegende Motivation für diese Arbeit diente die Frage nach der Eignung eines BaaS in Bezug auf verschiedene Anbieter dieses Serverless Konzeptes. Dabei wurden drei BaaS Anbieter untersucht: Google Firebase, Amplify von AWS und Back4App. Erwartet wurde eine bei allen Anbietern laufende Backendanwendung, mittels der dann der Vergleich der Anbieter durchgeführt werden konnte. Anhand der Ergebnisse dieses Vergleiches soll nun ermittelt werden, inwieweit die verschiedenen Backendanwendungen geeignet sind, um als Backend der mobilen App von Evender zu dienen.

6.1 Vergleich

Die implementierten Backendanwendungen bei Google Firebase, AWS Amplify und Back4App zeigen viele Gemeinsamkeiten, aber auch Unterschiede. Im Vergleich der Anbieter wurde

zuerst auf allgemeine Kriterien wie die Umsetzung der Anforderungen, die Architektur, die Kosten und die Regionen, in denen der Anbieter seine Dienste offeriert, eingegangen. Die Anforderungen aus der Analyse und Spezifikation beinhalten unter anderem das Laden und Speichern von Veranstaltungen, den Versand eines Newsletters und eine Authentifizierung, aber auch nicht-funktionale Anforderungen wie eine hohe Erreichbarkeit und Skalierbarkeit. Die funktionalen Anforderungen konnten bis auf das Empfehlungssystem bei allen Anbietern umgesetzt werden und auch die nicht-funktionalen Anforderungen wurden erfüllt, auf diese wird im Folgenden aber noch näher eingegangen, genauso wie auf das Empfehlungssystem.
Weiterhin zeigten sich in Bezug auf die in Kapitel vier erarbeitete Zielarchitektur nur geringfügige Unterschiede bei den resultierenden Architekturen der Backendanwendungen: Das eben erwähnte Empfehlungssystem fehlt als Komponente. Außerdem ist bei der Anwendung, die mittels Back4App implementiert wurde, der Storage für Bilder und andere Dateien zwangsläufig mit der Datenbank verbunden. Bei der Untersuchung der Kosten zeigte sich, dass die Verwendung von AWS Amplify die geringsten Kosten auf Basis eines geschätzten Nutzungsprofils verursacht. Bei den Regionen, in denen die Anbieter aktiv sind, fällt nur Back4App auf: Hier lassen sich im Gegensatz zu Firebase und Amplify keine spezifischen Regionen oder sogar Zonen auswählen. In den kostspieligeren Plänen lassen sich Regionen nur grob auf Kontinentalebene aussuchen.
Dass das Empfehlungssystem nicht umgesetzt werden konnte, lag an der fehlenden Datengrundlage und nicht an der grundsätzlichen Fähigkeit der Anbieter. Einzig Back4App bietet keine Möglichkeit zur Umsetzung eines Empfehlungssystems. Bei Verwendung dieser Backendanwendung müsste solch ein System außerhalb des eigentlichen Backends mittels eines Drittanbieters umgesetzt werden. Die resultierenden Architekturen zeigen, dass die Implementierungen der konzipierten Zielarchitektur sehr ähneln. Dies spricht für eine Eignung des BaaS Konzeptes aller Anbieter für den dargestellten Anwendungsfall. In Bezug auf die erhobenen Anforderungen sind Firebase und AWS Amplify hier aber zu bevorzugen, da diese Anbieter integrierte Machine Learning Dienste bieten. AWS bietet sogar einen Dienst zur Generierung von Empfehlungen an und ist in Bezug auf die Anforderungen am ehesten geeignet.
Dass Back4App die Storage Komponente gesondert behandelt und in die Datenbank integriert schränkt in keinster Weise die Funktionalität ein. Bei den Kosten überraschte hingegen, dass AWS Amplify am günstigsten ist. Hier zeigt sich, dass etablierte Anbieter auf Grund des Skaleneffektes ihre Dienste günstiger anbieten können. Bei Firebase und AWS besteht jedoch die Gefahr des Auftretens unerwarteter Kosten auf Grund des Pay as you go Prinzips. Da bei Back4App eine monatliche nicht variable Gebühr gezahlt

wird, besteht dieses Risiko hier nicht im gleichen Ausmaß. Wenn jedoch die bestehenden Grenzen für bspw. die Anzahl der Anfragen an das Backend überschritten werden, so müssen auch hier gesonderte Kosten getragen werden. AWS Amplify und Google Firebase bieten bezüglich der auswählbaren Regionen für das Backend eine große Auswahl, die Anwendung kann nahezu überall auf der Welt betrieben werden. Bei beiden Anbietern kann auch eine Replikation der Daten vollzogen werden, um eine höhere Ausfallsicherheit gewährleisten zu können. Back4App bietet hier keine feingranulare Einstellung der Region der Anwendung, sodass Firebase und Amplify in Bezug auf die Kosten und die verfügbaren Regionen positiver zu bewerten sind als Back4App, wobei AWS bei den Kosten am Günstigsten ist, was sich besonders für die frühen Phasen der Entwicklung eignet. Langfristig gesehen, bspw. in Bezug auf eine eventuelle Expansion von Evender, sind die verfügbaren Regionen von Back4App nicht ausreichend.

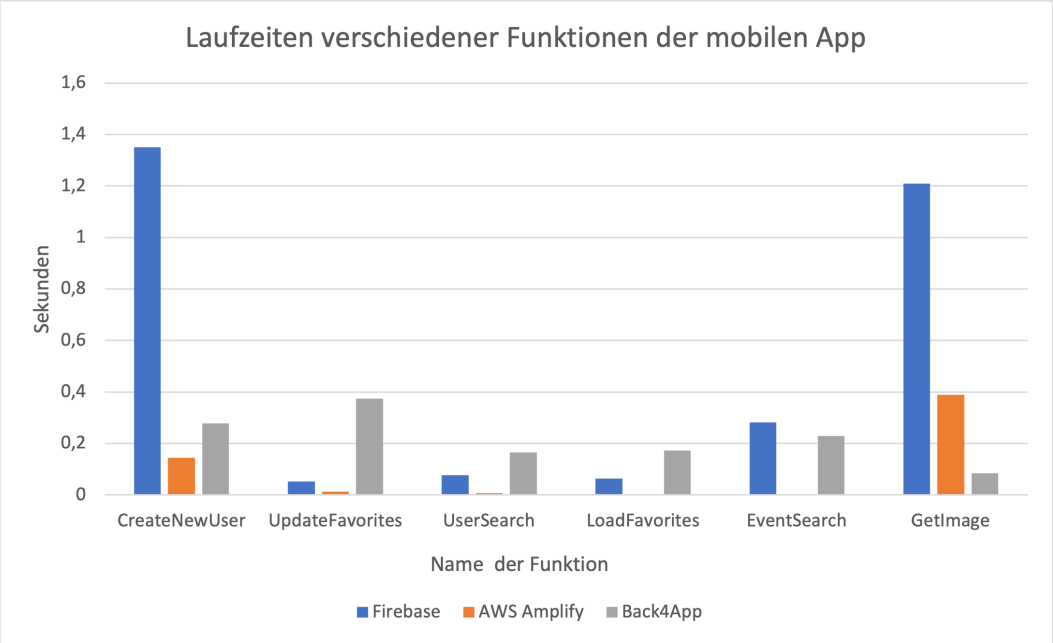
Bei den zur Umsetzung verwendeten Diensten der Anbieter sind die Unterschiede größer. Bei dem Angebot und der Vielfalt der angebotenen Dienste besitzen Google Firebase und AWS Amplify den großen Vorteil der Integration in das Ökosystem der Cloud Anbieter. Die Google Cloud Plattform und AWS gehören zu den führenden Cloud Anbietern und die Auswahl an Diensten ist dementsprechend groß [Zhang, 2022]. Firebase bietet, wenn bei AWS die Dienste außerhalb von Amplify außen vor gelassen werden, die meisten Dienste bezogen auf die Entwicklung einer Backendanwendung an. Auch die Möglichkeiten zur Steigerung des Engagements der Nutzer sind vielfältig, bspw. in Form von A/B Testing oder Google Analytics. Amplify ist hier im Vergleich eingeschränkter, was vergleichbare Funktionalitäten betrifft, während Back4App nur grundlegende Funktionen wie eine Datenbank und Cloud Functions bietet. Dies könnte darin begründet sein, dass kein großes Cloud Unternehmen hinter dem BaaS von Back4App steht. Dafür besitzt Back4App den Vorteil, dass das darunterliegende Framework auf Open Source basiert. Dies vermindert die Abhängigkeit von einem Anbieter, theoretisch ließe sich eine bei Back4App entwickelte Backendanwendung problemlos zu einem selbst gehosteten Parse Server migrieren. So kann auch die Funktionalität selbst verwaltet und erweitert werden. Die Integration des Backends in die mobile App verhält sich hingegen bei allen untersuchten Anbietern ähnlich. Grundlage sind vom Anbieter bereitgestellte Software Development Kits und Konfigurationsdateien, die eine Verbindung zu den Diensten des Backends ermöglichen. Positiv aufgefallen ist hier Back4App: Es müssen keine einzelnen Dienste aktiviert oder zum Backend hinzugefügt werden und die Dienste lassen sich mit dem Parse SDK direkt verwenden. Die Persistenz Komponente, die bei allen Anbietern mittels einer Datenbank umgesetzt wurde, weißt größere Unterschiede auf. Firebase bietet hier zwei

verschiedene Versionen einer Datenbank an, jeweils für andere Anwendungsfälle, jedoch basieren beide auf NoSQL. Die Realtime Database ermöglicht Echtzeitdatenverarbeitung und bietet ein Datenmodell basierend auf einem JSON Baum während der Firestore als Sammlung von Dokumenten speichert und komplexere Abfragen ermöglicht. Umgesetzt wurde die Persistenz in der Backendanwendung mit der Realtime Database auf Grund des vorteilhafteren Offline Synchronisationsverhaltens und der einfachen Struktur der anfallenden Daten. Die Datenbank ist mittels eines SDKs oder per REST API erreichbar. AWS Amplify bietet für die Speicherung von Daten nur den DataStore an. Dieser zeichnet sich jedoch durch ein ausgeklügeltes System der Datensynchronisierung aus. Es wird AWS AppSync verwendet, um Clients mit der eigentlich verwendeten DynamoDB zu verbinden und AppSync lässt sich auch ohne den DataStore als Schnittstelle zur Datenbank verwenden. Dies ermöglicht dem Entwickler viel Flexibilität. Das Datenmodell basiert auf GraphQL, was zusätzlich einige Vorteile wie einen geringere übertragene Datenmenge mit sich bringt. Auch eine API Referenz wird automatisch erstellt. Back4App bietet dem Entwickler die Möglichkeit, eine mongoDB oder eine PostgreSQL Datenbank zu verwenden. Grundlage für das Datenmodell ist die *Parse.Object* Klasse des Parse Frameworks. Weiterhin lässt sich die Datenbank nicht nur per SDK sondern auch über eine REST und eine GraphQL Schnittstelle erreichen, auch hier wird automatisch eine API Referenz für die Daten in der Datenbank erstellt. Auch eine Echtzeitdatenverarbeitung ist möglich, wenn die Live Queries aktiviert werden. Die einzelnen Anbieter weisen somit große Unterschiede auf: Die Datenmodelle unterscheiden sich genauso wie die angebotenen Schnittstellen. Alle angebotenen Dienste eignen sich für die Umsetzung des Backends, AWS bietet dem Entwickler jedoch ein äußerst ausgereiftes Datenmodell. Durch die obligatorische Modellierung der Daten werden Fehler in der Modellierung vermieden, auch die Schnittstellen insbesondere in Bezug GraphQL und das Offline Verhalten mit AWS AppSync wirken durchdacht. Firebase bietet zwar eine größere Auswahl an möglichen Datenbanken an, genauso wie Back4App, aber AWS Amplify überzeugt bei diesem Kriterium in einem noch größeren Ausmaß.

Die Dienste der Anbieter zur Umsetzung der Authentifizierung zeigen weniger große Unterschiede. Mittels der entsprechenden Libraries der SDKs war es bei jedem Anbieter möglich, einen Login auch mit Drittanbieteridentitäten umzusetzen. Weiterhin stellt jeder Anbieter eine FaaS Plattform bereit. AWS bietet hier die größte Auswahl an Laufzeitumgebungen an während Firebase und Back4App nur Java-/Typsecript bzw. nur Javascript anbieten. Die grundlegende Funktionsweise ist jedoch gleich: Durch einen selbstgewählten Trigger wie zum Beispiel eine HTTP Anfrage oder ein Upload eines Objektes in der

Datenbank wird die entsprechende Funktion ausgelöst. Firebase und AWS bieten hier zum Ausgleich eines Cold Starts die Möglichkeit des Warmhaltens einer Funktionsinstanz. Back4App hingegen bietet dies nicht und auch das automatische Skalieren fehlt. Somit sind Firebase und AWS hier besser zur Umsetzung von Geschäftslogik mittels Cloud Functions geeignet, wobei AWS in Bezug auf die Konfiguration der Funktionen noch mehr Möglichkeiten anbietet.
Die Speicherkomponente zum Speichern und Laden von Bildern und Dateien in der Anwendung besteht bei allen Anbietern aus einem Objektspeicher, der Daten in einem Bucket speichert. Dieser Objektspeicher ist bei Firebase und AWS Amplify unabhängig von der Persistenz, Dateien können per SDK oder anderer Schnittstellen wie REST hoch- und heruntergeladen werden. Bei Back4App hingegen ist der Storage an die Datenbank gekoppelt, wenn Dateien nicht mit einem Objekt in der Datenbank verknüpft werden sind diese Dateien nicht mehr auffindbar. Dies stellt zwar sicher, dass die grundlegende Struktur aus Daten und Dateien konsistent ist, aber es trägt auch dazu bei, dass die Flexibilität des Backends sinkt. Der Zugriff auf die Dateien im Storage erfolgt bei Back4App jedoch wiederum nicht über die Datenbank, sondern über eine URL, die in den Metadaten der Datei in der Datenbank enthalten ist.
In Bezug auf das Empfehlungssystem wurde bereits erwähnt, dass eine Umsetzung nicht möglich war. Firebase und AWS bieten jedoch jeweils die Möglichkeit zur Implementierung eines solchen Systems, Firebase bietet den Machine Learning Dienst an während AWS mit dem Personalize Dienst sogar einen konkreten Dienst zur Generierung von Empfehlungen bietet. Bei Back4App fehlt dieser Aspekt gänzlich, sodass in diesem Bereich AWS den größten Mehrwert bietet, da bei Firebase das Machine Learning Modell selbst erstellt werden muss und Back4App gar keinen Dienst für diese Anforderung anbietet.

Im Bereich der Leistungseffizienz zeigt sich bei Betrachtung des Diagramms, dass AWS bei nahezu allen gemessenen Operation die geringste Ausführungszeit in Anspruch nimmt Die Werte von Firebase bewegen sich hier bei zwei Funktionen im mittleren zweistelligen Millisekunden Bereich, was für den Nutzer einer mobilen App weiterhin das Gefühl einer Echtzeitdatenverarbeitung erzeugt. Bei der Erstellung neuer Nutzer und beim Laden eines Bildes aus dem Storage sind die Ausführungszeiten jedoch hoch. Bei der Registrierung eines Nutzers mag dies noch nicht negativ auffallen, aber beim Laden von Bildern aus dem Storage ist dies kaum vertretbar. Auch Back4App sticht bei den Messwerten negativ hervor. Dies ist auf zwei einfach Gründe zurückzuführen: Zum Einen ist im kostenlosen Plan von Back4App das Bereitstellen des Backends nur in den USA möglich, was zu einer erhöhten Latenz führt und zum Anderen konnten auch die Live Queries



Quelle: Eigene Darstellung

Abbildung 6.1: Übersicht der Leistungseffizienz verschiedener Funktionen in der mobilen App

nicht aktiviert werden, da sie nicht im kostenlosen Plan verfügbar sind. Das Abschneiden von Back4App lässt sich an dieser Stelle also nicht abschließend bewerten. AWS hingegen zeigt eine außerordentlich hohe Performanz. Grundlage dafür ist der bereits erwähnte DataStore, der auf mobile Anwendungen und Webanwendungen zugeschnitten wurde. Dabei steht die Schnelligkeit von Datenbankabfragen im Vordergrund, was damit zu tun hat, dass der DataStore zu jeder Zeit eine lokale Version der Datenbank vorhält. Dies beschleunigt den Zugriff auf Daten enorm. Der Zugriff auf den Storage ist bei Back4App hingegen sehr schnell. Da keine genauen Hintergründe der Architektur des bereitgestellten Parse Servers bekannt sind, lässt sich an dieser Stelle nur mutmaßen, dass die Storage Komponenten im Gegensatz zum Parse Server nicht in der Region USA bereitgestellt wird. Insgesamt zeigt also AWS die beste Performanz in diesem Vergleich, während die Messungen von Back4App nicht abschließend bewertet werden können. Firebase bewegt sich bei den meisten Messungen im Mittelfeld, während zwei Operationen negativ hervorstechen. Die Gründe können vielfältig sein. Es könnten Fehler im Bereich der Implementierung oder Konfiguration vorliegen. Da aber alle Funktionen der mobilen App, die sich auf das Backend beziehen, auf gleiche Weise implementiert wurden sind die

Messergebnisse zumindest überraschend. Eine Verwendung von AWS Amplify wäre hier in jedem Fall vorzuziehen.

Die Benutzbarkeit in Bezug auf die Dokumentation und die Bedienbarkeit ist bei den untersuchten Anbietern sehr unterschiedlich. Firebase und AWS bieten eine ausführliche Dokumentation mit vielen Guides und Tutorials an. Es fällt aber auf, dass die von AWS bereitgestellte Dokumentation für Amplify weniger strukturiert ist als die von Firebase. Dienste werden teilweise abhängig von der Plattform dargestellt und sind so gelegentlich schlecht zu finden. Back4App hingegen bietet eine vergleichsweise schlechte Dokumentation der Dienste. Erklärungen fehlen an vielen Stellen, es werden fast ausschließlich Guides für die Implementierung angeboten. Auch die Struktur fällt negativ auf. Dies kann bei der Implementierung zu erheblichen Nachteilen führen. Wenn wichtige Funktionalitäten nicht erklärt werden vergrößert sich der Aufwand für den Entwickler. Außerdem trägt dies auch dazu bei, dass Entwickler ausgiebiger online nach Informationen suchen müssen. Da Back4App bzw. Parse jedoch vergleichsweise wenig genutzt wird, sind die Ergebnisse einer Suche begrenzt. Firebase und AWS haben hier den Vorteil, dass mehr Menschen diese Anbieter verwenden, was auch zu mehr online abrufbaren Informationen führt [Zhang, 2022]. Somit eignet sich Back4App in Bezug auf diesen Bereich nur bedingt zur Implementierung, gerade, wenn die technischen Kenntnisse des Entwicklers begrenzt sind.
Bei der Bedienbarkeit zeigen die Anbieter wiederum viele Gemeinsamkeiten. Alle Anbieter bieten dem Entwickler eine oder mehrere Schnittstellen zur Erstellung und Konfiguration der Dienste an. Firebase kombiniert dabei ein Web Interface und eine CLI, die sich gegenseitig in der Funktionalität hervorragend ergänzen. Das Web Interface ist hochgradig übersichtlich und bietet ein integriertes Monitoring aller Dienste. Bei Amplify hingegen ist das Web Interface nur eingeschränkt funktional, dafür ist die angebotene CLI in Bezug auf die Funktionalität lobenswert. Jeder Dienst kann detailliert konfiguriert werden. Das Bereitstellen der Dienste wird dann mittels eines Infrastructure as Code Dienstes umgesetzt. Back4App bietet ein Web Interface und die CLI, wobei die CLI nur wenig Funktionalität bereitstellt. Da keine Dienste eingerichtet werden müssen, konzentriert sich das Web Interface auf die Darstellung der Daten in der Datenbank. Das Hinzufügen von Cloud Functions ist nur mittels des Web Interfaces erreichbar. Alle Anbieter lassen sich also auf ihre eigene Art gut bedienen und setzen die benötigte Funktionalität um, wobei Firebase das stimmigste Gesamtkonzept bietet. Die Kombination aus Web Interface und CLI wirkt überlegt, alle Dienste und Informationen lassen sich

jederzeit schnell erreichen. AWS bietet dafür jedoch eine äußerst mächtige CLI, mittels der Dienste umfangreich konfiguriert werden können.

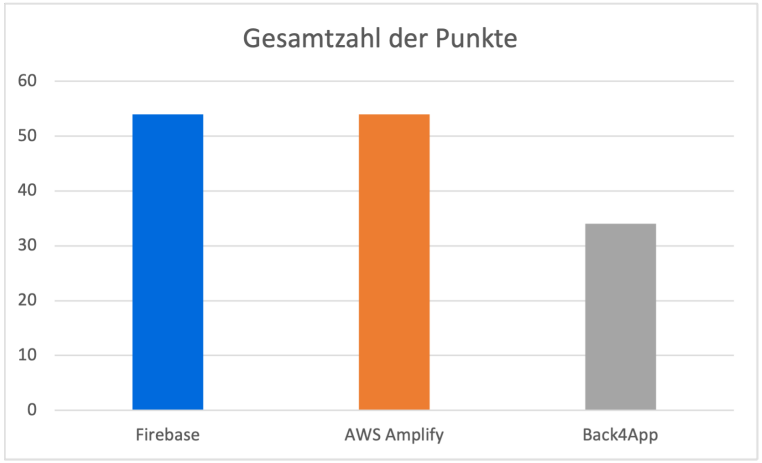
Die Verlässlichkeit lässt sich vergleichsweise weniger gut beurteilen, vor allem in Bezug auf die Verfügbarkeit. Back4App stellt keinerlei Informationen bezüglich der Service Level Agreements dar, es lässt sich nur mutmaßen, dass die SLAs von AWS auch für Back4App gelten, da der Parse Server bei AWS gehostet wird. AWS und Firebase bieten für alle Dienste eine Verfügbarkeit von mindestens 99,9 % an.
AWS und Firebase skalieren bei den Cloud Functions automatisch, bei den anderen Diensten kann eine Skalierung eingestellt werden oder ist bereits automatisch verfügbar. Für Back4App lässt sich die Skalierbarkeit nur schwierig bewerten, da hierzu kaum Informationen verfügbar sind. Aber da in jedem der verfügbaren Kostenpläne Instanzen eines Parse Servers bereitgestellt werden, ist eine rein vertikale Skalierung wahrscheinlicher. Dies kann in den verschiedenen Kostenplänen dazu führen, dass bei hoher Last Limitationen bezüglich der Leistung erreicht werden. Alle Anbieter bieten außerdem die Möglichkeit, Backups der Daten der Anwendung zu erstellen. Dies geschieht bei allen Anwendern automatisch, sodass eine einfache Disaster Recovery gewährleistet ist. Die automatische Skalierung und die festgeschriebene Verfügbarkeit spricht für die Verwendung von Firebase und AWS Amplify, wobei die Unterschiede der Anbieter in diesem Bereich vergleichsweise gering sind.

Die Wartbarkeit wurde hinsichtlich des Monitorings und der Debugging Möglichkeiten der Anbieter untersucht. Firebase stellt im Web Interface viele Informationen dar, die einzelnen Dienste lassen sich genauso überwachen wie die mobile App an sich, bspw. bezüglich der Startzeit der App. Durch die Integration mit der Google Cloud lassen sich hier auch umfangreichere Reportings und Dashboards erstellen. Bei AWS Amplify wird das Monitoring mittels des AWS Dienstes CloudWatch umgesetzt. Dies ist wie das Monitoring in der Google Cloud umfangreich. Für alle Dienste sind diverse Metriken verfügbar, auch Dashboards lassen sich automatisch erstellen. Bei Back4App konzentriert sich das Monitoring auf das Nutzerverhalten und die Datenbank. Für Dienste wie bspw. die Cloud Functions ist gar kein Monitoring vorhanden und in Bezug auf die anderen Dienste stark limitiert.
Firebase und AWS Amplify bieten dem Entwickler außerdem eine lokale Testumgebung, mit der Dienste vor dem Bereitstellen getestet werden können. Dies erleichtert die Konfiguration enorm, da nicht auf Produktionssysteme zugegriffen werden muss. Firebase bietet eine lokale Emulator Suite, bei Amplify lassen sich die einzelnen Dienste simulieren. Back4App bietet diese Möglichkeit nicht, lokal lässt sich zwar ein Parse Server

starten, um das Verhalten des Backends zu simulieren, aber auf Grund des hohen Konfigurationsaufwandes ist dies nicht so leicht umzusetzen wie bei den anderen Anbietern. Diese Tatsachen sprechen insgesamt für die Verwendung von Firebase oder AWS Amplify, wobei die Emulator Suite von Firebase noch mehr Möglichkeiten bietet als das Mocking von AWS Amplify. Auch das Monitoring von Firebase bietet dem Entwickler viele Möglichkeiten zur Überwachung der eigenen Anwendung. Einfach zu verstehende Diagramme und Metriken für die einzelnen Dienste werden hervorragend durch das detaillierte Monitoring der Google Cloud ergänzt. Back4App hingegen bietet kaum Überwachungsmöglichkeiten.

Die Sicherheit der Daten einer Anwendung ist von hoher Bedeutung. Hier zeigt vor allem AWS Amplify große Stärken: Beim Bereitstellen der Dienste werden entsprechend der Konfiguration Richtlinien (Policys) erstellt, die den Zugriff auf die Dienste regeln. Der Entwickler muss hier also keine eigenen Zugriffsrichtlinien erstellen. Außerdem können die Daten des Datenmodells mittels erstellter Regeln zusätzlich gesichert werden, damit bspw. nur authentifizierte Nutzer Zugriff erhalten. Auch Firebase bietet solche Regeln für alle Dienste, bei denen Daten im Mittelpunkt stehen. Für einige Dienste wurde sogar eine umfangreiche Regelsprache entwickelt, die feingranular den Zugang zu den Daten administriert. So kann auch die Authentifizierung in den Zugriff auf die Daten integriert werden. Die Access Control Lists von Back4App sind im Gegensatz dazu nicht sehr umfangreich und der Zugang zu den Daten lässt sich nur grob festlegen. Firebase und AWS bieten hier also weitaus umfangreichere Sicherheitskonzepte, sodass eine zukünftige Verwendung von Back4App als Backend kaum in Frage kommt. Dieser Eindruck verstärkt sich noch durch immer strengere Datenschutzrichtlinien und auch das Bedürfnis der Nutzer nach mehr Sicherheit. AWS Amplify ist in diesem Fall Firebase vorzuziehen, denn dem Entwickler wird bei der Einrichtung der Dienste von der CLI einiges an Aufwand abgenommen. Auch der Zugriff der Dienste untereinander wird so eingeschränkt, was für erhöhte Sicherheit sorgt.

Abschließend lässt sich für den Vergleich ein doch recht deutliches Gefälle zwischen den etablierten Cloud Anbietern und Back4App feststellen. AWS mit Amplify und Google mit Firebase bieten in vielen Bereichen der untersuchten Kriterien überlegene Technologien zu einem günstigeren Preis. Einzig die Tatsache, dass Back4App auf dem Open Source Framework Parse basiert, kann eine Nutzung rechtfertigen, denn damit wird einem der größten Kritikpunkte von BaaS begegnet: Dem Vendor lock-in.



Quelle: Eigene Darstellung

Abbildung 6.2: Übersicht über die Gesamtzahl der vergebenen Punkte

In Abbildung 6.2 ist die Gesamtzahl der vergebenen Punkte für jeden Anbieter dargestellt. Dabei bestätigt sich der oben erwähnte Eindruck: Zwischen Firebase bzw. AWS Amplify und Back4App ist ein großes Gefälle sichtbar. Firebase und AWS Amplify kommen insgesamt auf die gleiche Anzahl an Punkten. Die Unterschiede sind hier nur in Teilbereichen des Vergleiches erkennbar und beide eignen sich insgesamt sehr gut, um als Backend der Evender App zu dienen. Auf Grund der besseren Sicherheitsmaßnahmen und des ausgereiften Datenmodells bzw. des leistungsfähigen DataStores schneidet AWS Amplify aber in wichtigen Bereichen des Kriterienkataloges noch besser ab als Firebase und wäre die richtige Wahl als BaaS Anbieter.

6.2 Limitationen

Limitationen sind im Rahmen der Umsetzung kaum aufgetreten, jedoch stößt Serverless und das BaaS Konzept an sich in manchen Bereichen an Grenzen. Durch die Verwendung kann es zu einem Vendor lock-in kommen. Je weiter die Entwicklung mittels eines dieser Anbieter voranschreitet desto schwieriger wird auch eine spätere Migration der Anwendung. Dadurch, dass verschiedene Anbieter die Dienste unterschiedlich implementieren lassen sie sich oft nur schwierig übertragen [Sbarski und Kroonenburg, 2017]. Auch der Verlust an Kontrolle kann problematisch werden: Es lässt sich schwierig einschätzen, welche Kosten wirklich bei einer produktiven Verwendung entstehen, sodass hier die Gefahr einer Kostenexplosion besteht [Roberts, 2018]. Weiterhin bringen die standardisierten

Dienste eine Verlust an möglicher Konfiguration mit sich. Dies kann in Zukunft dazu führen, dass Innovationen in der Architektur von Evender ausgebremst werden, wenn ein Anbieter möglicherweise keine passenden Dienste anbietet. Auch das Debugging und Testen wird in der Literatur als Herausforderung gesehen. Da das Serverless Konzept immer noch vergleichsweise neu ist, sind auch die Entwicklungstools, Konzepte und Programmiermodelle nocht nicht ausgereift [Shafiei u. a., 2019]. Dies fiel auch im Rahmen der Implementierung auf: Einzelne Dienste lassen sich zwar teilweise lokal testen und auch debuggen, aber auf Grund der Verbindungen der Dienste zueinander und der Integration in eine mobile App kann dies sehr schnell komplex werden. Eine weiteres Problem betrifft die Sicherheit von BaaS Anwendungen. Eine mobile App oder auch eine Webapp muss sich authentifizieren, um Zugang zum Backend zu erhalten. Dies funktioniert bei allen drei untersuchten Anbietern mittels einer ID und eines Secret Keys, die in der App hinterlegt werden. Somit besteht zumindest theoretisch die Möglichkeit, dass diese Zugangsdaten extrahiert und missbraucht werden könnten [Rasthofer u. a., 2016]. Roberts und Chaping sind jedoch der Auffassung, dass den meisten dieser Limitierungen in der Zukunft begegnet werden kann. Serverless Plattformen würden wachsen und und mit der Zeit ausgereifter werden. Auch die Bereitstellung und Konfiguration würde leichter werden, genauso wie sich das Monitoring weiterentwickeln würde [Roberts, 2018]. Auch eine Standardisierung der Dienste und Plattformen könnte in Zukunft dazu beitragen, den verschiedenen Problemen von Serverless und BaaS zu begegnen [Koschel u. a., 2021].

6.3 Implementierung

Die Umsetzung der geforderten Anwendung lief in nahezu allen Bereichen problemlos. Das konzipierte Backend ließ sich bei allen Anbietern umsetzen, bis auf das schon angesprochene Empfehlungssystem. Auf Grund fehlender Daten von Nutzern war es nicht möglich, hier ein vollständiges Empfehlungssystem zu implementieren. Hier zeigt sich allerdings, dass dies bei einer Verwendung von Back4App auch zukünftig nicht möglich sein wird, weil Back4App hierfür keinen Dienst anbietet. Dass Back4App trotzdem zur Entwicklung ausgewählt wurde liegt daran, dass zu Beginn der Arbeit die Umsetzung eines vollständigen Empfehlungssystems noch nicht geplant war. Weiterhin kam es bei Back4App zu zusätzlichen Limitierungen. Im kostenlosen Plan ist nur eine begrenzte Anzahl an Anfragen an das Backend enthalten. Zum Testen der Leistungseffizienz wurden viele Testveranstaltungen in die Datenbank eingefügt, was dazu führte, dass dieses Limit überschritten wurde. Die Anwendung wurde daraufhin vollständig deaktiviert und ließ

sich nicht mehr erreichen. Des Weiteren wäre es auch sinnvoll gewesen, die Nutzersuche mit einer höheren Anzahl an Nutzern zu testen, dies war jedoch nicht möglich, da Back4App in der Datenbank nur authentifizierte Nutzer zulässt. Firebase und Amplify besitzen keine solchen festgelegten Limitationen. Dies hängt aber auch mit dem Kostenmodell zusammen: Bei Firebase und Amplify besteht die Gefahr der oben erwähnten Kostenexplosion wegen das Pay as you go Modells. Wenn das Empfehlungssystem außer Acht gelassen wird, so eignen sich alle untersuchten Anbieter für die Verwendung mit der mobilen App von Evender. Dass bei Back4App das Empfehlungssystem extern umgesetzt werden müsste stellt prinzipiell kein Problem dar, jedoch wäre bei Firebase und Amplify die Entwicklungszeit dafür deutlich verkürzt, sodass sich diese Anbieter für eine schnellere Umsetzung dieses Features eher eignen. Außerdem war das Implementieren der Cloud Functions bei allen Anbietern aufwändig. Firebase und Amplify bieten zwar die Möglichkeit eines lokalen Testens und Debuggens, aber der Aufwand ist hoch im Vergleich zum Debugging eines Programms, das lokal läuft. Bei Back4App war dies noch schwieriger, da lokal ein Parse Server aufgesetzt werden müsste und zudem ist der Logging Bereich auf den gesamten Parse Server bezogen. Logs zu einzelnen Funktionen lassen sich nicht herausfiltern. Weiterhin ließen sich die erstellten Cloud Functions im Bereich der Leistungseffizienz nicht oder nur schlecht vergleichen, weshalb auch keine Messungen vorgenommen wurden. Dies liegt daran, dass innerhalb der Funktionen bei jedem Anbieter auf unterschiedliche Weise auf andere Dienste wie den Storage zugegriffen wird. Hier wäre es im Nachhinein sinnvoller gewesen, die einzelnen Funktionen noch kleinteiliger zu konstruieren, denn so wären diese letztendlich auch vergleichbar gewesen. Es entstanden zudem auch Kosten bei der Verwendung der Backendanwendungen, jedoch nur bei Firebase und Amplify, was den Vorteil von Back4App in Bezug auf das starre Kostenmodell untermauert. Durch einen Programmierfehler in der Firebase Cloud Function zum Laden der Veranstaltungen in die Datenbank wurde jede Veranstaltung 200 Mal hinzugefügt. Dies führte beim Abfragen der Daten zu einem erhöhten Datentransfer, was zu Kosten in Höhe von 4 Euro führte. Bei Amplify ist die Anzahl der Abfragen an den Storage begrenzt: Durch das Hinzufügen von Testveranstaltungen zur Durchführung der Messungen entstanden auch hier Kosten in Höhe von 50 Cent. Überdies hinaus gestaltete sich an manchen Stellen die Integration des Backends in die mobile App schwieriger als gedacht, da dafür entsprechende Kenntnisse in Swift vorausgesetzt werden, die zu Beginn der Arbeit noch nicht vorhanden waren.

7 Fazit und Ausblick

Das Hauptziel dieser Arbeit war es, einen Einblick in die Funktionsweise eines BaaS zu erlangen und bei verschiedenen Anbietern zu implementieren, um die Eignung dieses Konzeptes in Bezug auf den dargestellten Anwendungsfall von Evender beurteilen zu können. Dafür wurden die Anforderungen an die mobile App erhoben, um daraus die Anforderungen an eine Backendanwendungen ableiten zu können und zu spezifizieren. Im Entwurf wurde eine Zielarchitektur entworfen, die es dann bei allen Anbietern umzusetzen galt. Die Anbieter für den Vergleich wurden entsprechend der Anforderungen ausgewählt und es wurde ein Kriterienkatalog entwickelt, anhand dessen im Folgenden der Vergleich der resultierenden Backendanwendungen durchgeführt werden konnte.

Im Vergleich der Anbieter ergab sich, dass sich alle implementierten Backendanwendungen grundsätzlich dazu eignen, als Backend der mobilen Evender App zu fungieren. Im Rahmen des Vergleiches traten jedoch gravierende Unterschiede der Anbieter untereinander auf. Google Firebase und AWS Amplify sind dabei in größerem Ausmaß für die Verwendung geeignet als Back4App, wobei insgesamt Amplify von AWS den besten Eindruck hinterlässt. Die implementierten Backendanwendungen zeigen weiterhin, dass auch das BaaS Konzept an sich für den dargestellten Anwendungsfall geeignet ist. Es konnten innerhalb einer angebrachten Zeitspanne drei verschiedene Backends entwickelt werden, die zudem auch noch größtenteils wenig bis keine Kosten verursacht haben. Somit konnte tatsächlich genau das erreicht werden, was sich durch die Verwendung eines BaaS erhofft wurde.

Das geforderte Empfehlungssystem konnte zwar bei keinem der Anbieter umgesetzt werden, jedoch bieten Google Firebase und AWS Amplify immerhin die Möglichkeit, solch ein System mittels eines Dienstes zu implementieren. Alle außerdem erhobenen Anforderungen ließen sich bei allen Anbietern problemlos umsetzen. Bei der Evaluation der Kosten zeigte sich, dass die beiden großen Cloud Anbieter günstiger sind. Back4App bietet jedoch einen entscheidenden Vorteil, denn das System basiert auf dem Open Source Framework Parse, sodass der möglicherweise entstehende Vendor lock-in bei Back4App

nicht besteht. In Bezug auf die Dienste spielen Google Firebase und AWS Amplify ihre Stärken aus: Druch die Integration mit den Cloud Plattformen Google Cloud und AWS kann eine Vielzahl an Diensten angeboten werde. Back4App kann hier nur grundlegende Funktionalitäten bieten. Bei AWS Amplify fällt das ausgereifte Datenmodell besonders auf: Die Daten der Anwendung werden im Vorhinein basierend auf GraphQL modelliert und die Synchronisierung funktioniert mit dem Middleware Dienst AWS AppSync. Die Erstellung der Datenbank und entsprechenden Tabellen übernimmt dann AWS Amplify für den Entwickler. Die Datenmodelle von Firebase und vor allem von Back4App wirken weniger durchdacht, obwohl beide verschiedene Datenbanktypen zur Verfügung stellen. Dies zeigt sich auch in der Auswertung der Leistungseffizienz: Die AWS Dienste schneiden hier am Besten ab, jedoch ist die Vergleichbarkeit zu Back4App nicht unbedingt gegeben: Im kostenlosen Plan konnten die Dienste von Back4App nur in den USA bereitgestellt werden, was zu einer höheren Latenz führt. Auch bei der Benutzbarkeit zeigten sich große Unterschiede, vor allem in Bezug auf die Dokumentationen der Anbieter. Back4App fällt hier negativ mit einer unstrukturierten und wenig ausführlichen Dokumentation auf. Hingegen konnte die CLI von AWS Amplify überzeugen: Dienste können vor dem Bereitstellen umfangreich konfiguriert werden und das Erstellen von Sicherheitsrichtlinien für Zugriffe auf Daten und Dienste wird automatisiert. Bei der Verlässlichkeit zeigten sich keine großen Differenzen der Anbieter, aber die Wartbarkeit in Bezug auf das Debuggen ist bei Google Firebase und AWS Amplify durch lokale Emulatoren eindeutig besser als bei Back4App. Auch die Sicherheit, vor allem in Bezug auf die Daten innerhalb der Anwendung, ist bei Google Firebase und AWS Amplify positiv hervorzuheben. Es lassen sich für alle Dienste, die mit Daten arbeiten, Regeln erstellen. Amplifys CLI erstellt außerdem zusätzliche Zugriffsrichtlinien für die Dienste untereinander. Dies erlaubt eine feingranulare Zugriffskontrolle für die Daten.

Die Arbeit konnte somit die Frage nach der Eignung verschiedener BaaS Systeme und der grundsätzlichen Eignung eines BaaS beantworten. Die Umsetzung lief in vielen Bereichen problemlos, einzig das fehlende Empfehlungssystem sticht negativ hervor. Hier hätte schon bei Erhebung der Anforderungen klar sein können, dass die Umsetzung auf Grund fehlender Daten schwierig werden könnte. Abgesehen davon bietet die Arbeit gerade für Evender als junges Start-up einen umfangreichen Mehrwert, denn eine solche Evaluation verschiedener Systeme ist meist aufwändig und schwierig. Auch der eigene Lernfaktor darf nicht unterschätzt werden: Im Rahmen der Arbeit wurden viele, auch unbekannte Technologien verwendet, was zu einer persönlichen Weiterentwicklung führte.

Ausblick

Diese Arbeit bildet eine gute Grundlage für die weitere Arbeit von Evender. Es wurden drei funktionierende Backendanwendungen implementiert, die produktiv verwendet werden könnten. In naher Zukunft wird das Team die Ergebnisse dieser Arbeit in die Entscheidungsfindung bezüglich der Auswahl eines BaaS einfließen lassen. Im Rahmen der Arbeit wurden jedoch nur drei Vertreter des BaaS Konzeptes von vielen untersucht. Somit kann es Sinn ergeben, auch andere Anbieter zu evaluieren, was mit dieser Arbeit als Grundlage leichter fallen sollte. Mittels eines ausgewählten Systems soll dann der Launch der mobilen App Anfang des Jahres 2023 erfolgen. Auf dem Weg dahin wird der Fokus im Backend vor allem auf der Integration weiterer Quellsysteme liegen.

BaaS weist insgesamt ein großes Potenzial auf. Als noch vergleichsweise junge Technologie gibt es noch einige Herausforderungen wie bspw. den Vendor Lock-in oder fehlende Kontrolle, denen begegnet werden muss. Aber durch immer mehr Mobilgeräte und somit auch mobilen Apps wird die Verwendung von BaaS Systemen weiter zunehmen. Ob klassische Modelle wie die eigenständige Entwicklung eines Backends vollständig abgelöst werden lässt sich nicht vorhersagen, jedoch können die vorgestellten Konzepte in vielen Bereichen enorme Vorteile aufweisen und tragen so zu einer Weiterentwicklung und Festigung neuer Strategien bei.

Literaturverzeichnis

[Apple 2022] Apple: *Swift*. https://www.apple.com/de/swift/. 2022. – Accessed: 2022-08-04

[AWS 2022a] AWS: *Amazon DynamoDB Preise*. https://aws.amazon.com/de/dynamodb/pricing/. 2022. – Accessed: 2022-10-30

[AWS 2022b] AWS: *Amazon Personalize - Getting started*. https://docs.aws.amazon.com/personalize/latest/dg/getting-started-console.html. 2022. – Accessed: 2022-10-15

[AWS 2022c] AWS: *Amplify Authentication*. https://docs.amplify.aws/lib/auth/overview/q/platform/js/#authentication-with-aws. 2022. – Accessed: 2022-09-20

[AWS 2022d] AWS: *Amplify endpoints and quotas*. https://docs.aws.amazon.com/general/latest/gr/amplify.html. 2022. – Accessed: 2022-09-29

[AWS 2022e] AWS: *Authorization Rules*. https://docs.amplify.aws/cli/graphql/authorization-rules/#global-authorization-rule-only-for-getting-started. 2022. – Accessed: 2022-10-12

[AWS 2022f] AWS: *AWS Amplify*. https://aws.amazon.com/de/amplify/. 2022. – Accessed: 2022-08-26

[AWS 2022g] AWS: *Datastore - How it works*. https://docs.amplify.aws/lib/datastore/how-it-works/q/platform/ios/. 2022. – Accessed: 2022-08-30

[AWS 2022h] AWS: *Funktionen von AWS Lambda*. https://aws.amazon.com/de/lambda/features/. 2022. – Accessed: 2022-09-30

[AWS 2022i] AWS: *How Amplify works with IAM*. https://docs.aws.amazon.com/amplify/latest/userguide/security_iam_service-with-iam.html. 2022. – Accessed: 2022-10-12

[AWS 2022j] AWS: *Lambda Quotas*. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html. 2022. – Accessed: 2022-10-29

[AWS 2022k] AWS: *Managing AWS Regions*. https://docs.aws.amazon.com/general/latest/gr/rande-manage.html. 2022. – Accessed: 2022-10-01

[AWS 2022l] AWS: *Preise für AWS Amplify*. https://aws.amazon.com/de/amplify/pricing/. 2022. – Accessed: 2022-10-29

[AWS 2022m] AWS: *Recording Events*. https://docs.aws.amazon.com/personalize/latest/dg/recording-events.html. 2022. – Accessed: 2022-10-15

[AWS 2022n] AWS: *Regionen und Availability Zones*. https://aws.amazon.com/de/about-aws/global-infrastructure/regions_az/. 2022. – Accessed: 2022-09-29

[AWS 2022o] AWS: *Sicherung und Wiederherstellung*. https://aws.amazon.com/dynamodb/backup-restore/. 2022. – Accessed: 2022-09-14

[AWS 2022p] AWS: *Storage Concepts*. https://docs.amplify.aws/lib/storage/overview/q/platform/js/. 2022. – Accessed: 2022-10-08

[Back4App 2022a] Back4App: *Activating your Live Query*. https://www.back4app.com/docs/platform/parse-live-query. 2022. – Accessed: 2022-09-12

[Back4App 2022b] Back4App: *Back4App Pricing*. https://www.back4app.com/pricing. 2022. – Accessed: 2022-09-04

[Back4App 2022c] Back4App: *Better understand your mobile app with Analytics Tool*. https://www.back4app.com/docs/parse-dashboard/analytics/mobile-app-analytics. 2022. – Accessed: 2022-09-29

[Back4App 2022d] Back4App: *Cloud code local debugging*. https://www.back4app.com/docs/local-development/local-debugging. 2022. – Accessed: 2022-10-06

[Back4App 2022e] Back4App: *Deploy and call your first Cloud Function*. https://www.back4app.com/docs/get-started/cloud-functions. 2022. – Accessed: 2022-10-08

[Back4App 2022f] Back4App: *How to make a Secure App using Parse*. https://www.back4app.com/docs/security/parse-security. 2022. – Accessed: 2022-10-13

[Back4App 2022g] Back4App: *How to setup a local Parse Server*. https://www.back4app.com/docs/local-development/parse-server-local. 2022. – Accessed: 2022-10-05

[Back4App 2022h] Back4App: *Local Development*. https://www.back4app.com/docs/local-development/parse-cli. 2022. – Accessed: 2022-10-26

[Back4App 2022i] Back4App: *Read and write Data*. https://www.back4app.com/docs/get-started/read-and-write-data. 2022. – Accessed: 2022-09-13

[Back4App 2022j] Back4App: *Relational Schema on Back4App*. https://www.back4app.com/docs/get-started/relational-schema. 2022. – Accessed: 2022-09-10

[Back4App 2022k] Back4App: *Save Files from a React Native App*. https://www.back4app.com/docs/react-native/parse-sdk/working-with-files/react-native-save-file. 2022. – Accessed: 2022-10-14

[Back4App 2022l] Back4App: *Sign In with Facebook*. https://www.back4app.com/docs/ios/parse-swift-sdk/users/sign-in-with-facebook. 2022. – Accessed: 2022-09-23

[Back4App 2022m] Back4App: *Understand user behavior by tracking any event with one line of code*. https://www.back4app.com/docs/parse-dashboard/analytics/mobile-app-event-tracking. 2022. – Accessed: 2022-10-18

[Back4App 2022n] Back4App: *User Registration for React Native*. https://www.back4app.com/docs/react-native/parse-sdk/working-with-users/react-native-user-registration. 2022. – Accessed: 2022-09-22

[Back4App 2022o] Back4App: *Using Cloud Functions in a React Native App*. https://www.back4app.com/docs/react-native/parse-sdk/cloud-functions/react-native-cloud-functions. 2022. – Accessed: 2022-10-09

[Back4App 2022p] Back4App: *What is the data and backup ownership policy?* https://help.back4app.com/hc/en-us/articles/115001377111-What-is-the-backup-and-data-ownership-policy-. 2022. – Accessed: 2022-09-15

[Baldini u. a. 2017] Baldini, Ioana ; Castro, Paul ; Chang, Kerry ; Cheng, Perry ; Fink, Stephen ; Ishakian, Vatche ; Mitchell, Nick ; Muthusamy, Vinod ; Rabbah, Rodric ; Slominski, Aleksander u. a.: Serverless computing: Current trends and open problems. In: *Research advances in cloud computing*. Springer, 2017, S. 1–20

[Berenbrink u. a. 2013] Berenbrink, Verena ; Purucker, Jörg ; Bahlinger, Thomas: Die Bedeutung von Wireframes in der agilen Softwareentwicklung. In: *HMD Praxis der Wirtschaftsinformatik* 50 (2013), Nr. 2, S. 27–34

[Brandt-Pook und Kollmeier 2015] Brandt-Pook, Hans ; Kollmeier, Rainer: *Softwareentwicklung kompakt und verständlich*. Springer, 2015

[Broy und Kuhrmann 2021] Broy, Manfred ; Kuhrmann, Marco: *Einführung in die Softwaretechnik*. Springer, 2021

[Buxmann u. a. 2008] Buxmann, Peter ; Hess, Thomas ; Lehmann, Sonja: Software as a Service. In: *Wirtschaftsinformatik* 50 (2008), Nr. 6, S. 500–503

[Clark 2022] Clark, Jessica: *Parse BaaS*. https://blog.back4app.com/parse-baas/. 2022. – Accessed: 2022-09-04

[Eismann u. a. 2020] Eismann, Simon ; Scheuner, Joel ; Van Eyk, Erwin ; Schwinger, Maximilian ; Grohmann, Johannes ; Herbst, Nikolas ; Abad, Cristina L. ; Iosup, Alexandru: Serverless applications: Why, when, and how? In: *IEEE Software* 38 (2020), Nr. 1, S. 32–39

[Firebase] Firebase: *Firebase Pricing*. https://firebase.google.com/pricing. – Accessed: 2022-09-03

[Firebase 2022a] Firebase: *Add recommendations to your app with Tensorflow Lite*. https://firebase.google.com/codelabs/contentrecommendation-ios. 2022. – Accessed: 2022-10-15

[Firebase 2022b] Firebase: *Automated Backups*. https://firebase.google.com/docs/database/backups. 2022. – Accessed: 2022-09-12

[Firebase 2022c] Firebase: *Choose a Database: RTDB vs Firestore*. https://firebase.google.com/docs/database/rtdb-vs-firestore. 2022. – Accessed: 2022-08-28

[Firebase 2022d] Firebase: *Cloud Functions for Firebase*. https://firebase.google.com/docs/functions. 2022. – Accessed: 2022-09-06

[Firebase 2022e] Firebase: *Cloud Storage for Firebase*. https://firebase.google.com/docs/storage. 2022. – Accessed: 2022-10-12

[Firebase 2022f] Firebase: *Firebase Authentication*. https://firebase.google.com/docs/auth. 2022. – Accessed: 2022-09-16

[Firebase 2022g] Firebase: *Firebase iOS Codelab Swift*. https://firebase.google.com/codelabs/firebase-ios-swift#0. 2022. – Accessed: 2022-10-22

[Firebase 2022h] Firebase: *Firebase Locations*. https://firebase.google.com/docs/projects/locations. 2022. – Accessed: 2022-09-30

[Firebase 2022i] Firebase: *Firebase security rules*. https://firebase.google.com/docs/rules. 2022. – Accessed: 2022-10-10

[Firebase 2022j] Firebase: *Google Firebase*. https://firebase.google.com/. 2022. – Accessed: 2022-08-25

[Firebase 2022k] Firebase: *Manage Functions*. https://firebase.google.com/docs/functions/manage-functions. 2022. – Accessed: 2022-09-28

[Firebase 2022l] Firebase: *Quotas and Limits*. https://firebase.google.com/docs/functions/quotas. 2022. – Accessed: 2022-09-25

[Firebase 2022m] Firebase: *Quotas and Limits*. https://firebase.google.com/docs/functions/quotas. 2022. – Accessed: 2022-10-09

[Firebase 2022n] Firebase: *Read and Write Data on Apple Platforms*. https://firebase.google.com/docs/database/ios/read-and-write. 2022. – Accessed: 2022-08-29

[Firebase 2022o] Firebase: *Realtime Database Limits*. https://firebase.google.com/docs/database/usage/limits. 2022. – Accessed: 2022-10-08

[Firebase 2022p] FIREBASE: *Security rules language*. https://firebase.google.com/docs/rules/rules-language#database. 2022. – Accessed: 2022-10-10

[Firebase 2022q] FIREBASE: *Usage and Limits*. https://firebase.google.com/docs/database/usage/limits. 2022. – Accessed: 2022-10-09

[Friedrichsen 2022] FRIEDRICHSEN, Uwe: *The Public Cloud Revolution*. https://www.ufried.com/blog/public_cloud_revolution_2/. 2022. – Accessed: 2022-08-04

[Holt u. a. 2011] HOLT, Eva-Maria ; WINTER, Dominique ; THOMASCHEWSKI, Jörg: Personas als Werkzeug in modernen Softwareprojekten. In: *Tagungsband UP11* (2011)

[ISO/IEC 2011] ISO/IEC: *ISO/IEC 25010:2011(en)*. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. 2011. – Accessed: 2022-08-05

[Janson 2017] JANSON, Matthias: *Darum löschen Deutsche Apps von ihrem Handy*. https://de.statista.com/infografik/12297/gruende-fuer-app-deinstallation/. 2017. – Accessed: 2022-11-30

[javaTpoint] JAVATPOINT: *Firebase History*. https://www.javatpoint.com/firebase-introduction. – Accessed: 2022-08-10

[Koschel u. a. 2021] KOSCHEL, Arne ; KLASSEN, Samuel ; JDIYA, Kerim ; SCHAAF, Marc ; ASTROVA, Irina: Cloud computing: Serverless. In: *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)* IEEE (Veranst.), 2021, S. 1–7

[Krzmar 2015] KRZMAR, Helmut: *Informationsmanagement*. Springer-Verlag, 2015. – ISBN 9783662458631

[Mell u. a. 2011] MELL, Peter ; GRANCE, Tim u. a.: The NIST definition of cloud computing. (2011)

[Nandyal und Rafi 2020] NANDYAL, Aslam B. ; RAFI, Mohammed: Determining Feature Gaps Of Open Source Cloud Platforms for Mobile Backend as service (MBaaS) in Enterprise Mobile Applications. In: *2020 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)* IEEE (Veranst.), 2020, S. 200–205

[Rasthofer u. a. 2016] RASTHOFER, Siegfried ; ARZT, Steven ; HAHN, Robert ; KOLHAGEN, Max ; BODDEN, Eric: (In)Security of Backend-as-a-Service. (2016)

[Reinheimer 2018] Reinheimer, Stefan: *Cloud Computing: Die Infrastruktur der Digitalisierung*. Springer-Verlag, 2018

[Roberts und Chapin 2017] Roberts, Michael ; Chapin, John: *What is Serverless?* O'Reilly Media, Incorporated, 2017

[Roberts 2018] Roberts, Mike: *Serverless Architectures*. https://martinfowler.com/articles/serverless.html. 2018. – Accessed: 2022-08-04

[Saha 2022] Saha, Sudip: *Backend-As-A-Service Market*. https://www.futuremarketinsights.com/reports/backend-as-a-service-baas-market. 2022. – Accessed: 2022-11-09

[Sbarski und Kroonenburg 2017] Sbarski, Peter ; Kroonenburg, Sam: *Serverless architectures on AWS: with examples using Aws Lambda*. Simon and Schuster, 2017

[Shafiei u. a. 2019] Shafiei, Hossein ; Khonsari, Ahmad ; Mousavi, Payam: Serverless Computing: A Survey of Opportunities, Challenges, and Applications. In: *ACM Computing Surveys (CSUR)* (2019)

[Stanoevska u. a. 2009] Stanoevska, Katarina ; Wozniak, Thomas ; Ristol, Santi: *Grid and cloud computing: a business perspective on technology and applications*. Springer Science & Business Media, 2009

[Vaquero u. a. 2008] Vaquero, Luis M. ; Rodero-Merino, Luis ; Caceres, Juan ; Lindner, Maik: *A break in the clouds: towards a cloud definition*. 2008

[Yoder und Shriver 2022] Yoder, Daniel ; Shriver, Sean: *Amazon DynamoDB Auto scaling*. https://aws.amazon.com/blogs/database/amazon-dynamodb-auto-scaling-performance-and-cost-optimization-at-any-scale/. 2022. – Accessed: 2022-10-28

[Zhang 2022] Zhang, Mary: *Top 10 Cloud Service Providers Globally in 2022*. https://dgtlinfra.com/top-10-cloud-service-providers-2022/. 2022. – Accessed: 2022-10-31

Anhang

Initiale Menge an User Storys, die im Evender Workshop erarbeitet wurden

User:

- U1: Als User möchte ich eine an Layout in der App, damit ich Spaß beim Verwenden habe.
- U2: Als User möchte ich ein einfaches und intuitives Layout haben, damit das Onboarding schneller geht.
- U3: Als User möchte ich auf einer Karte Veranstaltungen in meiner Nähe sehen, damit ich nicht extra nach einem Gebiet suchen muss.
- U4: Als User möchte ich auf einer karte meine favorisierten Veranstaltungen sehen, damit ich genau weiß wie weit entfernt sie von mir sind
- U5: Als User möchte ich den Ort und Radius angeben können, in dem ich Suche.
- U6: Als User möchte ich Newsletter erhalten können, damit ich über Events up-to-date bleibe.
- U7: Als User möchte ich ein Incentive, wenn ich Evender nutze (z.B. Gästelistenplatz, Rabatt)
- U8: Als User möchte ich Events in meinem Kalender übertragen können, damit ich immer weiß wann sie stattfinden.
- U9: Als User möchte ich unbegrenzte Swipe Möglichkeiten haben, um die volle Bandbreite zu sehen
- U10: Als User möchte ich bereits geswipte Events wiederherstellen, um die Möglichkeit zu haben an allen Events teilnehmen zu können

- U11: Als User möchte ich Events filtern können, damit meine Auswahl nicht so groß ist.
- U12: Als User möchte ich Events zu denen ich gehen möchte speichern können, damit die Informationen zu dem Event für mich immer zugänglich sind.
- U13: Als User möchte ich eine KI haben, die mir automatisch Events, die zu mir passen, vorschlägt.
- U14: Als User möchte ich auf dem Homebildschirm Events swipen können, um Vorschläge für mögliche coole Events zu bekommen.
- U15: Als User möchte ich Informationen zu einem Event bekommen für das ich mich interessiere, damit ich einschätzen kann ob das Event gut ist.
- U16: Als User möchte ich internationale Events angezeigt bekommen, damit ich auch im Urlaub Inspirationen bekommen. Insbesondere an Orten. an denen ich mich nicht auskenne.
- U17: Als User möchte ich eine große Bandbreite an Events angezeigt bekommen, damit ich genügend Inspiration erhalte.
- U18: Als User möchte ich Tickets direkt über die App buchen können, damit ich einen geringeren Aufwand habe.
- U19: Als User möchte ich mein Ticket für ein Event buchen können.
- U20: Als User möchte ich mein Ticket in der App speichern können und anzeigen können.
- U21: Als User möchte ich Tickets direkt über die App buchen können, damit ich einen geringeren Aufwand habe.
- U22: Als User möchte ich die Events direkt buchen können/zur Buchung weitergeleitet werden, um nicht noch selbst im Internet danach suchen zu müssen.
- U23: Als User möchte ich Tickets direkt aus der App in mein "Wallet"packen können.
- U24: Als User möchte ich meine Profildaten anpassen können.
- U25: Als User möchte ich Veranstalter bewerten können, damit ersichtlich für andere ist, ob die Events dieses Veranstalters gut sind.

- U26: Als User möchte ich mit einem bestehenden Google Account/Apple ID/FB/EMail einloggen können, damit ich nicht extra ein Konto erstellen muss.
- U27: Als User möchte ich sehen, für welche Events sich meine Freunde interessieren.
- U28: Als User möchte ich meine Freunde direkt zu dem Event einladen können, damit sie sich auch die Details anschauen und Tickets buchen können.
- U29: Als User möchte ich über meine Telefonkontakte Zugriff auf meine Freunde bekommen, damit ich sehen kann zu welchen Events sie gehen.
- U30: Als User möchte ich von Events inspiriert werden, um auf neue Möglichkeiten aufmerksam gemacht zu werden und neues auszuprobieren
- U31: Als User möchte ich Events finden, die ich sonst nirgends finde.
- U32: Als User möchte ich, dass meine Daten sicher sind
- U33: Als User möchte ich, dass die Applikation geringe Ladezeiten/Latenzen hat, damit ich nicht unnötig Zeit verschwende.
- U34: Als User möchte ich keine langen Ladezeiten, um möglichst schnell nach Events swipen zu können.

Entwickler:

- E1: Als Entwickler möchte ich neue Eventquellen anschließen können, damit neue Events für die User einsehbar sind.
- E2: Als Entwickler möchte ich eine gute Branchstruktur, damit es IMMER möglich ist, Änderungen zu pushen und zu deployen.
- E3: Als Entwickler möchte ich eine performanten Code, damit die Applikation zu jeder Zeit (99,9 %) gut läuft.
- E4: Als Entwickler möchte ich Component Code, um Funktionen/ Bestandteile einfach auszutauschen bzw. erweitert werden.
- E5: Als Entwickler möchte ich Code Kommentare und Funktionsbenennungen auf Englisch haben, damit auch andere nicht-deutschsprachige Leute alles verstehen.
- E6: Als Entwickler möchte ich übersichtlichen Code (Struktur), um beispielsweise Fehler schneller zu identifizieren.

- E7: Als Entwickler möchte ich schnell neue Releases der App veröffentlichen, damit neue Funktionen schnell zum User gelangen/um schnell auf Feedback der user reagieren zu können.
- E8: Als Entwickler möchte ich, dass das Backend der Anwendung automatisch skaliert, damit ich mich nicht selbst um eine Erhöhung der Kapazität kümmern muss.
- E9: Als Entwickler der App möchte ich, dass die App tatsächlich nur Kosten verursacht, wenn auch Services der Provider in Anspruch genommen werden, damit ich nicht im Leerlauf Geld bezahle.

Investoren:

- I1: Als Investor möchte ich den ROI verstehen und wie ich helfen kann.
- I2: Als Investor möchte ich eine hohe Nutzeranzahl, um den Wert der App zu steigern
- I3: Als Investor möchte ich sicher gehen, dass Evender langfristig Umsatz generiert, um mit einem sicheren Gefühl investieren zu können
- I4: Als Investor möchte ich eine gute Skalierbarkeit (z.B. neue Features, andere Zielgruppe, Internationalisierung, etc.) sehen, um mehr Marktanteile zu gewinnen
- I5: Als Investor möchte ich ein starkes und motiviertes Team unterstützen, damit dies als sichere Basis besteht
- I6: Als Investor möchte ich einen "wasserdichten"Businessplan vorgelegt bekommen, damit ich die wichtigsten Fakten auf einen Blick erhalte
- I7: Als Investor möchte ich realistische und ambitionierte Meilensteine präsentiert bekommen, damit ich eine Zukunftaussicht analysieren kann

Veranstalter:

- V1: Als Veranstalter möchte ich, dass mein Event möglichst weit vorne platziert wird, damit es vielen Usern angezeigt wird.
- V2: Als Veranstalter möchte ich meine Events hervorheben können, um die Möglichkeit zu haben mehr User zu erreichen

- V3: Als Veranstalter möchte ich Events eintragen können, um mein Event einer größeren Reichweite zu präsentieren.
- V4: Als Veranstalter möchte ich von Evender Daten meiner Zielgruppe erhalten, um diese effizienter zu erreichen
- V5: Als Veranstalter möchte ich einen möglichst einfachen Onboarding Prozess, um mein Event einstellen zu können.
- V6: Als Veranstalter wünsche ich mir automatische Unterstützung, um mein Event möglichst effizient und Zielgruppen orientiert zu verbreiten.
- V7: Als Veranstalter möchte ich die wichtigsten KPIs in Echtzeit sehen.
- V8: Als Veranstalter möchte ich eine digitale Gästeliste haben.
- V9: Als Veranstalter möchte ich ein Follow Up Report mit den wichtigsten Daten (Bsp. Bewertungen etc.)
- V10: Als Veranstalter möchte ich auch nicht monetäre Veranstaltungen anbieten können, um Aufmerksamkeit zu generieren, z.B. für einen guten Zweck

Verfeinerte zur Entwicklung ausgesuchte User Storys

Funktionale Anforderungen

- UF1: Als User möchte ich auf einer Karte Veranstaltungen in meiner Nähe sehen, damit ich einen Überblick darüber bekomme, welche Veranstaltungen in meiner Nähe stattfinden.
- UF2: Als User möchte ich beim Swipen nach Ort, Radius und Zeitraum filtern können, damit ich keine Veranstaltungen sehe, die mich nicht interessieren.
- UF3: Als User möchte ich die Möglichkeit haben, mich für einen Newsletter anmelden zu können, damit ich über Veranstaltungen, die mich vielleicht interessieren, auf dem Laufenden bleibe.
- UF4: Als User möchte ich Veranstaltungen in meinen Telefon Kalender übertragen können, damit ich bei einem Blick in den Kalender direkt Bescheid weiß, wann eine Veranstaltung stattfindet.

- UF5: Als User möchte ich bereits geswipte Veranstaltungen wiederherstellen könne, damit ich auch bei einem Versehen eine möglicherweise interessante Veranstaltung wiederfinden kann.
- UF6: Als User möchte ich Veranstaltungen, die mich interessieren, in einer Favoritenliste speichern können, damit ich jederzeit auf die Veranstaltungen und die entsprechenden Informationen zugreifen kann.
- UF7: Als User möchte ich im Swipe Stack Vorschläge sehen, die für mich interessant sind, damit ich nicht zu viele Veranstaltungen sehe, die mich nicht interessieren.
- UF8: Als User möchte ich zusätzliche Informationen zu einer angezeigten Veranstaltung bekommen, damit ich einschätzen kann, ob mir die Veranstaltung gefällt.
- UF9: Als User möchte ich die Möglichkeit haben, mit Hilfe eines Links direkt zum Ticketkauf für eine Veranstaltung zu kommen, damit ich nicht online danach suchen muss.
- UF10: Als User möchte ich mich in der App anmelden können, damit ich Zugriff auf meine gespeicherten Events bekomme.
- UF11: Als User möchte ich mich mit bereits bestehenden Accounts (Google, Apple, Facebook) einloggen können, damit ich nicht extra einen Account innerhalb der App erstellen muss.
- UF12: Als User möchte ich sehen können, für welche Veranstaltungen sich meine sozialen Kontakte, die die App auch verwenden, sich interessieren damit ich mit ihnen zusammen zu einer Veranstaltung gehen kann.
- UF13: Als User möchte ich Freunden die Informationen zu einer Veranstaltung zukommen lassen können, um sie auf die Veranstaltung aufmerksam zu machen.
- UF14: Als User möchte ich nach dem Öffnen der App einen Stapel an Events sehen, durch die ich swipen kann, damit ich spielerisch neue interessante Veranstaltungen finden kann.

Nicht-funktionale Anforderungen

User:

- UNF1: Als User möchte ich, dass die Anwendung jederzeit erreichbar ist, damit ich sie auch immer benutzen kann.

- UNF2: Als User möchte ich, dass die App geringe Ladezeiten und Latenzen hat, damit die Verwendung der App angenehm ist.
- UNF3: Als User möchte ich möglichst viele verschiedene Veranstaltungen vorgeschlagen bekommen, damit ich auch wirklich neue Dinge entdecken kann.

Entwickler:

- ENF1: Als Entwickler möchte ich einfach neue Quellen für Veranstaltungen anschließen können, damit nicht bei jeder neuen Quelle zu viel Zeit mit der Implementierung verbracht wird.
- ENF2: Als Entwickler möchte ich eine gut durchdachte und modulare Architektur haben, damit bei Bedarf schnell und einfach jeder Teil des Codes angepasst werden kann.
- ENF3: Als Entwickler der App möchte ich, dass das Backend der Anwendung automatisch skaliert, damit ich mich nicht selbst um eine Erhöhung der Kapazität kümmern muss.
- ENF4: Als Entwickler der App möchte ich, dass die App tatsächlich nur Kosten verursacht, wenn auch Services der Provider in Anspruch genommen werden, damit ich nicht im Leerlauf Geld bezahle.
- ENF5: Als Entwickler der App möchte ich, dass die Ausführung von Programmcode so effizient wie möglich ist, damit die Kosten, die dadurch verursacht werden, so gering wie möglich bleiben.

User Story Cards

UF1: Karte für Veranstaltungen
Beschreibung: Als User möchte ich auf einer Karte Veranstaltungen in meiner Nähe sehen, damit ich einen Überblick darüber habe, welche Veranstaltungen an meinem aktuellen Aufenthaltsort stattfinden.
Akzeptanzkriterien:
Der User hat die Möglichkeit, in einem Tab der App auf ein Symbol zu tippen, damit die Karte erscheint.

In der Kartenansicht sind in einem Standardausschnitt um den aktuellen Aufenthaltsort alle Veranstaltungen zu sehen, die in der Datenbank existieren und an diesem Ort stattfinden.

UF2: Filtern
Beschreibung: Als User möchte ich beim Swipen nach Ort, Radius und Zeitraum filtern können, damit ich keine Veranstaltungen sehe, die mich nicht interessieren. Akzeptanzkriterien:
In der Standardansicht der Applikation mit dem Swipe Stack gibt es einen Button, mit dem man den Filter einstellen kann.
In der Filteransicht kann der user einstellen, ob und mit welchen Werten für die Parameter Radius, Zeitraum und Ort er filtern möchte.
Mit einem Button kann der User diesen Filter dann anwenden und der Swipe Stack wird neu geladen.

UF3: Newsletter
Beschreibung: Als User möchte ich die Möglichkeit haben, mich für einen Newsletter anmelden zu können, damit ich über Veranstaltungen, die mich vielleicht interessieren, auf dem Laufenden bleibe.
Akzeptanzkriterien:
Im Info Bereich der App gibt es einen Button, mit dem sich der User für den Newsletter anmelden kann.
In einem Textfeld kann er seine gewünschte E-Mail Adresse hinterlegen.

UF4: Kalender
Beschreibung: Als User möchte ich Veranstaltungen in meinen Telefon Kalender übertragen können, damit ich bei einem Blick in den Kalender direkt Bescheid weiß, wann eine Veranstaltung stattfindet.
Akzeptanzkriterien:
Wenn der User in der Favoritenansicht auf eine Veranstaltung drückt gibt es einen Button, mit dem sich die Veranstaltung in den Kalender übertragen lässt.
Nach Drücken dieses Buttons ist die Veranstaltung mit dem Namen und der Beschreibung am entsprechenden Datum und zur richtigen Uhrzeit im Kalender eingetragen.

UF5: Wiederherstellung
Beschreibung: Als User möchte ich bereits geswipte Veranstaltungen wiederherstellen

könne, damit ich auch bei einem Versehen eine möglicherweise interessante Veranstaltung wiederfinden kann.
Akzeptanzkriterien:
In der Haupt Ansicht der App gibt es einen Rewind Button, mit dem sich der letzte Swipe rückgängig machen lässt.
Nach drücken dieses Buttons ist die letzte Karte wiederhergestellt.

UF6: Favoriten
Beschreibung: Als User möchte ich Veranstaltungen, die mich interessieren, in einer Favoritenliste speichern können, damit ich jederzeit auf die Veranstaltungen und die entsprechenden Informationen zugreifen kann.
Akzeptanzkriterien:
Wenn der User in der Hauptansicht eine Veranstaltung nach rechts swiped, wird diese automatisch in seine Favoritenliste hinzugefügt.
Wenn der User in der Hauptansicht auf einen Button drückt, werden ihm alle Veranstaltungen, die er geliked hat, als Liste angezeigt.
Wenn der User in der Favoritenliste auf eine Veranstaltung drückt werden ihm alle Informationen zu dieser Veranstaltung angezeigt.

UF7: Recommender System
Beschreibung: Als User möchte ich im Swipe Stack Vorschläge sehen, die für mich interessant sind, damit ich nicht zu viele Veranstaltungen sehe, die mich nicht interessieren.
Akzeptanzkriterien:
Der User bekommt beim Swipen in der App Vorschläge entsprechend seines Verhaltens angezeigt, z.B. in Form eines „Für dich vorgeschlagen“ Stacks.

UF8: Zusätzliche Informationen
Beschreibung: Als User möchte ich zusätzliche Informationen zu einer angezeigten Veranstaltung bekommen, damit ich einschätzen kann, ob mir die Veranstaltung gefällt.
Akzeptanzkriterien:
Wenn der User in der Hauptansicht auf die Swipe Karte drückt werden ihm alle verfügbaren Informationen zu dieser Veranstaltung angezeigt: Ort, Zeit, Beschreibung, Ticket Link, Foto, Art der Veranstaltung.

UF9: Ticket Link
Beschreibung: Als User möchte ich die Möglichkeit haben, mit Hilfe eines Links direkt zum Ticketkauf für eine Veranstaltung zu kommen, damit ich nicht online danach suchen muss.
Akzeptanzkriterien:
Wenn der User sich in der Hauptansicht zusätrzliche Informationen zu einer Veranstaltung anzeigen lässt steht dort auch ein Link zur Verfügung, der direkt zur Seite des Ticketanbieters führt.
Wenn der user in der Favoritenliste auf eine Verasntaltung drückt ist in den Informationen zu dem Event der Ticket Link sichtbar.

UF10: Anmeldung
Beschreibung: Als User möchte ich mich in der App anmelden können, damit ich Zugriff auf meine gespeicherten Veranstaltungen bekomme.
Akzeptanzkriterien:
Wenn der User die App öffnet, erscheint ein Fenster zur Anmeldung.
Wenn das Fenster zur Anmeldung erscheint, hat der User verschiedene Möglichkeiten sich anzumelden.
Wenn der User keine Drittanbieter Anmeldung vornehmen möchte kann er sich mit seiner E-Mail Adresse und einem Passwort anmelden.
Wenn der User sich angemeldet hat, sind seine Einstellungen und seine Favoritenliste wieder verfügbar.

UF11: Drittanbieter Anmeldung
Beschreibung: Als User möchte ich mich in der App anmelden können, damit ich Zugriff auf meine gespeicherten Veranstaltungen und Einstellungen bekomme.
Akzeptanzkriterien:
Wenn der User das Fenster zur Anmeldung sieht, kann er sich auch mit seinem Google/Apple oder Facebook Konto anmelden.
Wenn der User einen der Drittanbieter auswählt, wird er zur Anmeldung auf die entsprechende Seite weitergeleitet.
Wenn der User sich angemeldet hat, sind seine Einstellungen und seine Favoritenliste wieder verfügbar.

UF12: Soziale Kontakte
Beschreibung: Als User möchte ich sehen können, für welche Veranstaltungen sich meine Sozialen Kontakte, die die App auch verwenden, interessieren, damit ich mich mit ihnen diesbezüglich absprechen kann.
Akzeptanzkriterien:
In der App gibt es einen Reiter für soziale Kontakte.
Wenn der user auf diesen Reiter drückt, kann er in mittels eines Textfeldes nach anderen Usern suchen.
Wenn der User auf einen gefundenen User drückt, kann er sehen, für welche Veranstaltungen sich dieser interessiert.

UF13: Teilen
Beschreibung: Als User möchte ich Freunden die Informationen zu einer Veranstaltung zukommen lassen können, um sie auf die Veranstaltung aufmerksam zu machen.
Akzeptanzkriterien:
Wenn der User in der Hauptansicht auf eine Veranstaltung drückt und sich die zusätzlichen Informationen anzeigen lässt, kann er über einen Button den Link zur Veranstaltung teilen.
Wenn der User in der Favoritenliste auf eine Veranstaltung drückt und sich die zusätzlichen Informationen anzeigen lässt, kann er über einen Button den Link zur Veranstaltung teilen.
Wenn der User auf den Link zum Teilen drückt bekommt er alle für ihn möglichen Arten des Teilens angezeigt (Mail, WhatsApp, Telegram etc.).

UF14: Swipen
Beschreibung: Als User möchte ich nach dem Öffnen der App einen Stapel an Events sehen, durch die ich swipen kann, damit ich spielerisch neue interessante Veranstaltungen finden kann.
Akzeptanzkriterien:
Wenn der User die App öffnet und sich angemeldet hat, bekommt er einen Stapel an Veranstaltungen zu sehen.
Wenn der User eine der Veranstaltungen nach rechts swiped wird diese Veranstaltung zu seiner Favoritenliste hinzugefügt.
Wenn der User eine der Veranstaltungen nach links swiped wird diese Veranstaltung ver-

worfen und wird ihm nicht wieder angezeigt.

UNF1: Erreichbarkeit
Beschreibung: Als User möchte ich, dass die Anwendung jederzeit erreichbar ist, damit ich sie auch immer benutzen kann.
Akzeptanzkriterien:
Wenn der Anwender die Applikation verwenden möchte, sollte sie 24 Stunden am Tag und 7 Tage die Woche erreichbar sein und funktionieren.

UNF2: Latenz
Beschreibung: Als User möchte ich, dass die App geringe Ladezeiten und Latenzen hat, damit die Verwendung der App angenehm ist.
Akzeptanzkriterien:
Wenn der User die App verwendet, sollte jede Funktionalität in einem angemessenen Zeitrahmen reagieren (genau Grenzwerte sind noch festzulegen).

UNF3: Diversität
Beschreibung: Als User möchte ich möglichst viele verschiedene Veranstaltungen vorgeschlagen bekommen, damit ich auch wirklich neue Dinge entdecken kann.
Akzeptanzkriterien:
Wenn der User die App verwendet, sollen möglichst viele verschiedene Veranstaltungen angezeigt werden, das heißt es müssen möglichst viele Quellsysteme angeschlossen werden.

ENF1: Anschluss neuer Quellsysteme
Beschreibung: Als Entwickler möchte ich einfach neue Quellen für Veranstaltungen anschließen können, damit nicht bei jeder neuen Quelle zu viel Zeit mit der Implementierung verbracht wird.
Akzeptanzkriterien:
Wenn ein Entwickler ein neues Quellsystem anschließen möchte, soll die grundlegende Funktionalität dafür schon existieren.
Wenn ein Entwickler ein neues Quellsystem anschließen möchte, sollen die nötigen Anpassungen dafür minimal sein.

ENF2: Architektur
Beschreibung: Als Entwickler möchte ich eine gut durchdachte und komponentenartige Architektur haben, damit bei Bedarf schnell und einfach jeder Teil des Codes angepasst werden kann.
Akzeptanzkriterien:
Wenn ein Entwickler Teile der Architektur austauschen oder verändern möchte, soll dies durch eine gut durchdachte und Modulartige Architektur ermöglicht werden. Kriterien dafür können die SOLID Kriterien sein.

ENF3: Skalierbarkeit
Beschreibung: Als Entwickler der App möchte ich, dass das Backend der Anwendung automatisch skaliert, damit ich mich nicht selbst um eine Erhöhung der Kapazität kümmern muss.
Akzeptanzkriterien:
Wenn sich die Last durch Verwendung der App erhöhen sollte, soll eine automatische Skalierung durch den Backend Provider stattfinden, sodass nicht manuell neue Server gestartet werden müssen.
Die Skalierung soll dabei alle Komponenten der Anwendung betreffen.

ENF4: Kosten
Beschreibung: Als Entwickler der App möchte ich, dass die App tatsächlich nur Kosten verursacht, wenn auch Services der Provider in Anspruch genommen werden, damit ich nicht im Leerlauf Geld bezahle.
Akzeptanzkriterien:
Wenn die App nicht verwendet wird, sollten auch keine Kosten entstehen.
Der Entwickler sollte nur das bezahlen müssen, was er auch an Rechenleistung in Anspruch genommen hat.

ENF5: Effizienz
Beschreibung: Als Entwickler der App möchte ich, dass die Ausführung von Programmcode so effizient wie möglich ist, damit die Kosten, die dadurch verursacht werden, so gering wie möglich bleiben.
Akzeptanzkriterien:
Wenn Code ausgeführt wird, soll die Laufzeit so gering wie möglich sein, um Kosten zu

sparen.

Anwendungsfälle

Anwendungsfall 1: Anmeldung
Nutzungskontext: Wenn der User die App öffnet, muss er sich anmelden, um Zugriff auf seinen Account zu bekommen.
Vorbedingung: Die App wurde geöffnet und ist betriebsbereit.
Erfolgreicher Endzustand: Der User ist in der App angemeldet und kann auf seine Daten zugreifen und die Funktionen der App nutzen.
Nicht Erfolgreicher Endzustand: Bei nicht erfolgreicher Anmeldung wird eine Fehlermeldung gezeigt, die den User auf den Fehler hinweist. Der User bleibt im Anmeldungsbildschirm und kann nicht auf die Funktionen und seine Daten zugreifen.
Akteure: User, App
Trigger: Öffnen der App.
Beschreibung:
1. Der User öffnet die App.
2. Die App zeigt dem User die Möglichkeiten zur Anmeldung an.
3. Der User tippt auf eine dieser Möglichkeiten.
4. Der User meldet sich an.
5. Nach erfolgreicher Anmeldung kommt der User in die Hauptansicht der App und kann alle Funktionen nutzen und seine Daten einsehen.
Erweiterungen:
3a Der User meldet sich mit seiner E-Mail-Adresse an.
3b Der User meldet sich mit seinem Facebook Account an.
3c Der User meldet sich mit seiner Apple ID an.
3d Der User meldet sich mit seinem Google Account an.

Anwendungsfall 2: Veranstaltungen favorisieren
Nutzungskontext: Wenn der User sich angemeldet hat, kann er in der Hauptansicht oder in der Kartenansicht neue Veranstaltungen favorisieren.
Vorbedingung: Die App wurde geöffnet, der User hat sich angemeldet und ist in der Hauptansicht.
Erfolgreicher Endzustand: Der User hat in der Hauptansicht oder in der Kartenansicht

Veranstaltungen favorisiert und diese Veranstaltungen sind dann in seinen Favoriten verfügbar.
Nicht Erfolgreicher Endzustand: Der User ist nicht in der Lage, Veranstaltungen zu swipen. Wenn kein swipen möglich ist, soll eine Fehlermeldung mit dem Grund angezeigt werden.
Akteure: User, App
Trigger: Öffnen der App und Anmeldung.
Beschreibung:
1. Der User sieht einen Stapel an Veranstaltungen in der Hauptansicht.
2. A.) Der User swiped eine der Veranstaltungen nach rechts.
B.) Der User klickt auf den Herz Button.
3. Die entsprechende Veranstaltung befindet sich nun in den Favoriten des Users.

Anwendungsfall 3: Favoriten anzeigen
Nutzungskontext: Wenn der User angemeldet ist, kann er sich die Veranstaltungen, die er bisher favorisiert hat, anzeigen lassen.
Vorbedingung: Die App wurde geöffnet und der User hat sich angemeldet.
Erfolgreicher Endzustand: Der User sieht alle Events, die er bisher favorisiert hat.
Nicht Erfolgreicher Endzustand: Der User sieht keine favorisierten Events und es soll eine Fehlermeldung erscheinen, warum dies der Fall ist.
Akteure: User, App
Trigger: Klicken auf den „Favoriten“ Button in der Buttonleiste am unteren Rand der App.
Beschreibung:
1. Der User klickt auf den „Favoriten“ Button am unteren Rand der App.
2. Der User gelangt auf die Favoritenansicht und sieht seine favorisierten Events in chronologischer Reihenfolge.

Anwendungsfall 4: Veranstaltungen filtern
Nutzungskontext: Wenn der User in der Hauptansicht Veranstaltungen swiped, kann er den angezeigten Stapel an Veranstaltungen nach bestimmten Parametern filtern.
Vorbedingung: Die App wurde geöffnet, der User hat sich angemeldet und er befindet sich in der Hauptansicht.
Erfolgreicher Endzustand: Der User hat einen Filter eingestellt und sieht nur noch dem Filter entsprechende Veranstaltungen.

Nicht Erfolgreicher Endzustand: Der User sieht keine dem Filter entsprechende Veranstaltungen oder er war nicht in der Lage, überhaupt einen Filter einzustellen. In diesem Fall sollte der User eine Fehlermeldung mit der entsprechenden Begründung erhalten.
Akteure: User, App
Trigger: Klicken auf den „Filter“ Button oben rechts in der Hauptansicht.
Beschreibung:
1. Der User befindet sich in der Hauptansicht.
2. Der User klickt auf den „Filter“ Button oben rechts.
3. Die Filteransicht öffnet sich.
4. Der User kann bei den Parametern aus „Ort“, „Datum“ und „Radius“ wählen und diese einstellen.
5. Der User klickt auf „Bestätigen“.
6. Der User befindet sich wieder in der Hauptansicht und sieht die gefilterten Veranstaltungen.

Anwendungsfall 5: Veranstaltungen teilen
Nutzungskontext: Wenn sich der User in der Detailansicht einer Veranstaltung befindet, kann er diese Veranstaltung über einen Button teilen.
Vorbedingung: Die App wurde geöffnet, der User hat sich angemeldet und er befindet sich in der Detailansicht für eine Veranstaltung.
Erfolgreicher Endzustand: Der User hat über ein Medium seiner Wahl den Link zu der Veranstaltung geteilt.
Nicht Erfolgreicher Endzustand: Bei nicht-erfolgreichem Teilen der Veranstaltung bekommt der User eine Fehlermeldung zu der Begründung angezeigt.
Akteure: User, App
Trigger: Klicken auf den „Teilen“ Button in der Detailansicht.
Beschreibung:
1. Der User klickt auf die Detailansicht einer Veranstaltung.
2. In der Detailansicht klickt der User auf den „Teilen“ Button.
3. Der User bekommt eine Reihe an Möglichkeiten zum Teilen vorgeschlagen (z.B. WhatsApp, Mail).
4. Der User klickt auf eine der Möglichkeiten.
5. Die ausgewählte App öffnet sich.
6. Der Link zur Veranstaltung wurde automatisch mit übernommen.

7. Der User schickt die Nachricht mit dem Link ab.

Emma, 28 Jahre alt, End User: *„Warum ist es so schwierig und aufwändig, Veranstaltungen zu finden, die mich interessieren könnten?“*

Charakteristika

Wohnort: Hamburg | *Geschlecht*: Weiblich | *Beziehungsstatus*: Single
Sprachen: Deutsch, Englisch, Spanisch

Favorisierte Websites & Apps: Instagram, Tinder, Youtube, Netflix, Vinted, TikTok
Persönlichkeit: Extrovertiert, Humorvoll, Reflektiert, Locker, Abenteuerlustig, Gewissenhaft
Fachwissen: Masterabschluss in BWL in der Fachrichtung Marketing, enormes Fachwissen in Bezug auf Design
Fähigkeiten: Kann gut mit anderen Menschen umgehen, hat ein Auge für Ästhetik, schafft es, andere Menschen zu motivieren

Ziele und Aufgaben

Hobbies und Interessen: Freunde treffen und ausgehen (vorranging in Bars, angesagten Restaurants und Clubs), Reisen, Sport, Workshops
Tätigkeiten am Arbeitsplatz: Übernimmt das Social Media Marketing für eine bekannte Firma, dadurch viele Kontakte mit anderen Menschen und hohes Bewusstsein für Design
Werte: Freiheit, Hilfsbereitschaft, Spontaneität, Umweltbewusst, möchte eine ausgeglichene Work-Life-Balance, setzt sich für Feminismus und ein besseres Bewusstsein für Mental Health ein
Lebensziele: Möchte nicht die Arbeit priorisieren, sondern den anderen Teilen des Lebens mehr Aufmerksamkeit schenken. Emma möchte ein aufregendes Sozialleben und enge Freunde immer um sich haben.

Motivation

Was bewegt Emma? Was motiviert sie? Was beeinflusst ihre Entscheidungen?
Emma strebt nach Selbstverwirklichung und das treibt sie an, ihr Job wird diesem Ziel untergeordnet, sie möchte einen positiven Einfluss auf die Welt nehmen und Spuren in dieser hinterlassen. Ihr Umweltbewusstsein beeinflusst ihre Entscheidungen. Sie möchte ständig neue Dinge entdecken und sich neu vernetzen. Erfolg bedeutet für sie nicht ausschließlich Erfolg im Beruf: sie definiert ihr Glück über ihre Beziehungen und die Dinge, die sie tut und die sie glücklich machen. Auf Grund ihrer sozialen Art liebt sie es, mit Freunden zu Veranstaltungen zu gehen, die ihre Werte widerspiegeln (z.B. Workshops oder Lesungen).

Anforderungen, Fragen und Bedürfnisse

Wie informiert sich Emma?
Viel über mobile Applikationen, bspw. mit Hilfe des Tagesschau Kanals auf Instagram, d.h. sie informiert sich vorrangig über Social Media. Sie besitzt aber auch dediziert mobile Nachrichten-Apps auf dem Handy installiert in die sie manchmal schaut..

Wie ist ihr Umgang mit IT-Systemen?
Sie ist mit IT-Systemen in ihrem Alltag aufgewachsen, sehr sicher in der Verwendung neuer Systeme und sie kann sich gut anpassen an neue Gegebenheiten.

Welche Erwartungen und Bedürfnisse hat sie an IT-Systeme?
Schickes intuitives User Interface & Layout, optisch ansprechend, einfaches Onboarding, Gamification, emotionaler Faktor

Welche Sorgen und Ängste beschäftigen sie?
Fear of missing out, ihr Bedürfnis nach einer nachhaltigen Welt rührt von einer Angst um die eigene Zukunft her,

Bedürfnisse und Pain Points in Bezug auf Veranstaltungen:
Die Zeit, die sie zum Suchen von passenden Events benötigt, würde sie lieber in andere Dinge stecken. Durch ihre hohe Social Media Affinität bemerkt sie, dass es in jeder App andere Veranstaltungen zu entdecken gibt. Sie möchte dahingehend gern informiert und auf dem Laufenden bleiben. Somit braucht sie ein System, das genau diese Punkte angeht: Einfaches und schnelles finden von passenden Veranstaltungen, die sie mit ihren Freunden teilen kann.

Abbildung .1: Persona

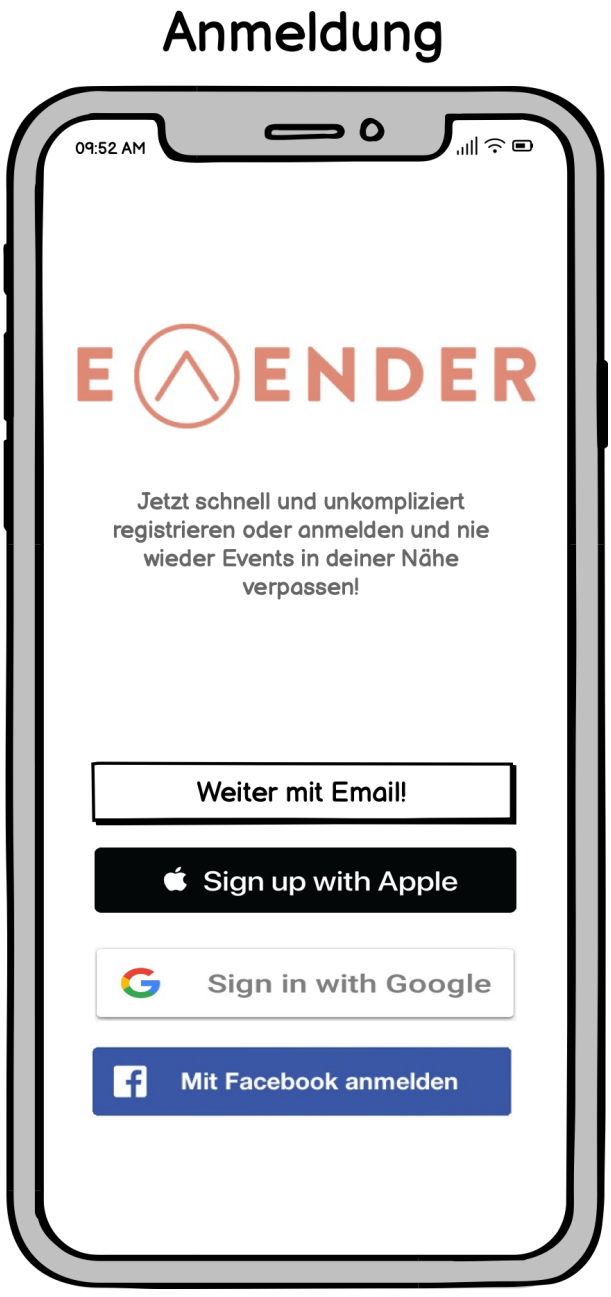


Abbildung .2: Anmeldungsbildschirm

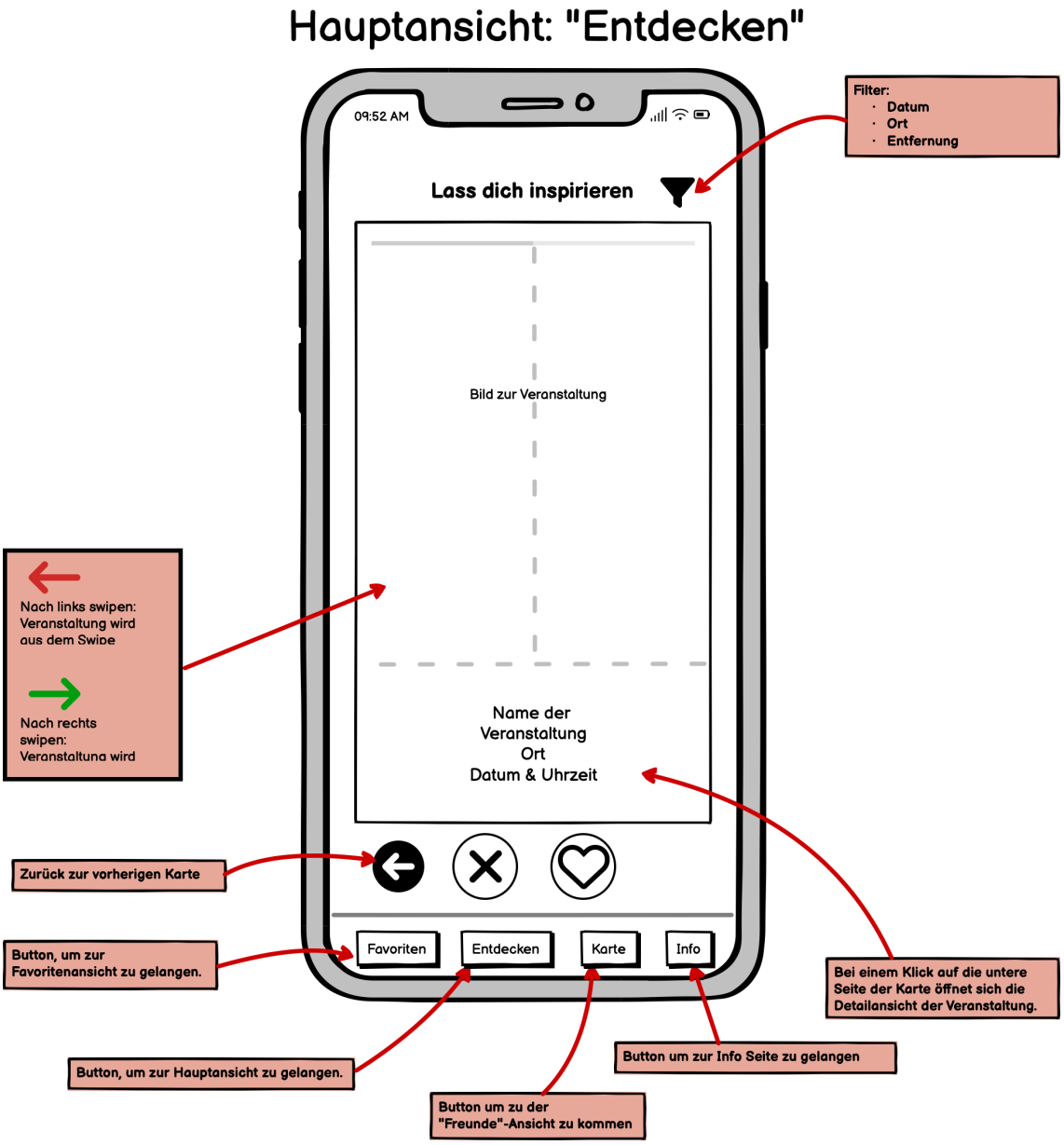


Abbildung .3: Hauptansicht

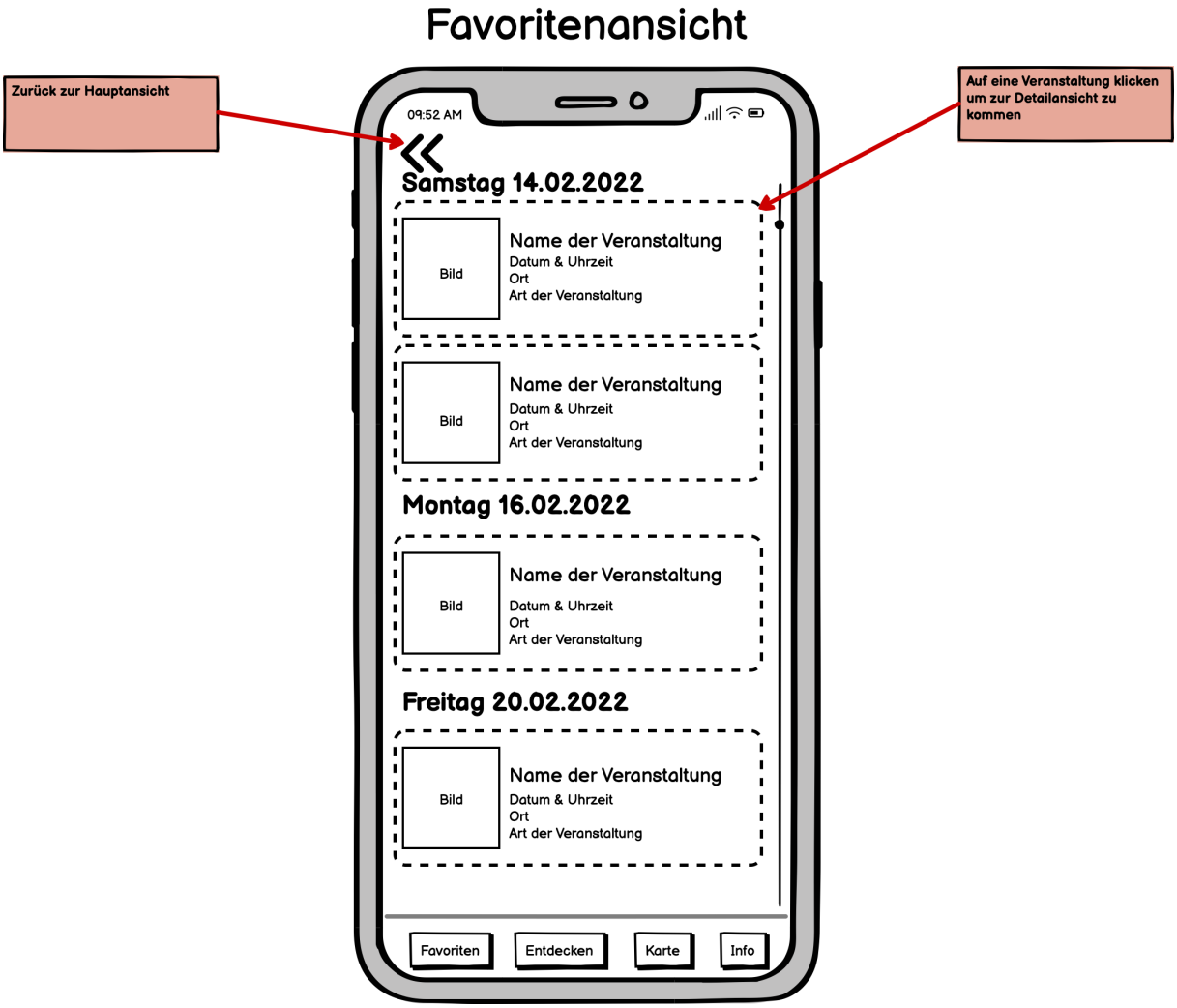


Abbildung .4: Favoritenansicht

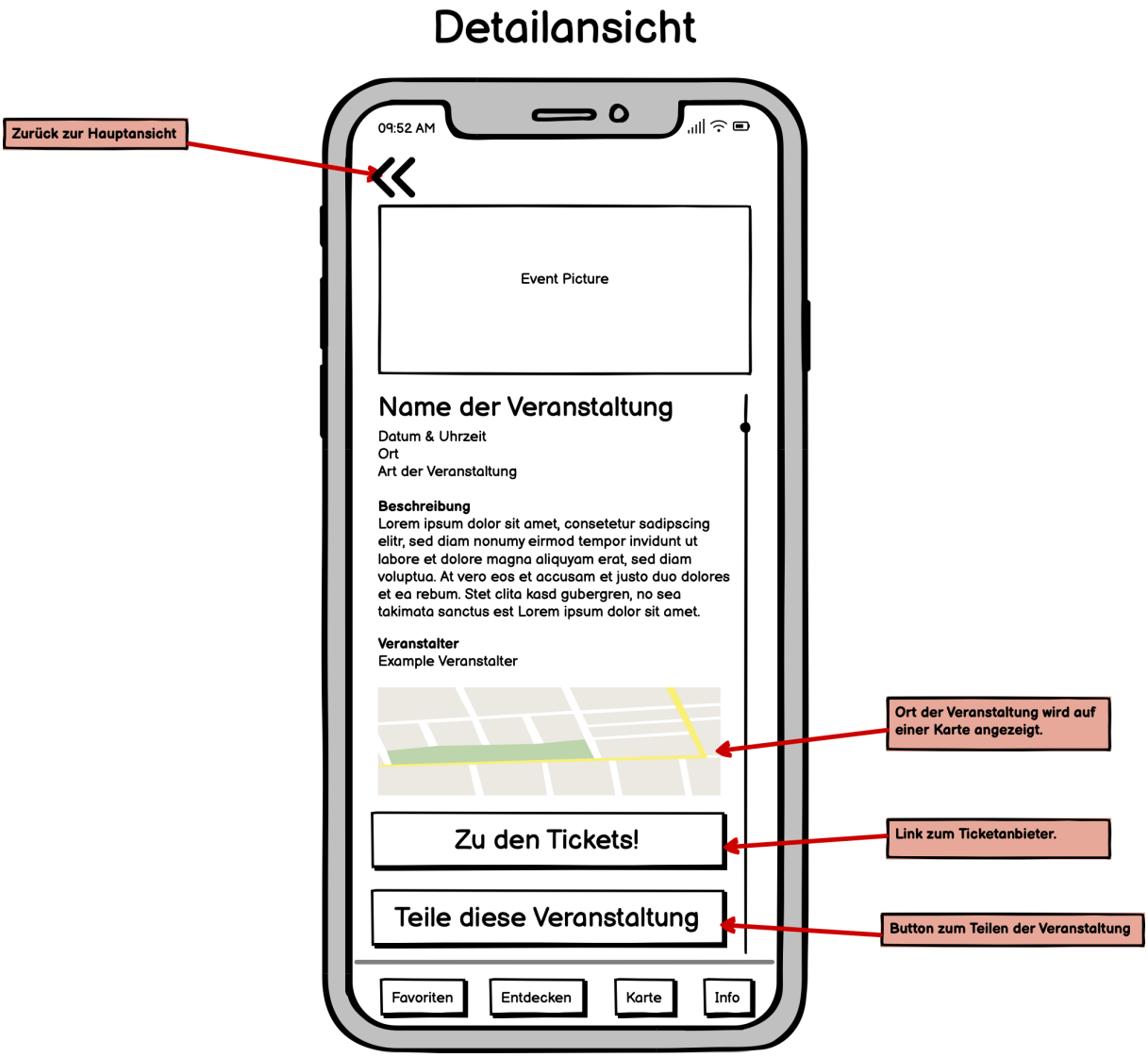


Abbildung .5: Detailansicht

Abbildung .6: Kartenansicht

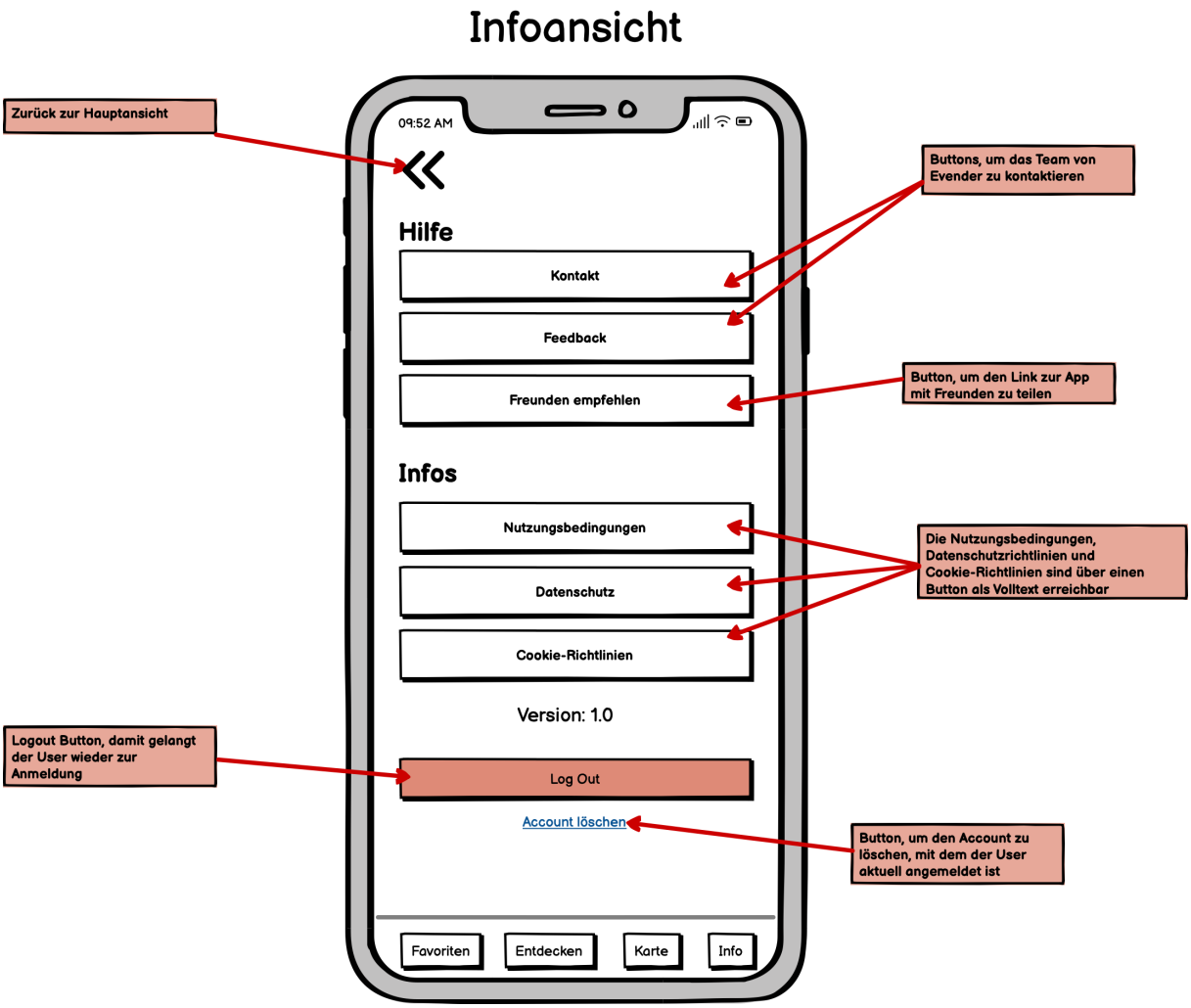


Abbildung .7: Infoansicht

Glossar

API Programmierschnittstelle.

Availability Zones Eine Region besteht aus mehreren Verfügbarkeitszonen, dient der Redundanz.

CLI Command Line Interface, Entwicklerschnittstelle, Bereich für die Eingabe von Text zur Steuerung von Software.

Cloud Formation Dienst von AWS, Infrastructure as Code Tool, erlaubt die Erstellung von Infrastruktur bei AWS mittels geschriebenen Codes.

Cloud Functions Leichtgewichtige Funktionen, die die Grundlage für Functions as a Service sind.

Container Virtualisierungstechnik, beinhaltet eine Anwendung und alle Abhängigkeiten zum Ausführen dieser Anwendung.

Debugging Der Vorgang des Findens und Behebens von Fehlern in einem Programm.

Framework Programmiergerüst, stellt den Rahmen zur Entwicklung von Software bereit.

GraphQL Abfragesprache für eine API.

Hosting Bereitstellung von Speicherplatz für Anwendungen und Systeme, meist über das Internet.

IAM Identity and Access Management, Dienst für das Managen von Identitäten.

Interface Schnittstelle.

Replikation Vervielfältigen von bspw. Daten, um eine höhere Ausfallsicherheit zu gewährleisten.

REST Representational State Transfer, Sammlung von Architekturbedingungen für APIs.

SDK Software Development Kit, Sammlung von Tools und Bibliotheken zum Erstellen von Software.

Sharding Methode der Datenbankpartitionierung, Aufteilung der Daten in mehrere Teile und speichern auf verschiedenen Instanzen.

Selbstständigkeitserkläung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original