Wirtschaft
Hauptcampus

HOCHSCHULE TRIER

Bachelor-Thesis

# Microservice-Modellierung und -Implementierung

# Darstellungsermöglichung von Userfeedback in Aflorithmics Voice Cloning App

Themensteller: Maximilian Meurer M. Sc.


**Name**: Julian Pretzsch
**Matrikelnummer**: 968896
**Fachbereich**: Wirtschaft
**Studiengang:** Wirtschaftsinformatik

**Abgabetermin:** 06.12.2021

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| AES | Audio Evaluation Service |
| API | Application Programming Interface |
| ARN | Amazon-Ressourcenname |
| AWS | Amazon Web Services |
| CLI | Command Line Interface |
| CSS | Cascading Style Sheets |
| DB | Database |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| JSON | JavaScript Object Notation |
| MS | Microservice |
| PK | Partition Key |
| REST | Representational State Transfer |
| SDK | Software Development Kit |
| SK | Sort Key |
| SQL | Structured Query Language |
| SQS | Simple Queue Service |
| S3 | Simple Storage Service |
| TTS | Text-To-Speech |
| VC | Voice Cloning |
| UTF-8 | 8-Bit UCS Transformation Format |
| XML | Extensible Markup Language |

# 1 Einleitung

„Alexa, wie ist das Wetter heute?“ „Okay Google, setze Bananen auf meine Einkaufsliste.“ Sprachassistenten sind nichts Neues mehr und längst in Gegenstände des Alltags integriert.[1] Juniper Research prognostizierte in einer Studie, dass bis 2024 8,4 Milliarden Sprachassistenten weltweit im Einsatz sein werden, was verglichen mit der Anzahl an verwendeten Sprachassistenten im Jahr 2019 (3.25 Milliarden) ein Wachstum von über 158 % bedeuten würde.[2] Währenddessen vergleicht Hörner den Einfluss digitaler Sprachassistenten auf die Digitalisierung und den Markt mit dem des Internetbrowsers oder Smartphones.[3] Doch worin liegt die Relevanz von Sprachassistenten?

Für User[4] liegt der Anreiz häufig an der großen Auswahl an Voice-Apps. Diese von Drittanbietern zur Verfügung gestellten Applikationen können einfach aktiviert werden und ermöglichen das, was der Assistent eigentlich soll, assistieren und den Alltag des Users bereichern. Je nach aktivierter App können sich User nach dem Wetter oder neuen Rezepten informieren.[5]

Der Anreiz für Unternehmen selbst im Markt der Voice-Apps tätig zu werden, besteht unter anderem darin, dass das Unternehmensimage durch häufigere, konsistente und jederzeit verfügbare Dialoge vermittelt werden kann, wodurch es leichter gelingt, Kunden langfristig zu binden. Hörner stellt diesbezüglich einen positiven Einfluss auf die direkte Kaufentscheidung der Kunden fest. [6]

Erweitert ein Unternehmen den Voice-Markt mit einem eigenen Produkt oder einer eigenen Marke, so ist festzustellen, dass sich der Begriff des Brand Images erweitern lässt. So wird das ursprüngliche, in der Zielgruppe existierende Bild der Marke um die nun vorhandene Stimme und Aussprache des Assistenten ergänzt, der dieses Image übermittelt. Das daraus resultierende Voice Branding befasst sich mit dem Positionieren der Marke im Voice-Markt, wobei eine Differenzierung von konkurrierenden

[1] Bedford-Strohm (2017), S. 486
[2] Juniper Research, zitiert nach de.statista.com (2020)
[3] Vgl. Hörner (2019), S. 1-35
[4] Aus Gründen der besseren Lesbarkeit wird in dieser Arbeit die Sprachform des generischen Maskulinums angewendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint.
[5] Vgl. ebd. S. 15f.
[6] Vgl. ebd. S. 229

Voice-Apps angestrebt wird. Auch hier existiert ein direkter Einfluss auf die Kaufentscheidung der Kunden.[7]

Hörner stellt das Verwenden einer eigenen Stimme, die repräsentativ für das eigene Produkt steht, zwar als erheblichen Fortschritt in Bezug auf die sich daraus ergebenden Möglichkeiten dar, formuliert den Einsatz solcher Stimmen allerdings als technologisch noch nicht umsetzbar.[8]

Mittlerweile bieten jedoch sowohl Microsoft[9] als auch AWS[10] einen Service an, der das Erstellen einer eigenen Stimme ermöglicht sowie die anschließende Integration in das eigene Produkt unterstützt. Dabei werden neuronale Text-To-Speech (TTS) Modelle, also computerbasierte Systeme, mit Audioaufnahmen eines Users trainiert, um anschließend hochqualitative, menschlich klingende Audioaufnahmen zu synthetisieren.[11] Die für das Training benötigten Audioaufnahmen der User werden über Microsofts Speech Studio[12] oder Amazons AWS Console[13] akquiriert und sind mithilfe selbiger Plattformen in jegliche Art von Apps oder Webseiten integrierbar.

Aflorithmic möchte im Gegensatz zu den Projekten Amazons oder Microsofts eine kostengünstige Alternative anbieten. Microsoft beispielsweise empfiehlt das Aufnehmen der Audiodateien in einem Tonstudio mit professionellem Equipment und einem Aufnahmeleiter.[14] Amazons Aufnahmekriterien werden zwar nicht konkret benannt, jedoch ist es erforderlich, Amazon selbst zu kontaktieren, um zusammen mit dem Amazon-Polly-Team die eigene Stimme zu gestalten.[15] Eine erforderliche hohe Aufnahmequalität wird hier dementsprechend angenommen.

Um seitens Aflorithmics eigene Audioaufnahmen zu akquirieren, arbeitete ich im Rahmen meines Praxissemesters an der Entwicklung einer Voice Cloning App[16]. Diese ermöglicht es, die zum Training benötigten Aufnahmen direkt über das Handy einzu-

[7] Vgl. ebd. S 226f.
[8] Vgl. Hörner (2019), S. 231-233
[9] Siehe dazu Microsoft Azure Documentation (2021a)
[10] Siehe dazu AWS (2020a)
[11] Vgl. Chen et al. (2021), S. 1, Vgl. Tan et al. (2021) S. 30f., Dutoit (1997), S. 1
[12] Vgl. Microsoft Azure Documentation (2021b)
[13] Vgl. AWS (2020b)
[14] Vgl. Microsoft Azure Documentation (2020)
[15] Vgl. AWS (2020c)
[16] Siehe dazu Aflorithmic (o. D. -a)

sprechen, wodurch seitens der User Aufwands- und Mietkosten aufgrund professioneller Aufnahmen wegfallen. Eine solche Kostenersparnis geht allerdings auch mit einem Qualitätsverlust der Aufnahmen einher. Statt in einem vor Hall gedämmten Tonstudio mit Kondensatormikrofon und qualitativem Feedback des Aufnahmeleiters können die Audioaufnahmen im hellhörigen Eigenheim aufgenommen werden. Die daraus resultierenden Qualitätsunterschiede würden sich schlussendlich auch in den TTS Modellen abbilden, da qualitativ hochwertigere Trainingsdaten mit hochwertigeren TTS Modellen korrelieren.[17] Demnach gilt es zu vermeiden, dass User viel Zeit und Mühe in das Aufnehmen der Daten investieren, wenn diese aufgrund einer zu geringen Qualität nicht verwendbar sind. Frustration, da ein TTS Modell mit kaum brauchbaren Aufnahmen gefüttert wurde und die synthetisierten Audioaufnahmen somit eine unbrauchbare Qualität mit sich bringen, könnte die Folge sein.

Um dem entgegenzuwirken, ist das Ziel dieser Arbeit die Gestaltung und Implementierung eines Microservices zur Darstellungsermöglichung von Userfeedback in Aflorithmics Voice Cloning App mittels eines Audio Feedback Systems.

Dabei wird in Kapitel 2 zunächst auf einige Grundlagen wie das HTTP (2.1) und einige ausgewählte Architekturansätze (2.2) eingegangen sowie die verwendeten Services des Cloud-Computing-Anbieters AWS erläutert (2.3). Im dritten Kapitel wird anschließend das Unternehmensmodell Aflorithmics dargestellt und diesbezüglich die Voice Cloning App eingeordnet. Schließlich werden im 4. Kapitel, nach einer kurzen Vorstellung der Voice Cloning App in 4.1, die drei Kern-Funktionalitäten der App

- (4.2) Registrierung und Login
- (4.3) Darstellung der vorzulesenden Sätze
- (4.4) Hochladen und Verwaltung der Audiodateien

erklärt. Kapitel 5 beginnt mit der Vorstellung des bereits existierenden Audio-Evaluierungs-Services (5.1) und den Anforderungen an den Microservice (5.2), den es anschließend zu modellieren (5.3) und zu implementieren (5.4) gilt. Abschließend wird die Arbeit im Rahmen einer Schlussbetrachtung im 6. Kapitel zusammengefasst und ein Ausblick auf die Zukunft des Projekts getätigt.

[17] Vgl. Chen et al. (2021), S. 1, Vgl. Tan et al. (2021), S. 22

# 2 Grundlagen

Innerhalb der Grundlagen werden zunächst die von Aflorithmic verfolgten Entwicklungsansätze erläutert. Dabei wird in einem ersten Schritt das Internet Protokoll HTTP in seinen grundlegenden Eigenschaften dargestellt. Anschließend folgt die Erklärung einiger Architekturansätze, welche innerhalb Aflorithmics ihre tägliche Anwendung finden. Das Kapitel schließt mit der Abgrenzung von Services des Cloud-Computing-Anbieters AWS, die zwecks Modellierung und Implementierung des Projektes verwendet werden sollen, ab.

## 2.1 HTTP

Das Hyper Transfer Protokoll wird seit 1990 für die Kommunikation zwischen Webclients und Webservern verwendet. HTTP ist dabei als Request-Response Protokoll zu verstehen. Der Client sendet eine Request, die unter anderem aus einer Request-Methode, einer URL, einem Header sowie einer Nachricht besteht.[18] Die gewählte Request-Methode definiert dabei, welche Operation der Client gerne auf die, per URL eindeutig identifizierbare, Ressource[19] anwenden möchte.[20] Der Header beinhaltet einige Metainformationen, wie beispielsweise das erwartete Format der Response.[21] Der Body hingegen kann als optionaler Parameter vom Client mitgeschickte Informationen enthalten und ist je nach Art der HTTP-Request relevant.[22] Eine weitere Option ist das Mitschicken von Informationen über die URL. Diese URL-Parameter sind Key-Value-Paare, die nach einem Fragezeichen an die URL angehängt werden können und mit einem Und-Zeichen voneinander getrennt werden.[23]

Die für dieses Projekt relevanten HTTP Methoden beschränken sich auf:

1. GET – der Client erhält die Informationen einer Ressource,
2. POST – der Client erstellt eine neue Ressource,
3. PUT – der Client aktualisiert eine Ressource.[24]

[18] Vgl. Fielding et al. (1999), S. 31-39
[19] Vgl. ebd. S. 18f.
[20] Vgl. ebd. S. 36
[21] Vgl. ebd. S. 39
[22] Vgl. ebd. S. 32f.
[23] Vgl. Google Support (o. D.)
[24] Vgl. Fielding et al. (1999), S. 36, Codecademy (o. D.)

Mit der aktuell verwendeten Version von HTTP/2 könnte eine beispielhafte Request an die Hochschulwebseite folgendermaßen aussehen.

GET https://www.hochschule-trier.de/ HTTP/2 [25]

Die Antwort des Hochschulservers enthält unter anderem einen Statuscode**,** der verdeutlicht, wie der Server die Request des Clients interpretiert und verarbeitet hat, einen Response-Header mit einigen Informationen über den Server und die angefragte Ressource sowie einen Body, der alle Daten zum Darstellen der Hochschulwebseite im Browser beinhaltet.[26] Der Body der Response entspricht dabei dem Format eines HTML-Dokuments, um das Darstellen der Website im Browser zu ermöglichen. Weitere Formate wie JSON oder XML sind je nach Art der Ressource und dem erwarteten Format, welches im Header angegeben wird, üblich.

## 2.2 Architekturansätze

### 2.2.1 API

APIs dienen der Wiederverwendbarkeit von Programm Code, ohne den unternehmensinternen Quellcode zu veröffentlichen. Externe Entwickler können lediglich den eigenen Code um die Funktionalität des über die API zur Verfügung gestellten Codes erweitern.[27] Der bestehende Datenaustausch zwischen Applikation des Entwicklers und dem Server, auf dem der Quellcode der API ausgeführt wird, bildet dabei zwei Kommunikationspartner ab.[28] Ein anfragender Client, der die Applikation ausführt und ein antwortender Server.

Vereinfacht wird eine API häufig mit einem Kellner in einem Restaurant verglichen. Der Gast (Client), der eine Speisekarte vor sich liegen hat, darf aus dieser ein Gericht wählen. Diese Speisekarte wird vom Restaurant vorgegeben und ist als Dokumentation zu verstehen, in welcher die möglichen Operationen mit der API aufgelistet und beschrieben werden.[29] Hat sich der Gast für ein Gericht entschieden (also für eine in der

[25] In enger Anlehnung an Fielding et al. (1999), S. 36f.
[26] Vgl. ebd. S. 39-42
[27] Vgl. Monperrus et al. (2012), S. 703f., Abstract, Supreme court of the United States (2020), S. 1-3, Wang et al. (2014), S. 246
[28] Vgl. Postman Blog (2020)
[29] Vgl. Robillard/Chhetri (2014), Abstract

Dokumentation stehende Interaktion mit der API), bestellt er das Gericht bei dem Kellner (Request). Dieser läuft zur Küche und bittet den Koch oder die Köchin, also die interne Software, die Bestellung zu bearbeiten. Dabei gelangt der Gast an keinerlei Informationen über das von dem Koch oder der Köchin verwendete Rezept (Quellcode). Ist das Gericht fertig, wird es von dem Kellner abgeholt und zum Gast gebracht (Response).[30]

### 2.2.2 REST

REST ist ein im Jahr 2000 entwickelter Architekturstil, der die Kommunikation und Zuständigkeitsbereiche im Client-Server-Modell voneinander neu absteckt und somit als Erweiterung der API-Architektur anzusehen ist. Dabei ist der Server für die Administration der Daten zuständig, der Client wiederum für das Userinterface, wie z. B. eine Webseite oder eine App auf dem Smartphone. Durch diese Struktur kann parallel an der clientseitigen (Frontend) sowie an der serverseitigen Infrastruktur (Backend) gearbeitet und entwickelt werden, da keine direkte Abhängigkeit zwischen Frontend und Backend besteht.[31] Im Falle eines clientseitigen Bugs[32] könnte zwar die Interaktion zwischen User und App beeinträchtigt oder sogar ganz verhindert sein, die Kommunikation zwischen dem Server und anderen, bugfreien Apps ist davon jedoch ungestört.

REST wurde zudem parallel zu HTTP/1.1 entwickelt und verwendet, auch aufgrund der ubiquitären Verfügbarkeit[33], meist HTTP zur Kommunikation zwischen Client und Server.[34] Somit konfiguriert der Client auch hier eine Request, bestehend aus einer HTTP-Methode, einer URL, einem Header und je nach Request einer Nachricht (Body).[35]

### 2.2.3 Microservice Architektur

Lewis und Fowler beschreiben die Microservice Architektur als ein Konzept zum Entwerfen von serverseitigen Applikationen. Die Idee besteht darin, den serverseitigen Teil einer Gesamt-Applikation in einzelne kleine Anwendungen zu unterteilen, die

[30] Vgl. Chaudhry (2017), Desai (2021)
[31] Vgl. Fielding (2000), S. 78
[32] Programmfehler
[33] Vgl. Drilling/Augsten
[34] Vgl. Fielding (2000), S. 116f.
[35] Vgl. Ekblom (2011), Abstract ff.

mithilfe der REST-API untereinander kommunizieren können.[36] Dabei erledigt jede einzelne Anwendung eine eigene - von anderen Anwendungen unabhängige - Aufgabe.

Im Kontrast dazu steht der herkömmliche Monolith-Architekturansatz, der ebenso aus den drei Bestandteilen Datenbank, Userinterface und Serverapplikation besteht. In diesem kann die serverseitige Anwendung als ein in sich geschlossenes Programm - welches nicht unterteilbar ist - angesehen werden. Dadurch bringt dieser Ansatz eine größere und komplexere Anwendung, die für Entwickler nur schwer zu pflegen und weiterentwickeln ist, mit sich. Des Weiteren muss bei kleinsten Veränderungen im Code die gesamte serverseitige Applikation neu deployt[37] werden. Wird die serverseitige Anwendung jedoch in einzelne Microservices unterteilt, ist bei richtiger Konzeption jeder einzelne Service einfach zu verstehen, zu entwickeln, zu testen und zu verwalten.

Ein weiterer Vorteil der Microservice-Architektur ist eine hohe Skalierbarkeit der einzelnen Services. Wird ein Microservice häufig benötigt, so genügt es, mehrere Instanzen desselben Services aufzurufen. Der Monolith-Architekturansatz bietet im Gegensatz dazu lediglich die Möglichkeit, die gesamte serverseitige Applikation als eine neue Instanz zu erstellen.[38]

## 2.3 Erklärung grundlegender AWS Services

Im Folgenden werden für das Projekt relevante Services der Cloud-Computing-Plattform **AWS** erläutert, die Aflorithmic zum Entwickeln und Bereitstellen eigener Produkte verwendet. Dabei existiert die Möglichkeit sowohl mit der AWS CLI als auch mit einer Auswahl an SDKs zu interagieren. Aufgrund der standardisierten Verwendung der Python SDK Boto3 in Aflorithmics Infrastruktur beziehen sich die folgenden Erklärungen und Beispiele auf ebendieses SDK.[39]

[36] Vgl. De Alwis et al. (2019), S. 496f.
[37] Prozess des Softwareupdates auf einem Server
[38] Vgl. Lewis/Fowler (2014)
[39] Siehe dazu Boto3 Documentation (o. D. -a)

### 2.3.1 Lambdas

Lambdas gehören zu den ausführenden Einheiten in der AWS Umgebung.[40] Ein Lambda kann dabei von anderen AWS-Services direkt oder indirekt mittels eines Ereignisses aufgerufen werden. Das Ereignis selbst ist ein JSON-Dokument, welches vom Lambda zu bearbeitende Informationen beinhaltet. Wird ein Lambda mithilfe eines solchen Ereignisses aufgerufen, so wird der in einer Lambdafunktion definierte Code ausgeführt und die im Ereignis übermittelten Informationen können verarbeitet werden.[41] Eine große Besonderheit bietet dabei das automatisierte Erstellen von Events mittels anderer AWS-Services.[42] So lässt sich ein Lambda beispielsweise so konzipieren, dass es aufgerufen wird, sobald eine Datei in eine gewisse AWS-Datenbank hochgeladen wird.[43] Im Anschluss daran kann im Lambda auf das Ereignis zugegriffen werden, in welchem sich Informationen über die hochgeladene Datei befindet.[44] In diesem Sinne ist das Lambda als eine Reaktion auf ein Ereignis zu verstehen, welches weiterverarbeitet werden soll.

Des Weiteren kann ein Lambda auch mittels AWS-Service-fernen Ereignissen aufgerufen werden, wie beispielsweise durch das Drücken eines Knopfes auf einer Webseite. Für diesen Zweck kann ein Endpoint erstellt werden, über welchen das Lambda eine Request erwartet.[45] Ein solcher Endpoint wird durch eine URL repräsentiert und erlaubt das Aufrufen eines Lambdas über eine HTTP-Request.[46] Lambdas sind dementsprechend ein sehr geeignetes Werkzeug zum serverseitigen Verarbeiten von Informationen.[47]

### 2.3.2 DynamoDB

DynamoDB ist ein NoSQL-Datenbank-Service, der sich von herkömmlichen SQL-Datenbanken erheblich unterscheidet.[48] Eine DynamoDB-Tabelle besteht aus einem

[40] Vgl. AWS Documentation (o. D. -a), S. 1
[41] Vgl. ebd. S. 18
[42] Vgl. ebd. S. 420
[43] Siehe dazu ebd. S. 476
[44] Vgl. ebd. S. 476f.
[45] Vgl. ebd. S 123-124, Vgl. Mathew/Varia (2014), S. 10f.
[46] Vgl. ebd. S. 426-432
[47] Vgl. ebd. S. 1
[48] Vgl. AWS Documentation (2012), S. 24ff

oder mehreren Elementen, welche wiederum durch ein oder mehrere Attribute beschrieben werden. Die grundlegende Struktur entspricht somit der einer SQL-Datenbank.[49]

Um innerhalb einer SQL-Datenbank eine Tabelle anzulegen, müssen die einzelnen Attribute der zukünftigen Tupel definiert werden. Ein solches Schema gibt also eine Grundstruktur der Elemente vor, die daraufhin der Tabelle hinzugefügt werden können. Im Gegensatz dazu ist eine NoSQL-Datenbank schemalos. Die einzelnen Elemente werden im Vergleich zu SQL-Datenbanken zwar ebenfalls mittels eines Primärschlüssels identifiziert, jedoch müssen weitere Attribute nicht definiert werden.[50] So können die Elemente einer Tabelle unterschiedliche und unterschiedlich viele Attribute besitzen.

**(1) Datentypen**

Attribute können verschiedene Datentypen annehmen. Seitens DynamoDB werden neben Strings und Integer auch Listen, Sets und Maps unterstützt. Letztere ermöglichen das Speichern von langen XML- oder JSON-Dokumenten innerhalb eines Attributs.[51] Es ist allerdings anzumerken, dass für den Primärschlüssel lediglich Integer, Strings und Booleans unterstützt werden.[52]

**(2) Primärschlüssel**

Der Primärschlüssel einer DynamoDB Tabelle identifiziert jedes Element innerhalb der Tabelle eindeutig. Im Gegensatz zu herkömmlichen SQL-Datenbanken kann der Primärschlüssel einer DynamoDB-Tabelle aus mehreren Attributen bestehen. Sowohl die Identifikation eines Elements mittels eines Partitionsschlüssels als auch die durch einen Composite Primary Key sind zulässig. Ein Composite Primary Key wird dabei aus einem Partitionsschlüssel und einem Sortierschlüssel zusammengesetzt und identifiziert die Elemente anhand der Kombination der beiden Schlüssel. Besteht der Primärschlüssel einzig aus einem Partitionsschlüssel, müssen die Elemente einer Tabelle, ähnlich den SQL-Datenbanken, nur anhand von diesem eindeutig identifizierbar sein. Hingegen können Tabellen die einen Composite Primary Key verwenden, Elemente

[49] Vgl. ebd. S. 3
[50] Vgl. AWS Documentation (2012), S.27-29 & S. 6
[51] Vgl. ebd. S. 13-16
[52] Vgl. ebd. S. 6

mit gleichem Partitionsschlüssel vorweisen, wenn sich diese in den Sortierschlüsseln unterscheiden. Elemente mit gleichem Partitionsschlüssel werden dabei zusammen gespeichert und anhand des Sortierschlüssels sortiert, wobei Integer absteigend und Strings nach ihren Werten in der UTF-8 Tabelle sortiert werden.[53] Das boolesche *True* hingegen wird höher eingestuft als das boolesche *False*.

**(3) Methoden**

Zum Interagieren mit den einzelnen Tabellen stellt die Boto3-Bibliothek eine Vielzahl an Methoden zur Verfügung.[54] Die für dieses Projekt verwendeten Methoden beschränken sich auf die Query- und die PutItem-Methode. Um mittels der Query-Methode einzelne oder mehrere Elemente zu suchen, muss mindestens der Name des Partitionsschlüssels sowie ein einzelner Wert für diesen angegeben werden. Optional lässt sich auch ein Sortierschlüssel angeben, um gefundene Elemente nach diesem zu sortieren. Folgende Vergleichsoperatoren sind dabei für einen Sortierschlüssel möglich:

- Der Sortierschlüssel ist = der Angabe
- Der Sortierschlüssel ist >, <, <=, >= der Angabe
- Der Sortierschlüssel ist *Between* der Angabe
- Der Sortierschlüssel *Begins with* der Angabe

Des Weiteren kann die Query-Methode um die Abfrage einzelner Attribute erweitert werden. Die Vergleichsoperatoren beschränken sich auf =, <, <=, >, >=, *Between, Begins with, Between And*[55]

Die PutItem-Methode wird verwendet, wenn einzelne Elemente zu einer Tabelle hinzuzufügen werden sollen. Wie zuvor erläutert muss dafür lediglich der Primärschlüssel des Elements angegeben werden. Falls der Primärschlüssel eines neuen Elements bereits existiert, wird das alte Element ersetzt.[56]

### 2.3.3 S3

Der Simple Storage Service (s3) ist der Objektspeicher in der AWS-Umgebung und ermöglicht das Speichern und Abrufen von Dateien unabhängig der Größe oder

[53] Vgl. AWS Documentation (2012), S. 6, Bouré (2021)
[54] Siehe dazu Boto3 Documentation (o. D. -b)
[55] Vgl. AWS Documentation (o. D. -b)
[56] Vgl. Boto3 documentation (o. D. -b)

Menge.[57] Die Objekte befinden sich dabei in einem sogenannten Bucket, in welchem ein Schlüssel für die eindeutige Identifikation des Objekts zuständig ist.[58] Dieser Schlüssel beinhaltet den Namen des Objekts, der beim Hochladen angegeben werden muss. Zum eindeutigen Identifizieren eines Objekts außerhalb eines Buckets, wird eine Objekt-URL verwendet, bestehend aus dem Namen des Buckets und dem Schlüssel des Objekts.[59]

Standardmäßig sind Objekte innerhalb eines Buckets privat, wodurch ein direkter Zugriff per Objekt-URL untersagt wird. Sollen private Objekte aufrufbar oder herunterladbar sein, eignet sich die Verwendung von presigned-URLs. Diese können vom Besitzer des Objekts erstellt werden und erlauben einen temporären Zugriff auf eine bestimmte Datei innerhalb eines bestimmten Buckets.[60]

Öffentliche (public) Objekte können hingegen direkt über die Objekt-URL aufgerufen werden. Somit sind öffentliche Objekte innerhalb eines s3-Buckets besonders für Webseiten oder sonstige öffentliche Applikationen geeignet.[61]

### 2.3.4 SQS

Amazons Simple Queue Service (SQS) bietet eine Warteschlange, die in Softwaresysteme bzw. Microservices zu Kommunikationszwecken integriert werden kann. Eine SQS-Queue wird ähnlich den s3-Buckets mittels einer URL eindeutig identifiziert. Diese SQS-Queue-URL wird beim Erstellen der Ressource automatisch generiert. Folglich wird eine solche SQS-Queue-URL verwendet, um Nachrichten in die Queue zu schicken, sie von der Queue zu erhalten oder sie zu löschen. Die Boto3 Bibliothek beinhaltet auch hier wieder eine Vielzahl an Methoden, die das Interagieren mit Amazons SQS ermöglichen. Wird eine Nachricht mittels *der send_message*-Funktion zu einer SQS-Queue hinzugefügt, befindet sie der Inhalt im *MessageBody* der Nachricht. Dieser kann jedoch lediglich unformatierte Texte oder Inhalte in JSON bzw. XML enthalten.[62]

[57] Vgl. AWS Documentation (2006), S. 1
[58] Vgl. ebd. S. 3
[59] Vgl. ebd. S. 4f.
[60] Vgl. AWS Documentation (2006), S. 236f.
[61] Vgl. ebd. S. 1045
[62] S 49

### 2.3.5 CloudFormation

CloudFormation ist ein Service, der das Erstellen von AWS-Ressourcen vereinfacht.[63] Innerhalb einer Vorlage können mehrere AWS-Ressourcen samt ihrer Eigenschaften definiert werden.[64] Eine Vorlage befindet sich dabei immer im YAML- oder JSON-Format[65] und muss mindestens eine AWS-Ressource beinhalten.[66] Nachdem die benötigten Ressourcen innerhalb der Vorlage definiert wurden, kann die YAML- oder JSON-Datei in den CloudFormation-Service hochgeladen werden. Die in der Vorlage definierten AWS-Ressourcen werden anschließend vom Service selbst erstellt, bereitgestellt, in einem Stack[67] zusammengefasst und für die Zusammenarbeit der Services konfiguriert. Stacks erleichtern dabei die Verwaltung der Ressourcen. Wird ein Stack gelöscht, so werden auch alle AWS-Ressourcen samt Inhalt gelöscht, die sich in diesem befinden.[68]

# 3 Vorstellung des Unternehmensmodells Aflorithmics

Aflorithmic ist ein 2019 gegründetes Tech-Startup, dass die Produkte ihrer Kunden um qualitativ hochwertige und auf den Endkunden zugeschnittene, synthetisierte Audioinhalte erweitern möchte. Dafür wird seit Beginn 2021 sowohl an einer API[69] als auch an einer Dokumentation[70] gearbeitet, um seitens der Kunden die Integration in das eigene Produkt zu vereinfachen.

Dabei werden vier Grundoperationen unterschieden:

1. Das Einspeisen jeglichen Textes in die API-Umgebung (Content),
2. das anschließende Überführen dieses Textes in Audio mittels Sprachsynthese (Speech),
3. das Hinzufügen von Hintergrundmusik und Effekten (Sound Design),
4. das Abstimmen des synthetisierten Audioinhalts und der ausgewählten Hintergrundmusik (Mastering).[71]

[63] AWS Documentation (2010), S. 1
[64] Vgl. ebd. S. 2
[65] Vgl. ebd. S. 25
[66] Vgl. ebd. S. 25
[67] Vgl. ebd. S. 2
[68] Vgl. ebd. S. 1
[69] Siehe dazu Coyle (o. D.)
[70] Siehe dazu Api.audio Documentation (o. D. -a)
[71] Vgl. Api.audio Documentation (o. D. -b)

Jede dieser Grundoperationen ist einzeln und unabhängig von den anderen Operationen verwendbar. Kunden sollen somit beispielsweise in der Lage sein, eigene Audiodateien um Hintergrundmusik zu erweitern und bereits existierenden Content neu zu synthetisieren.

Aflorithmic bietet ihren Kunden unterschiedliche Abonnement-Pakete zu unterschiedlichen Preisen an. Je teurer das gewählte Paket, desto mehr Production-Credits - welche für API-Calls verwendet werden - erhält der Kunde. Weitere Vorteile, wie das Synthetisieren von Audioinhalten ohne Audio-Watermark, Support per Telefon statt per E-Mail und die Möglichkeit, einen Report über die API-Verwendung zu erhalten, sollen den Kunden dazu anregen, ein tendenziell teureres Abonnement zu erwerben. In diesem Kontext soll auch die Voice Cloning App einen Anreiz schaffen, ein teureres Abonnement zu buchen, da der Zugang zu dieser erst ab 199 $ pro Monat aktiviert wird.[72]

Da es sich bei der Voice Cloning App um eine Erweiterung des API-Services handelt, ist es von großer Wichtigkeit, die eigens erstellte Stimme auch in die API zu integrieren. Die Speech-Operation soll somit um das individualisierte TTS Modell ausgebaut werden.

# 4 Die Voice Cloning App

In diesem Kapitel werden, nach einer kurzen Vorstellung der Voice Cloning App Aflorithmics in 4.1, die wichtigsten Prozesse innerhalb der App beschrieben, um dem Leser ein grobes Bild über die verwendeten Ressourcen und angewendeten Strukturen zu geben. So behandelt das Kapitel 4.2 den Registrierungs- und Login-Prozess. Anschließend wird innerhalb des Kapitels 4.3 die Darstellung der vorzulesenden Texte erläutert. Das Kapitel 4.4 setzt sich mit dem Hochladeprozess sowie der Verwaltung der hochgeladenen Audiodateien auseinander.

## 4.1 Vorstellung der Voice Cloning App

Loggt sich ein User in die Voice Cloning App ein, so wird diesem ein Satz angezeigt, welcher nach Drücken des Aufnahmeknopfes vorgelesen werden soll. Die Abbildung

[72] Vgl. Aflorithmic (o. D. -b)

1 stellt dabei den Aufnahmeprozess dar, welcher per Drücken des Mikrofonknopfes gestartet wird.

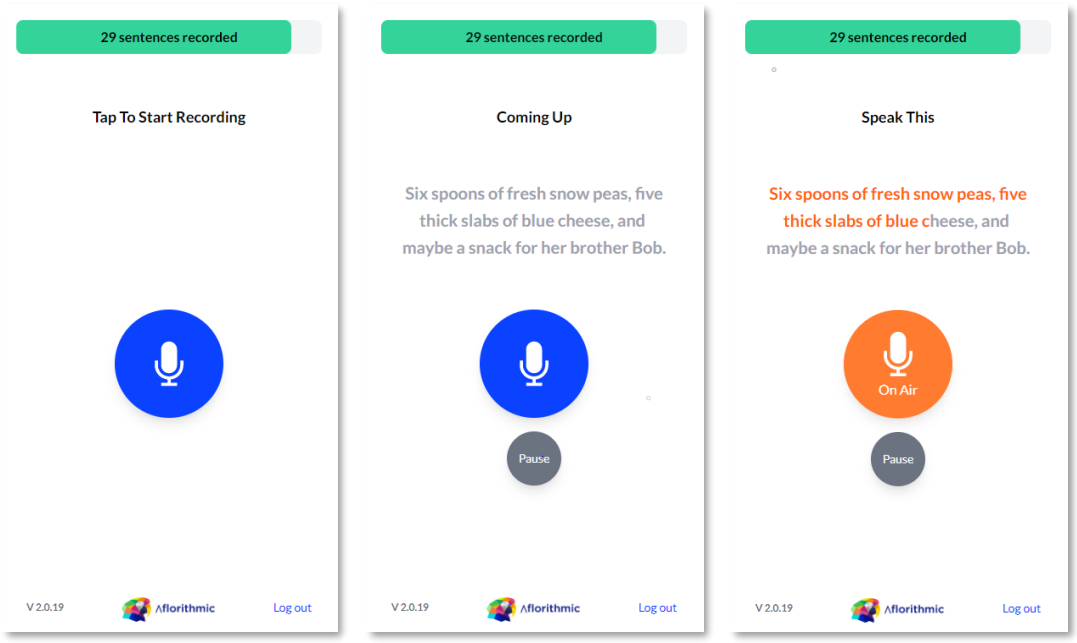


**Abbildung 1: Aufnahmeprozess in der Voice Cloning App**[73]

Die dabei erstellte Audiodatei wird direkt in die Aflorithmic-Umgebung hochgeladen. Hat ein User ausreichend viele Sätze aufgenommen, so wird ein TTS Modell mit diesen trainiert. Die Grundform des TTS Modells wurde dabei bereits mit Audioaufnahmen anderer trainiert und soll mit den Text-Audiopaaren des Users aus der Voice Cloning App fein abgestimmt werden.[74] Anfänglich liegt nur ein kleiner Datensatz mit den Audioaufnahmen und den korrespondierenden Texten der Aufnahmen vor. Um dennoch eine hohe Qualität der später synthetisierten Audioinhalte zu garantieren, werden dem Trainingsprozess Audioaufnahmen anderer Sprecher hinzugefügt. Diese wurden via Voice Conversion so verändert, dass sie sich wie Audioaufnahmen des Originalsprechers anhören.[75]

## 4.2 Registrierung und Login

Nach aktuellem Stand der Voice Cloning App ist dem User immer eine Organisation zuzuordnen. Diese Beziehung zwischen dem User und der Organisation kann aus diesem Grund als eine 1:1-Beziehung beschrieben werden. Demnach muss im Prozess

[73] Auszug aus der Voice Cloning App Aflorithmics
[74] Vgl. Tan et al. (2021), S. 30f.
[75] Vgl. ebd. S. 22, Sisman et al. (2020)

der Registrierung sowohl ein Organisationsname als auch ein Username angegeben werden. Um zu ermöglichen, dass User oder Organisationen mit gleichem Namen existieren können, wird beiden Parametern ein Hash angefügt. Dementsprechend erhält der User *Max Mustermann* beispielsweise den Usernamen *Max_Mustermann-8f6597aa* und ist der Organisation *Aflorithmic-d6d6f709* zugehörig. Die Hashes werden den Usern jedoch nicht angezeigt. Für jede Organisation wird im Rahmen der Registrierung ein Bearer-Token generiert, welcher die Organisation eindeutig identifiziert. Dieser Token garantiert einen autorisierten Zugang zu den Services Aflorithmics. Erhält das Backend der Voice Cloning App eine Request ohne validen Bearer-Token im Header der Request, so wird der Zugang verweigert (*statusCode 401*). Ergo erhalten nur legitimierte User Zugang zu Produkten Aflorithmics.

In diesem Konstrukt ist der MS-Auth für einen ordnungsgemäßen Registrierungs- und Login-Prozess verantwortlich. Beim Login des Users in die Voice Cloning App wird eine Request an diesen MS gesendet und die Logindaten auf ihre Richtigkeit überprüft. Resultierend daraus wird dem User der Zugang gewährt oder verweigert. Nach einem erfolgreichen Login erhält das Frontend den Bearer-Token der Organisation vom MS-Auth. Dieser wird schließlich in den Header aller ausgehenden Requests hinzugefügt. Ein Lambda-Decorator kann serverseitig zu Lambdafunktionen hinzugefügt werden. Ein solcher Decorator ermittelt den Organisationsnamens per Bearer-Token und fügt ihn dem eingehenden Event der Lambdafunktion hinzu. Der Lambda-Decorator selbst ist dabei im direkten internen Austausch mit dem MS-Auth und vereinfachen den Informationsaustausch zwischen den Microservices und der User-Datenbank, welche von dem MS-Auth verwaltet wird.

## 4.3 Darstellung der vorzulesenden Texte

Innerhalb Aflorithmics Voice Cloning App werden dem User einzelne Sätze angezeigt, welche es aufzunehmen gilt. Dabei wird jeder Satz genau einmal angezeigt und somit auch einmal aufgenommen. Der dafür zuständige Voice Cloning Microservice (MS-VC) beinhaltet das Lambda *get_text*, dessen Endpunkt nach einem erfolgreichen Login vom Frontend angefragt wird, um die Liste an aufzunehmenden Sätzen zu erhalten.

```
GET 'https://n4y7f73jng.execute-api.eu-west-1.amazonaws.com/staging/ulvc/clon-
ing/textfiles?userName=Max_Mustermann-8f6597aa' --header 'Authorization: <Bea-
rer-Token>
```

**Abbildung 2: HTTP-GET-Request zum Erhalten der vorzulesenden Texte-ID-Paare**[76]

Die in Abbildung 2 dargestellte Request des Frontends übermittelt den Usernamen des eingeloggten Users sowie die Identifikation der Organisation per Bearer-Token. Diese Parameter werden innerhalb des *get*_text-Lambdas aufgenommen, um den User und die Organisation zu identifizieren. Anschließend werden alle vorzulesenden Sätze der Voice Cloning Datenbank eingelesen. Die Elemente dieser DynamoDB-Tabelle bestehen dabei aus dem Attribut *Text* und dem Attribut *ID* zur eindeutigen Identifizierung des gesprochenen Satzes.

Um im nächsten Schritt abzugleichen, welche Texte bereits von einem User aufgenommen wurden, ist es naheliegend, dass die aufgenommenen Audiodateien nach der *ID* des korrespondierenden Textes benannt werden. Entspricht beispielsweise der *ID 0001* dem Text *Hello World*, so erhält die Audiodatei nach der Aufnahme des Satzes den Namen *0001*. Aus diesem Grund werden die *IDs* aller vorzulesender Texte mit den Namen der in s3 liegenden Audioaufnahmen des Users verglichen, um alle noch vorzulesenden Texte herauszufiltern. Die übrigbleibenden *IDs* werden samt den korrespondierenden Sätzen (*Text*) im JSON-Format (siehe Abbildung 3) und dem *Statuscode 200* an das Frontend zurückgeschickt.


```
{
    "textFiles": [
    {
                "ID": "0001",
                "text": "When the sunlight strikes raindrops in the air, they
act as a prism and form a rainbow."
            },
            {
                "ID": "0002",
                "text": "The rainbow is a division of white light into many be
autiful colors."
            },
],
    "fileCount": 29
}
```


[76] Eigene Darstellung

**Abbildung 3: Body der Response samt Text-ID-Paare und fileCount**[77]

Die einzelnen noch vorzulesenden Sätze können nun vom Frontend der Voice Cloning App nacheinander dargestellt werden.

## 4.4 Hochladen und Verwalten der Audiodateien

Der MS-File ist für das Verwalten von Dateien zuständig. Sollen Dateien hoch- oder heruntergeladen werden, so muss dieser Service angefragt werden. Der MS-File beinhaltet den *voice-cloning*-Bucket, in welchem die Audioaufnahmen der User gespeichert werden. Um die Audioaufnahmen der App eindeutig einem User zuzuordnen, wurde folgender Objekt-Schlüssel für die Objekte des *voice-cloning*-Buckets definiert.

$Objektname =$ Organisationsname/Username/Name der Audiodatei

Wie zu erkennen ist, wird der zuvor erläuterte Name der Audiodatei um den Organisationsnamen und den Usernamen erweitert. Nimmt beispielsweise der User mit dem Usernamen *Max_Mustermann-8f6597aa*, welcher der Organisation *Aflorithmic-d6d6f709* zugehörig ist, den Text mit der *ID 0001* auf, so wird dieses Objekt im s3-Bucket mit dem Schlüssel *Max_Mustermann-8f6597aa/Aflorithmic-d6d6f709/0001* eindeutig identifiziert. Für die anschließenden Audioaufnahmen des Textes mit der *ID 0002* und *0003* gilt selbiges Schema.

Um die größtmögliche Sicherheit der hochgeladenen Objekte zu gewährleisten, werden die Objekte im s3-Bucket als *private* Objekte deklariert. Somit müssen presigned-URLs verwendet werden, um Objekte in den s3-Bucket hoch- bzw. herunterzuladen. Die Boto3 Funktion *generate_presigned_url* wird zum Generieren von solchen URLs verwendet. Wurde eine Audiodatei im Frontend der Voice Cloning App erfolgreich aufgenommen, so wird der HTTP-Endpunkt des Lambda *get_vc_upload_url* - welcher dem MS-File zugehörig ist - angefragt. Dieser benötigt lediglich die URL-Parameter *name* (Name des Users) und *id* (Name der Datei), um eine presigned-URL zu erstellen. Diese URL wird an das Frontend zurückgesendet, welches eine HTTP-PUT-Request an die erhaltende URL sendet. Die Audiodatei befindet sich dabei im Body der Request. Mit diesem Prozess wurde das Objekt erfolgreich in den s3-Bucket hochgeladen.

[77] Eigene Darstellung

# 5 Audio Evaluation

Die Aufgabe der Audioevaluierung lässt sich in vier Unterpunkte unterteilen. Zum einen sollen die Audioaufnahmen sowohl in ihrer Qualität eingeschätzt als auch interpretiert werden, zum anderen müssen die daraus resultierenden Daten verwaltet und bei Bedarf kommuniziert werden. Während der Audio-Evaluation-Service (AES) für die Qualitätsermittlung der Audiodateien zuständig ist, muss der zu konzipierende Microservice für die Verwaltung und Kommunikation zwischen Applikationen und dem Service modelliert werden.

Dafür wird innerhalb dieses Kapitels zunächst auf die wichtigsten Eigenschaften und Strukturen des AES (5.1) sowie die damit verbundenen Anforderungen an den Microservice (5.2) eingegangen. Anschließend folgt die Modellierung (5.3) und Implementierung des Microservices (5.4).

## 5.1 Der Audio-Evaluierungs-Service

Der Audio-Evaluation-Service (AES) bietet die Möglichkeit eine oder mehrere Audiodateien auf ihre Qualität zu überprüfen. Der vom Service zur Verfügung gestellte Endpunkt erwartet einen HTTP-POST-Request samt einer Liste im Body der Request, die einen oder mehrere s3-Schlüssel beinhalten kann. Die entsprechenden Objekte der Schlüssel werden mithilfe von presigned-URLs heruntergeladen und in folgenden Kategorien überprüft:

1. Allgemeine Lautstärke (*too_loud*, *too_quiet*),
2. Inkonsistenzen der Lautstärke (inconsistent),
3. Unterbrechungen im Lesefluss (interruptions)
4. Den Hall des Raumes (*high_reverb*),
5. Übersteuerung (clipping)
6. Die Verständlichkeit des eingesprochenen Textes (low_text_coverage)

Das jeweilige Ergebnis der untersuchten Kategorie wird dabei von verschiedenen Python-Bibliotheken ermittelt und zunächst durch einen einzigen Wert repräsentiert. So ist bspw. die Audiodatei mit dem Wertepaar *low_text_coverate* = 0 unverständlicher als eine Audiodatei mit dem Wertepaar *low_text_coverate = 13,09*. Auf Basis von bisherigen, nicht evaluierten Audioaufnahmen wurden für jede Kategorie gewisse

Schwellenwerte festgelegt, die zur anschließenden Interpretation dienen. Wird der Schwellenwert der Kategorie übertroffen, entspricht das Ergebnis der Audiodatei dem booleschen *True* (*too_loud = True*). Wird ein Schwellenwert nicht übertroffen, gilt gegenteiliges (too_loud = False). Somit wird eine Kategorie fortan mit zwei Werten beschrieben, welche in einer Liste festgehalten werden. (*low_text_coverate = [0, True])*. Soll eine Datei evaluiert werden, deren Audiolänge der festgelegten Eine-Sekunde-Marke unterliegt, so wird diese als ungültig deklariert (invalid = True).

Nach der Bewertung aller zum AES gesendeten Audiodateien, werden die individuellen Evaluierungsergebnisse in einer Liste zusammengefasst. Da es sich ursprünglich um einen HTTP-POST-Request handelt, besteht die Option, die Liste an Evaluierungsergebnisse als HTTP-Response zurückzuschicken. Jedoch obliegt diese Entscheidung dem verknüpften Microservice.

## 5.2 Anforderungen an den Microservice

Wie zuvor erläutert, dient der Microservice des Audio-Evaluierungs-Services der Kommunikation und Datenverwaltung zwischen dem AES und der Voice Cloning App. Im Sinne des Microservice-Begriffs soll dabei eine in sich geschlossenen Funktionalität abgebildet werden. Zusätzlich dazu soll die Audioevaluierung die Voice Cloning App erweitern, wodurch die wichtigste Voraussetzung für den MS-AES eine einfache Implementierung in die bereits vorhandene Architektur ist. Für die clientseitige Applikation soll es zwar möglich sein, die Ergebnisse des AES zwecks Darstellung zu erhalten, jedoch soll das Frontend weitgehend unverändert bleiben, wodurch die Audioevaluierung automatisch aktiviert bzw. auslöst werden soll.[78] Demnach soll der zu konzipierende Microservice eine Request an den AES kreieren und senden, welche durch ein backendseitiges Ereignis ausgelöst wurde. Zusätzlich soll jede evaluierte Audiodatei lediglich einmal angezeigt werden, um den User nicht mit einem Überfluss an Evaluierungsergebnissen zu überschütten. Neben der Zurverfügungstellung der Ergebnisse soll bezüglich der Datenverwaltung jede neu aufgenommene Datei nur einmal evaluiert werden. Seitens des AES wurde für den Evaluierungsvorgang eine

[78] Eine weitere Möglichkeit wäre das Auslösen des Evaluierungsprozesses durch den User, also durch ein frontendseitiges Ereignis.

Charge von 10 Audiodateien vorgegeben. Im Falle einer ungültigen Datei ist ein automatisierter Löschvorgang erwünscht. Die zu verwendenden Ressourcen beschränken sich auf die der AWS Umgebung.

## 5.3 Modellierung des Microservices

Aufgrund der angestrebten möglichst einfachen Implementierung wurde zunächst die aktuell verwendete Infrastruktur der Voice Cloning App in ihren relevanten Ressourcen und Funktionalitäten modelliert (siehe Abbildung 4), um diese anschließend zu erweitern.

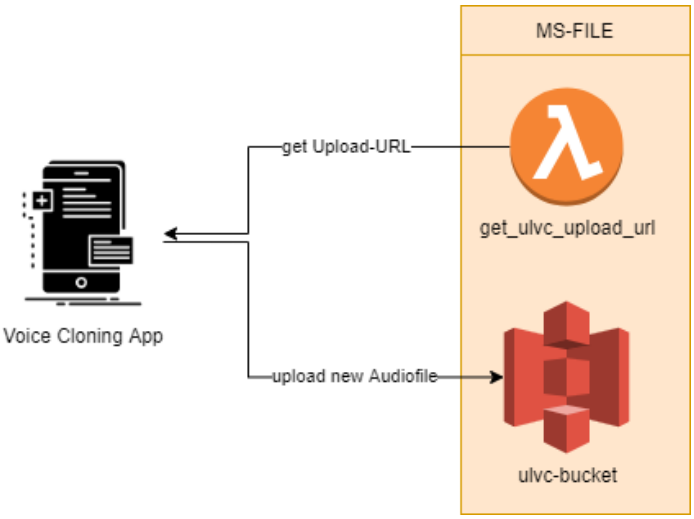


**Abbildung 4: Modell der relevanten Infrastruktur der Voice Cloning App**[79]

Dabei fokussiert sich das Kapitel 5.3.1 zunächst darauf, den Evaluierungsprozess zu automatisieren. Anschließend wird eine mögliche Struktur zum sinnvollen Speichern der daraus resultierenden Ergebnisse und die damit einhergehende Bereitstellung eines Endpunktes zur Informationsbeschaffung zwecks Datendarstellung in 5.3.2 modelliert. Zuletzt wird diese Struktur um die zuvor genannten Ansprüche der Datenverwaltung in Kapitel 5.3.3 erweitert.

### 5.3.1 automatisierter Aufruf des AES

Wie in Kapitel 5.1 erläutert, werden die s3-Schlüssel der aufgenommenen Audiodateien für den Evaluierungsprozess benötigt. Zudem befinden sich die zu evaluierenden Audiodateien im *voice-cloning*-Bucket, welcher dem MS-File angehörig ist. Um jede

[79] Eigene Darstellung

neue Audiodatei innerhalb des s3-Buckets zum AES zwecks Evaluation weiterzuleiten, müssen diese zunächst vom MS-File selbst erfasst werden. Im Sinne der ereignisgesteuerten Architektur der AWS-Services wird der MS-File um das Lambda *aes-audiofile-forwarder* erweitert. Die Funktion dieses Lambdas soll aufgerufen werden sobald ein neues Objekt in den *voice-cloning*-Bucket hochgeladen wurde[80] (siehe Abbildung 5).[81] Dieses eingehende Event beinhaltet unter anderem den s3-Schlüssel des neu hochgeladenen Objektes.[82] Die Aufgabe des Lambdas ist es, diesen Schlüssel zu extrahieren und der SQS-Queue *aes-audiofile-queue* als Nachricht hinzuzufügen. Die SQS-Queue *aes-audiofile-queue* ist dabei als erster Baustein des MS-AES zu verstehen und ermöglicht die Kommunikation zwischen dem MS-AES und dem MS-File (siehe Abbildung 5). Um die einzelnen s3-Schlüssel aus der SQS-Queue herauszulesen und diese für eine Evaluierung zum AES weiterzuleiten, wird das Modell auf der Seite des MS-AES um das Lambda *aes-audiofile-queue-forwarder* erweitert, welches auf neue Nachrichten innerhalb der SQS-Queue wartet (siehe Abbildung 5). Sendet das *aes-audiofile-queue-forwarder*-Lambda anschließend eine Request an den AES, startet der in Kapitel 5.1 beschriebene Prozess.

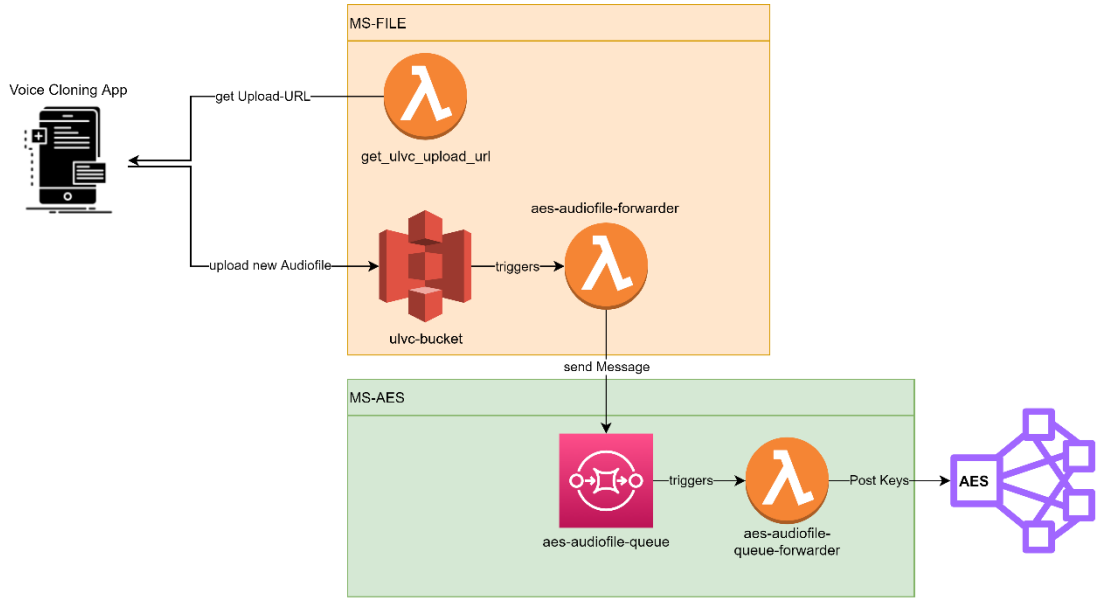


**Abbildung 5: Modell zum automatisierten Aufrufen des AES[83]**

[80] Wobei das Hochladen des Objekts das Event darstellt
[81] Vgl. Serverless Framework Documentation (o. D. -a)
[82] Vgl. AWS Documentation (o. D. -a), S. 578
[83] Eigene Darstellung

Mittels der in Abbildung 5 dargestellten Modellierung werden die Schlüssel der neu hinzugefügten Objekte des *voice-cloning*-Buckets automatisch an den MS-AES übermittelt und die korrespondierenden Audiodateien mittels des AES evaluiert.

### 5.3.2 Erstellung eines Endpoints zur Darstellungsermöglichung

Innerhalb dieses Unterkapitels sollen die Ergebnisse des AES strukturiert, gespeichert und zur clientseitigen Darstellung bereitgestellt werden. Dafür wird das Modell zunächst um die SQS-Queue *aes-eval-score-queue* ergänzt, an welche die Evaluierungsergebnisse des AES gesendet werden[84] (siehe Abbildung 6). Um die Nachrichten innerhalb dieser SQS-Queue zu verarbeiten, wird dem Modell zusätzlich das *aes-feedback-safer*-Lambda hinzugefügt, welches für jede Nachricht innerhalb der *aes-eval-score-queue*-SQS-Queue ausgelöst werden soll. Des Weiteren sollen mittels dieses Lambdas die Evaluierungsergebnisse gespeichert werden. Dafür wird das Modell um die DynamoDB-Tabelle *aes-feedback-table* erweitert. Komplettiert wird die Modellerweiterung mit dem *aes-feedback-provider*-Lambda, welches einen HTTP-Endpunkt für das Frontend der Voice Cloning App zu Verfügung stellen soll. Wird der Endpunkt dieses Lambdas angefragt, so soll die *aes-feedback-table*-DynamoDB-Tabelle nach Inhalten abgefragt werden, die dem User frontendseitig angezeigt werden sollen. Die Abbildung 6 fasst die Erweiterungen des Modells zusammen.

[84] Diese Entscheidung erfordert es den AES zu updaten und die Evaluierungsergebnisse der *aes-eval-score-queue*-SQS-Queue mittels der send_message()-Funktion hinzuzufügen. (Siehe dazu Kapitel 2.3.4)

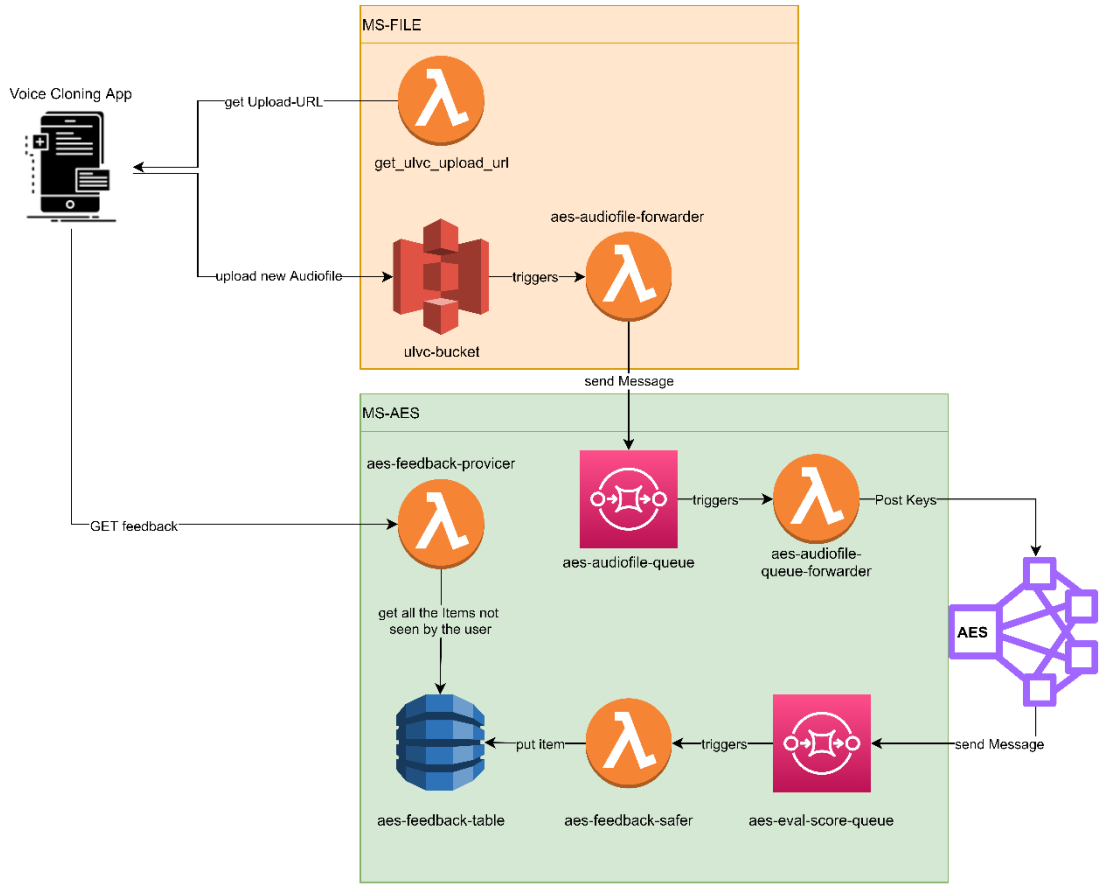


**Abbildung 6: Modell zur Darstellungsermöglichung der evaluierten Dateien**

Ferner gilt es gemäß Kapitel 2.3.2 einen Primärschlüssel für die eben hinzugefügte DynamoDB-Tabelle zu erstellen. Dabei soll jedes Element der Tabelle eine evaluierte Audiodatei verkörpern, deren Evaluierungsergebnis dem User frontendseitig angezeigt werden kann. Dementsprechend können und müssen Audiodateien immer auch dem User zuordenbar sein, da dieser die Audiodatei per Voice Cloning App aufgenommen hat. Naheliegend ist somit ein zusammengesetzter Partitionsschlüssel bestehend aus dem jeweiligen Organisationsnamen und Usernamen. Ebendiese Struktur ermöglicht das Abfragen aller Elemente der DynamoDB-Tabelle, die dem User bzw. der Organisations-User-Kombination zuordenbar sind. Für eine eindeutige Identifikation der Elemente muss zusätzlich ein Sortierschlüssel definiert werden. Hierbei lässt sich die *ID* der Audiodatei als würdige Kenngröße bezeichnen, da sich die einzelnen Audiodateien eines Users in diesem Parameter unterscheiden. Zur Verbesserung der Lesbarkeit werden die Schlüssel um Trennelemente sowie Trennparameter erweitert. Die Struktur des Primärschlüssels kann dem in Abbildung 7 dargestellten Beispiel entnommen werden.

| Partitionsschlüssel | Sortierschlüssel |
|---|---|
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0001 |
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0002 |
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0003 |

**Abbildung 7: feedback-Tabelle, Struktur des Primärschlüssels**[85]

Um die Evaluationsergebnisse des AES anschließend sinngemäß und strukturiert der DynamoDB-Tabelle hinzuzufügen, wird die Tabelle zunächst um das *evaluationCode*-Attribut erweitert. Darin soll das *aes-feedback-safer* Lambda - nach Bildung des Primärschlüssels - die Evaluierungsergebnisse der entsprechenden Audiodatei im JSON-Format speichern.

Der mittels des *aes-feedback-provider*-Lambdas zur Verfügung gestellte HTTP-Endpunkt soll im Anschluss daran angefragt werden können, um die Evaluierungsergebnisse eines Users aus der DynamoDB-Tabelle zusammenzufassen und zurückzuschicken. Dabei gilt es zusätzlich zu beachten, dass lediglich noch nicht dargestellte Evaluierungsergebnisse zurückgesendet werden sollen. In diesem Sinne wird die Struktur der Datenbanktabelle um den *sentToFrontend*-Parameter erweitert (siehe Abbildung 8), dessen Wert beim Eintragen der Evaluationsergebnisse auf *False* gesetzt wird.

Wird anschließend der HTTP-Endpunkt des *aes-feedback-provider*-Lambdas angefragt, wird sowohl ein Bearer-Token zur Autorisierung und Identifikation der Organisation[86] als auch der Username als URL-Parameter, zur Identifikation des Users benötigt, da sich anhand dieser Parameter der Primärschlüssel der DynamoDB-Tabelle bilden lässt. Dieser Primärschlüssel kann im Anschluss daran verwendet werden, um alle Elemente des Users innerhalb der DynamoDB-Tabelle abzufragen, wobei der *sentToFrontend*-Parameter *False* entsprechen soll. Anschließend kann dieser Parameter vom *aes-feedback-provider*-Lambda zu *True* upgedatet werden.

Folglich ist die Struktur der DynamoDB-Tabelle der Abbildung 8 zu entnehmen.

[85] Eigene Darstellung
[86] Siehe hierzu Kapitel 4.3

| Partitionsschlüssel | Sortierschlüssel | evaluationCode | sentToFronend |
|---|---|---|---|
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0001 | {...} | True |
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0002 | {...} | False |
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0003 | {...} | False |

**Abbildung 8: feedback-Tabelle, Ergänzung um die Attribute evaluationCode und sentToFrontend**[87]

### 5.3.3 Datenverwaltung innerhalb des Microservices

Innerhalb der Datenverwaltung soll dafür gesorgt werden, dass jede Audiodatei genau einmal evaluiert wird. Zudem wurde seitens des AES vorgegeben, die Audiodateien in userbezogene Chargen von je 10 Audiodateien zu evaluieren. Im Falle einer ungültigen Audiodatei soll außerdem ein zu automatisierender Löschvorgang aktiviert werden, der die ungültige Datei löscht, wodurch der User den jeweiligen Satz erneut in der Voice Cloning App einsprechen muss.

Um zunächst zu registrieren, ob eine Audiodatei bereits evaluiert wurde, ist es vonnöten, die Elemente der SQS-Queue *aes-audiofile-queue* in einer Tabelle zu speichern. Dieser Prozess soll mittels des *aes-audiofile-queue-forwarder*-Lambdas realisiert werden (siehe Abbildung 10). Zudem wird für diesen Zweck die DynamoDB-Tabelle *aes-feedback-table* um das boolesche Attribut *sentToAes* erweitert, um zu signalisieren, ob eine Evaluation bereits durchgeführt wurde (siehe Abbildung 9). Speichert das aes-*audiofile-queue-forwarder*-Lambda ein Element in der DynamoDB-Tabelle ab, so soll der *sentToAes*-Parameter als *False* deklariert sein. Wird der s3-Schlüssel eines Elements anschließend an den AES gesendet, um die Audiodatei zu evaluieren, so soll der *sentToAes*-Parameter zu *True* upgedatet werden. Mittels einer solchen Struktur können noch zu evaluierende Audiodateien ausfindig gemacht werden. Zudem kann das *evaluationCode*-Attribut durch das *aes-audiofile-queue-forwarder*-Lambda auf *empty* gesetzt werden, was unausgefüllte Attributs-Werte in der DynamoDB-Tabelle verhindert. Folglich muss im Rahmen des *aes-feedback-provider*-Lambdas ebenfalls auf den Wert des *evaluationCode*-Attributs geachtet werden. Entspricht dieser dem String *empty*, so darf das Element nicht an das Frontend weitergeleitet werden, obwohl der *sentToFrontend*-Parameter den Wert *False* aufweist.

[87] Eigene Darstellung

Da je 10 Audiodateien desselben Users gesammelt zum AES gesendet werden sollen, muss das aes-*audiofile-queue-forwarder*-Lambda die DynamoDB-Tabelle *aes-feedback-table* nach der Anzahl der Elemente des jeweiligen Users abfragen, deren *sentToAes*-Attribut *False* entspricht. Sollte die Anzahl 10 entsprechen, so können die s3-Schlüssel der 10 Audiodateien an den AES gesendet werden. Daran anschließend kann das *sentToAes*-Attribut der 10 Elemente innerhalb der DynamoDB-Tabelle von *False* zu *True* upgedatet werden.

Um ungültige Audiodateien mittels eines automatischen Löschvorgangs zu entfernen, müssen die Evaluierungsergebnisse der Audiodateien interpretiert werden. Sollte sich innerhalb eines Evaluierungsergebnisses der *invalid*-Parameter (invalid = True) befinden, gilt es die jeweilige Audiodatei zu löschen. Wie zuvor erläutert befinden sich die Audiodateien unter der Verwaltung des MS-File. Ebendieser Microservice ist somit auch für das Löschen der Audiodateien zuständig.

Da die Evaluierungsergebnisse des AES vom *aes-feedback-safer*-Lambda aufgegriffen werden sollen, kann dieses Lambda die Ergebnisse auf den *invalid*-Parameter (invalid = True) überprüfen. Dem Modell wird zusätzlich die SQS-Queue *aes-delete-audiofile-forwarder* hinzugefügt (siehe Abbildung 10). Dieser Queue kann im Falle einer ungültigen Audiodatei der s3-Schlüssel dieser Datei hinzufügen werden, wodurch der automatische Löschvorgang aktiviert wird. Um diesen Prozess nachzuvollziehen und zu signalisieren, wird die DynamoDB-Tabelle *aes-feedback-table* um das *deleted*-Attribut ergänzt (siehe Abbildung 9). Dieses boolesche Attribut soll, vergleichbar mit dem *sentToAes*-Attribut, bei anfänglicher Abspeicherung durch das aes-*audiofile-queue-forwarder*-Lambda, als *False* deklariert werden. Wird die Audiodatei gelöscht bzw. der s3-Schlüssel der Audiodatei der *aes-delete-audiofile-forwarder*-SQS-Queue hinzugefügt, so soll das *deleted*-Attribut zu *True* upgedatet werden. Auch hier soll die SQS-Queue als Vermittler zwischen dem MS-AES und dem MS-File fungieren. Somit wird dem Modell auf der Seite des MS-File das *aes-audiofile-deleter*-Lambda hinzugefügt (siehe Abbildung 10). Dieses Lambda soll für jede neue Nachricht innerhalb der *aes-delete-audiofile-forwarder*-SQS-Queue ausgelöst werden und das Objekt des übermittelten s3-Schlüssels aus dem s3-Bucket *voice-cloning* löschen. Sinngemäß wird aufgrund der geänderten Funktionalität das *aes-feedback-safer*-Lambda zu das *aes-feedback-interpreter* umbenannt.

Die finale Version der DynamoDB-Tabelle aes-*feedback-table* kann der Abbildung 9 entnommen werden.

| Partitionsschlüssel | Sortierschlüssel | evaluationCode | sentToFronend | sentToAes | deleted |
|---|---|---|---|---|---|
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0001 | {...} | True | False | False |
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0002 | empty | False | False | False |
| orgId\|Aflorithmic-d6d6f709\|userId\|Max_Mustermann-8f6597 | filename\|0003 | {...} | True | False | False |

**Abbildung 9: feedback-Tabelle, Ergänzung um die Attribute sentToAes und deleted**[88]

Die finale Modellierung des Microservices samt Erweiterung des MS-File zu Kommunikationszwecken wird in der Abbildung 10 dargestellt.

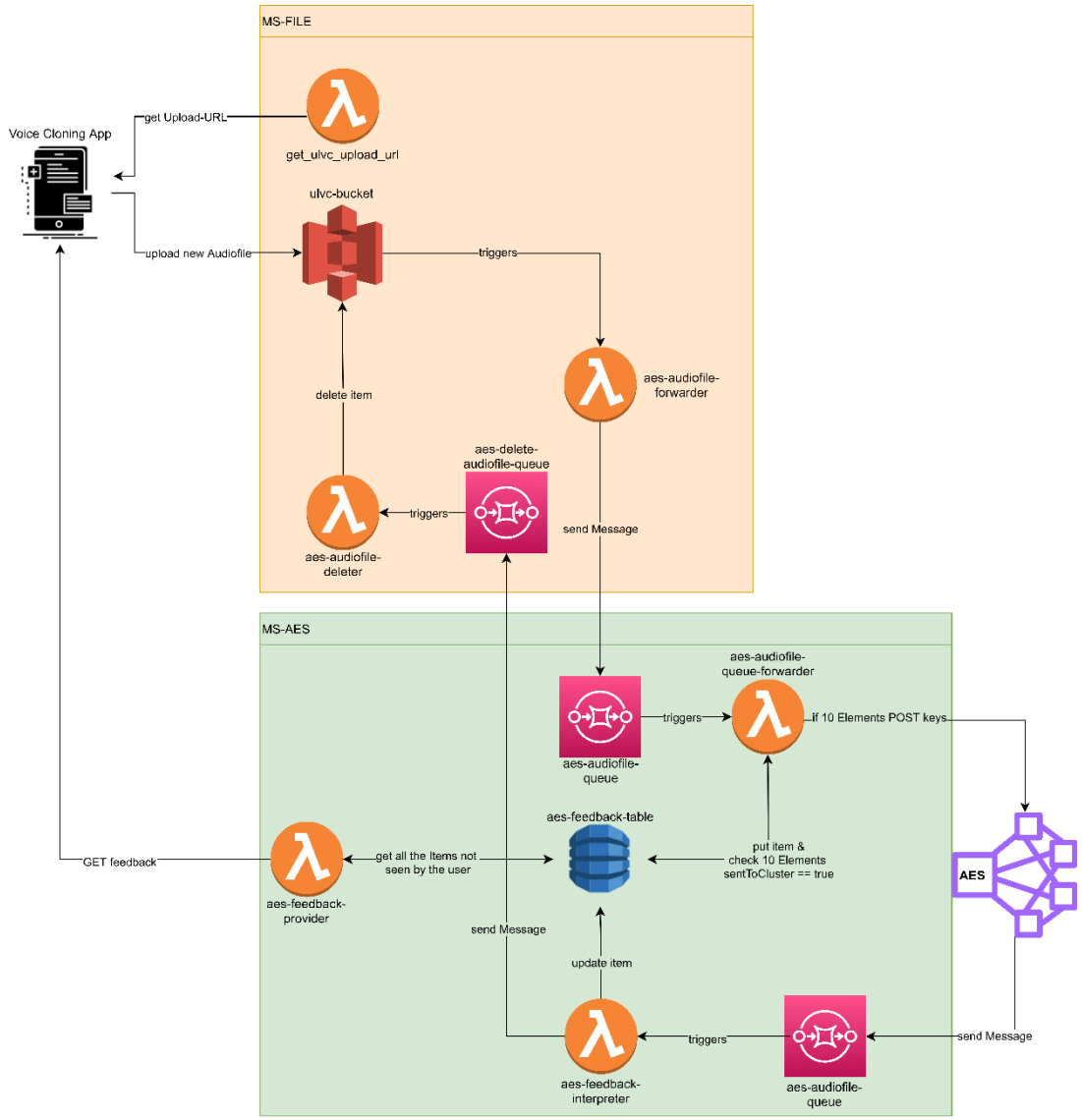


**Abbildung 10: Finales Modell des MS-AES**[89]

[88] Eigene Darstellung
[89] Eigene Darstellung

## 5.4 Implementierung des Microservices

Im Laufe des Kapitels 5.3 stellten sich unterschiedliche Anforderungen an die einzelnen Lambdas heraus. Diese werden innerhalb dieses Kapitels kurz zusammengefasst und anschließend programmatisch umgesetzt. Dabei werden zunächst alle benötigten Ressourcen erstellt, um anschließend die einzelnen Lambdafunktionen mit Quellcode zu füllen.

### 5.4.1 Funktionalitäten der Lambdas

Das Modell beinhaltet insgesamt fünf Lambdas, deren Funktionalitäten im Laufe der Modellierung anstiegen und variierten. Somit werden diese vor der Phase der Implementierung, sortiert nach der Reihenfolge ihres Aufrufs, zusammengefasst.

1. aes-audiofile-forwarder

   Dieses Lambda wird für jedes neue Item in dem *voice-cloning*-Bucket aufgerufen und hat die Aufgabe den s3-Schlüssel aus dem eingehenden Event zu extrahieren, um diesen anschließend der SQS SQS-Queue *aes-audiofile-queue* als Nachricht hinzuzufügen.

2. aes-audiofile-queue-forwarder

   Das *aes-audiofile-queue-forwarder*-Lambda wird von jeder Nachricht innerhalb der SQS-Queue *aes-audiofile-queue* aufgerufen. Dabei wird auch hier zunächst der s3-Schlüssel aus der Nachricht extrahiert. Anschließend wird mithilfe des Bearer-Tokens und dem s3-Schlüssel der Primärschlüssel - bestehend aus Partitionsschlüssel und Sortierschlüssel - der Audiodatei erstellt. Der Partitionsschlüssel dient dem Abfragen der DynamoDB-Tabelle *aes-feedback-table* nach Elementen mit gleichen Partitionsschlüssel und einem *sentToAes*-Attribut mit der Wertigkeit *False*. Werden von der Tabelle neun Elemente zurückgegeben, so befinden sich die s3-Schlüssel von zehn nicht evaluierten Audiodateien in der Lambdafunktion. Infolgedessen wird eine HTTP-POST-Request an den AES gesendet, wobei die s3-Schlüssel der zehn Audiodateien übermittelt werden. Abschließend wird das in der DynamoDB-Tabelle fehlende Element hinzugefügt. Dabei entspricht der Wert der Attribute *sentToFrontend*, *sentToAes* und *deleted*, jeweils *False*. Das *evaluationCode*-Attribut wird mit dem String *empty* versehen.

3. aes-feedback-interpreter

Hier wird das Evaluierungsergebnis des AES interpretiert. Dafür wird das Ergebnis jeder Audiodatei auf den *invalid*-Parameter überprüft (invalid = True). Im Falle einer ungültigen Audiodatei wird der s3-Schlüssel dieser der *aes-delete-audiofile-queue*-SQS-Queue hinzugefügt, wodurch der Löschprozess beginnt. Anschließend gilt es, die Elemente der DynamoDB-Tabelle *aes-feedback-table* upzudaten, wobei der *evaluationCode*-Parameter mit dem jeweiligen Evaluierungsergebnis der Audiodatei ausgefüllt wird. Das *deleted*-Attribut kann innerhalb dieses Prozesses, abhängig vom Evaluierungsergebnis, von *False* auf *True* gesetzt werden.

4. aes-audiofile-deleter

Auch dieses Lambda wird anhand einer SQS-Queue aufgerufen, befindet sich jedoch innerhalb des MS-File. Befindet sich eine Nachricht innerhalb der SQS-Queue *aes-delete-audiofile-queue*, wird der darin übermittelte s3-Schlüssel extrahiert und die damit verbundene Audiodatei aus dem s3-Bucket *voice-cloning* gelöscht.

5. aes-feedback-provider

Um die Darstellung der Evaluierungsergebnisse zu ermöglichen, kann seitens des Frontends die URL des *aes-feedback-provider*-Lambdas per HTTP-GET-Request angefragt werden. Die Request muss dabei sowohl einen validen Bearer-Token als auch einen Usernamen als URL-Parameter beinhalten. Infolgedessen werden alle Elemente der DyanmoDB-Tabelle *aes-feedback-table* abgefragt, die dem User noch nicht angezeigt wurden (sentToFrontend = False). Zudem muss der Wert des *evaluationCode*-Attributs ungleich dem String *empty* sein. Nach Sammeln dieser Daten werden diese an das Frontend zurückgeschickt und können dort dem User angezeigt werden.

### 5.4.2 Erstellung der Ressourcen

Für eine schnellstmögliche und einfache Entwicklung des Microservices wird neben der Boto3 Bibliothek auf das Serverless Framework zurückgegriffen. Dieses bietet Entwicklern einen schnellen und unkomplizierten Weg an, eigene Services zu erstellen

sowie diese zu verwalten und upzudaten.[90] Um mittels Serverless einen neuen Service zu erstellen, muss lediglich das Framework installiert sowie dieses mit dem eigenen AWS Konto verknüpft[91] werden. Wird anschließend der Befehl „$ serverless“ in die CLI eingegeben, so wird ein Leitfaden hervorgerufen, welcher die Erstellung des neuen Services unterstützt. Dabei muss der Name des Services (MS-AES) sowie die darin verwendete Programmiersprache (Python) definiert werden. Anschließend werden zwei Dateien generiert, die dem Service eine Grundstruktur vorgeben. Das Python-Dokument *handler.py* entspricht der ersten vordefinierten Lambdafunktion des Services und beinhaltet den Quellcode, der beim Aufrufen des Lambdas ausgeführt wird. Die *serverless.yml*-Datei hingegen ist das Kernstück des Services und beinhaltet die allgemeine Konfiguration. Somit werden innerhalb dieser Datei alle benötigten Ressourcen sowie alle Lambdas und deren Trigger definiert.[92] Sollen Ressourcen oder Funktionen erweitert oder aktualisiert werden, bietet das Framework das Updaten des gesamten Services mittels eines einzigen Befehls an.[93] Dabei wird die YAML-Datei automatisch in ein CloudFormation-Template übersetzt und hochgeladen, wodurch ein neuer Stack erstellt wird. Dieser Stack beinhaltet dementsprechend die neuen Ressourcen und Funktionen bzw. die Veränderungen bestehender Ressourcen und Funktionen, wodurch diese erstellt, bereitgestellt bzw. aktualisiert werden.

Um Tippfehler zu vermeiden und den Entwicklungsprozess zu vereinfachen, werden die Namen der zu erstellenden Ressourcen als Umgebungsvariablen gespeichert. Eine solche Variable lässt sich in einer *serverless.yml*-Datei innerhalb des *custom*-Parameters definieren, wie in Abbildung 11 dargestellt.

```
custom:
  sqs:
    aesAudiofileQueue: AesAudiofileQueue
```

**Abbildung 11: serverless.yml-Datei, Definition einer Umgebungsvariable[94]**

Anschließend kann auf solch eine Variable entsprechend der Abbildung 12 referenziert werden[95]

[90] Vgl. Serverless Framework Documentation (o. D. -b)
[91] Vgl. Serverless Framework Documentation (o. D. -c)
[92] Vgl. Serverless Framework Documentation (o. D. -d)
[93] Vgl. Serverless Framework Documentation (o. D. -e)
[94] In enger Anlehnung an Serverless Framework Documentation (o. D. -f)
[95] Vgl. Serverless Framework Documentation (o. D. -f)

```
${self:custom.sqs.aesAudiofileQueue}
```

**Abbildung 12: serverless.yml-Datei, Referenzieren auf eine Umgebungsvariable[96]**

### (1) SQS

Um die beiden SQS-Queues *aes-audiofile-queue* und *aes-audiofile-queue* der Infrastruktur des MS-AES hinzuzufügen, wird die *serverless.yml*-Datei wie folgt erweitert (siehe Abbildung 13).

```
custom:
  sqs:
    aesAudiofileQueue: AesAudiofileQueue
    deadSqsAudiofileQueue: AesDeadAudiofileQueue
    aesEvalScoreQueue: AesEvalScoreQueue
    deadSqsEvalScoreQueue: AesDeadEvalScoreQueue
resources:
  Resources:
    AesAudiofileQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.aesAudiofileQueue}
    DeadSqsAudioQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.deadSqsAudiofileQueue}
        MessageRetentionPeriod: 1209600
    AesEvalScoreQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.aesEvalScoreQueue}
    DeadEvalScoreQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.deadSqsEvalScoreQueue}
        MessageRetentionPeriod: 1209600
```

**Abbildung 13: serverless.yml-Datei, Ergänzung um vier SQS-Queues[97]**

Wie zu erkennen, werden insgesamt vier Ressourcen innerhalb des *Resources*-Parameters definiert. Dabei lässt sich jede Ressource als SQS-Queue identifizieren, wie der Type-Parameter mit dem Wert *AES::SQS::QUEUE* signalisiert. Zudem werden die vier Queues durch Referenzierungen auf zuvor definierte Umgebungsvariablen im

[96] In enger Anlehnung an Serverless Framework Documentation (o. D. -f)
[97] in Anlehnung an DeBrie (2018a)

Queuename-Parameter namentlich voneinander abgesteckt. Die zwei zusätzlich erstellten Queues *AesDeadAudiofileQueue* und *AesDeadEvalScoreQueue* sollen als dead-letter Queues fungieren. Diese sollen Nachrichten der beiden Haupt-Queues *AesAudiofileQueue* bzw. *AesEvalScoreQueue* erhalten, falls diese nicht zu verarbeitende Nachrichten beinhalten. Mit solch einer Struktur werden die Haupt-Queues nicht durch feststeckende Nachrichten blockiert. Zudem dient dieses Gerüst dem Finden von Fehlern im Code sowie dem Testen des Quellcodes mit Nachrichten, die Fehler auslösen. Die SQS-Queue *DeadSqsAudioQueue* soll demnach die nicht zu verarbeitenden Nachrichten der SQS-Queue *AesAudiofileQueue* und die SQS-Queue *AesDeadEvalScoreQueue* die nicht zu verarbeitenden Nachrichten der SQS-Queue *AesEvalScoreQueue* sammeln. Der *MessageRetentionPeriod*-Parameter definiert in diesem Zusammenhang den Zeitraum in Sekunden, in welcher die Nachrichten innerhalb der dead-letter Queues gespeichert werden. Hier wurde jeweils ein Wert von 1209600 Sekunden - also zwei Wochen - gewählt, wodurch ausreichend viel Zeit verbleibt, um die Nachrichten innerhalb der dead-letter Queues auszulesen.[98] Eine Queue, die einer weiteren Queue als dead-letter Queue hinzugefügt werden soll, muss vor dieser Zuordnung bereits existieren. Somit muss der Service mit den in Abbildung 13 dargestellten Inhalten neu deployt[99] werden.[100]

Anschließend kann innerhalb der *RedrivePolicy* auf die jeweilige SQS-Queue referenziert werden, die als dead-letter Queues agieren sollen. Dafür muss innerhalb des *RedrivePolicy*-Parameters eine sogenannte ARN angegeben werden, welche AWS-Ressourcen mittels eines Namens eindeutig identifiziert. Um die ARN der jeweiligen dead-letter Queue zu erhalten, wird die Methode *FN:GetAtt* angewendet, welche den Namen und das gesuchte Attribut einer AWS-Ressource als Funktionsparameter benötigt. Diese Funktion wird als *deadLetterTargetArn* innerhalb der *RedrivePolicy* mit dem jeweiligen Namen der dead-letter Queues für die beiden SQS-Queues *AesEvalScoreQueue* und *AesAudiofileQueue* angegeben (siehe Abbildung 14).

[98] Vgl. Boto3 Documentation (o. D. -c)
[99] Dafür genügt das Eingeben des Befehlt *serverless deploy* in die CLI
[100] Vgl. AWS Documentation (o. D. -c), S. 15

```
resources:
  Resources:
    AesAudiofileQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.aesAudiofileQueue}
        RedrivePolicy:
          deadLetterTargetArn:
            "Fn::GetAtt":
              - DeadSqsAudioQueue
              - Arn
    DeadSqsAudioQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.deadSqsAudiofileQueue}
        MessageRetentionPeriod: 1209600
    AesEvalScoreQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.aesEvalScoreQueue}
        RedrivePolicy:
          deadLetterTargetArn:
            "Fn::GetAtt":
              - DeadEvalScoreQueue
              - Arn
    DeadEvalScoreQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: ${self:custom.sqs.deadSqsEvalScoreQueue}
        MessageRetentionPeriod: 1209600
```

**Abbildung 14: serverless.yml-Datei, Zuordnung der dead-letter Queues**

Bleiben Nachrichten innerhalb einer Queue stecken, da sie nicht verarbeitet werden können, so werden sie nach einem weiteren Deploy des Services an die dead-letter Queues weitergeleitet. Die gleiche Struktur wird auch für die SQS-Queue *aes-delete-audiofile-queue* angewendet, jedoch wird dafür die serverless.yml-Datei des MS-File um die Syntax einer SQS-Queue und einer dead-letter Queue erweitert sowie mit diesen Erweiterungen deployt.

### (2) DynamoDB-Tabelle

Um eine Ressource namens *aes-feedback-table* innerhalb der serverless.yml-Datei zu konfigurieren, wird dem *custom*-Parameter eine weitere Umgebungsvariable, deren

Wert dem Namen der Tabelle entspricht, hinzugefügt (siehe Abbildung 15).

```
feedbackTable: aes-feedback-table
```

**Abbildung 15: serverless.yml-Datei, Namensdeklarierung der feedback-Tabelle als Umgebungsvariable [101]**

Anschließend wird dem *Resources*-Parameter eine weitere Ressource namentlich hinzugefügt (*FeedbackTable*). Der daran anschließende *Type* der Ressource entspricht dem Wert *AES::DynamoDB::Table*, wodurch signalisiert wird, dass es sich bei der Ressource um eine DynamoDB-Tabelle handelt.

Die Eigenschaften der Tabelle werden innerhalb der *Properties* definiert. Dabei wird der Name der Tabelle durch eine Referenzierung auf die zuvor erstellte Umgebungsvariable im *TableName*-Parameter festgehalten. Der *AttributeDefinitions*-Parameter beinhaltet die Identifizierung des Primärschlüssels samt Datentyp-Bestimmung und steckt dabei ab, bei welchem Attribut es sich um den Partitionsschlüssel bzw. den Sortierschlüssel handelt. Der *KeyType Hash* bezeichnet dabei den Partitionsschlüssel der Tabelle, der *KeyType Range* wiederum den Sortierschlüssel. Zuletzt folgt die Angabe des *BillingMode*-Parameters. Erhält dieser den Wert *PAY_PER_REQUEST*[102], so wird die Lese- und Schreibkapazität der Tabelle automatisch geregelt. Da der Datenverkehr des neuen MS-AES noch nicht feststeht, wird diese Option zunächst bevorzugt. Die Kapazitäten der Tabelle werden dementsprechend automatisch erweitert, sollte ein hoher Datentransfer bestehen.[103] Zusammenfassend wird zum Erstellen der DynamoDB-Tabelle *aes-feedback-table* die serverless.yml-Datei des MS-AES um die in Abbildung 16 dargestellte Struktur erweitert.

[101] In enger Anlehnung an Serverless Framework Documentation (o. D. -f)
[102] Vgl. AWS Documentation (o. D. -d), S. 355
[103] Vgl. AWS Documentation (2012), S. 17ff., Vgl. DeBrie (2018b)

```
FeedbackTable:
  Type: "AWS::DynamoDB::Table"
  Properties:
    TableName: ${self:custom.feedbackTable}
    AttributeDefinitions:
      - AttributeName: pk
        AttributeType: S
      - AttributeName: sk
        AttributeType: S
    KeySchema:
      - AttributeName: pk
        KeyType: HASH
      - AttributeName: sk
        KeyType: RANGE
    BillingMode: PAY_PER_REQUEST
```

**Abbildung 16: serverless.yml-Datei, Erstellung der feedback-Tabelle[104]**

Der Primärschlüssel der Tabelle besteht somit aus den beiden Attributen *pk* und *sk*, welche beide als String definiert werden. *Pk* beschreibt dabei den Partitionsschlüssel und *sk* den Sortierschlüssel der Tabelle.

### (3) Das aes-audiofile-forwarder-Lambda

Innerhalb des MS-File soll das Lambda gemäß Kapitel 5.4.1 konzipiert werden. Dafür wird innerhalb des *functions*-Parameters eine neue Funktion namentlich angelegt (*aes_audiofile_forwarder)*. Innerhalb dessen wiederum wird ein *handler* definiert. Dieser verweist auf die *handler*-Funktion innerhalb der Python-Datei *aes_audiofile_forwarder.py*. Der sich darin befindende Quellcode wird ausgeführt, sobald ein Event das Lambda auslöst.[105] Dafür muss innerhalb der serverless.yml-Datei das eigentliche Event definiert werden. Da die Quelle des Events ein s3-Bucket ist, wird der *events*-Parameter als – *s3* definiert. Um das Lambda für jedes neue Element auszulösen, welches im *voice-cloning*-s3-Bucket hochgeladen wurde, wird der *bucket*-Parameter als *voice-cloning* und der *event*-Parameter als *s3:ObjectCreated:** definiert. Sollte es sich um einen Bucket handeln, welcher noch nicht als Ressource existiert, kann der *existing*-Parameter als *false* deklariert werden. Andernfalls wird dem Parameter der Wert *true* zugeordnet.[106]

---

[104] In Anlehnung an DeBrie (2018b)
[105] Siehe dazu stackoverflow (2020)
[106] Vgl. Serverless Framework Documentation (o. D. -a)

Rekapitulierend wird die *serverless.yml*-Datei um die Syntax der Abbildung 17 erweitert.

```
functions:
  aes_audiofile_forwarder:
    handler: aes_audiofile_forwarder.handler
    events:
      - s3:
          bucket: "voice-cloning"
          event: s3:ObjectCreated:*
          existing: true
```

**Abbildung 17: serverless.yml-Datei, Ergänzung um das aes-audiofile-forwarder-Lambda[107]**

Infolgedessen kann die Python-Datei *aes_audiofile_forwarder.py* erstellt und an der darin erhaltenden *handler*-Funktion gearbeitet werden. Diese Funktion erhält das Event, im JSON-Format sowie ein Context-Objekt - mit hier irrelevanten Informationen über den Lambda-Aufruf sowie über die Funktion und über die Laufzeitumgebung - .[108] Innerhalb des Handlers wird die *get_queue_url*-Funktion aufgerufen. Diese benötigt den Namen der SQS-Queue als Parameter, um die URL der SQS-Queue zurückzugeben. Falls dieser Prozess fehlschlägt, wird eine Exception geworfen und die Nachricht wird der SQS-Queue nicht hinzugefügt. Andernfalls wird der s3-Schlüssel des s3-Objekts aus dem Event extrahiert und die Nachricht mittels der *send_message*-Funktion und der Queue-URL der SQS-Queue im JSON Format angefügt.[109] Zuletzt folgt die *return*-Anweisung, welche das Lambda bzw. die Handler-Funktion der Lambdafunktion beendet.[110] Der vollständige Quellcode der Lambdafunktion des *aes_audiofile_forwarder*-Lambdas ist der Abbildung 18 zu entnehmen.

[107] In Anlehnung an Serverless Framework Documentation (o. D. -a)
[108] Vgl. AWS Documentation (o. D. -a), S. 267f.
[109] Siehe dazu Boto3 Documentation (o. D. -d)
[110] Vgl. AWS Documentation (o. D. -a), S. 268f.

```python
import json

import boto3

queue_name = "AesAudiofileQueue"
sqs = boto3.client("sqs")


def handler(event, context):
    try:
        response = sqs.get_queue_url(QueueName=queue_name)
        queue_url = response["QueueUrl"]
    except sqs.exceptions.QueueDoesNotExist as e:
        raise e

    key = event["Records"][0].get("s3").get("object").get("key").rstrip("/")

    response = sqs.send_message(
        QueueUrl=queue_url,
        MessageBody=json.dumps({"key": key}),
    )
    return True
```

**Abbildung 18: Quellcode der Lambdafunktion des aes-audiofile-forwarder-Lambdas[111]**

### (4) Das aes-audiofile-queue-forwarder-Lambda

Da das *aes-audiofile-queue-forwarder*-Lambda das erste Lambda des neuen MS-AES ist, wurde hierfür die serverless.yml-Datei des MS-AES um die in Abbildung 19 dargestellte Struktur erweitert.

```yaml
  aes_audiofile_queue_forwarder:
    handler: aes_audiofile_queue_forwarder.handler
    events:
      - sqs:
          arn:
            Fn::GetAtt:
              - AesAudioFileQueue
              - Arn
```

**Abbildung 19: serverless.yml-Datei, Ergänzung um das aes-audiofile-queue-forwarder-Lambda[112]**

Mit dieser Struktur wird das Aufrufen des Handlers für jede neue Nachricht innerhalb der SQS-Queue *AesAudioFileQueue* ermöglicht und automatisiert.[113]

---

[111] Eigener Quellcode
[112] In Anlehnung an Serverless Framework Documentation (o. D. -h)
[113] Vgl. DeBrie (2018b)

Auch hier wird anschließend die Python-Datei *aes_forwarder.py* samt der Handler-Funktion darin erstellt. Nachkommend werden drei unterschiedliche Funktionen definiert. Die erste Funktion (*create_new_item*) hat zur Aufgabe ein neues Element zu erstellen, welches der DynamoDB-Tabelle hinzugefügt werden kann. Die zweite Funktion (*send_to_aes*) soll den AES mit einer Liste von s3-Schlüssel anfragen. Die letzte Funktion (*update_keys*) hat die Aufgabe, die Elemente der DynamoDB-Tabelle zu updaten, deren Primärschlüssel zum AES gesendet wurden.

Der ersten Funktion wird das gesamte Event des Handlers als Parameter übergeben, um den Schlüssel des neuen s3-Objektes zu extrahieren und in einer Variablen zu speichern. Darauffolgend wird aus diesem Event der Organisationsname (*orgId*), der Name des Users (*userId*) als auch der Name der Audiodatei (*filename*) herausgefiltert. Als Resultat dieser Operation wird ein neues Dictionary (*new_file*) erstellt, welches die Attribute der DynamoDB-Tabelle beschreibt. Abschließend wird dieses Dictionary zurückgegeben, um von der Handler-Funktion weiter bearbeitet werden zu können. Zusammenfassend stellt die Abbildung 20 den Quellcode dieser *create_new_item*-Funktion dar.

```
def create_new_item(event):
    # get Variables out of sqs event
    key = json.loads(event.get("Records")[0].get("body")).get("key")
    key_values = key.rstrip("/").split("/")
    orgId, userId, filename = [key_values[i] for i in (0, 1, -1)]

    # create dict of new Item
    new_file = {
        "pk": f"orgId|{orgId}|userId|{userId}",
        "sk": f"filename|{filename}",
        "evaluationCode": {},
        "sentToFrontend": False,
        "sentToCluster": False,
        "deleted": False
    }

    return new_file
```

**Abbildung 20: Quellcode der create_new_file_item-Funktion[114]**

Die nächste Funktion erwartet eine Liste von s3-Schlüssel, welche an den AES weitergeleitet werden sollen. Da der AES eine HTTP-POST-Request als Anfrage erwartet,

[114] Eigener Quellcode

muss sowohl ein Header als auch ein Body definiert werden. Die s3-Schlüssel, welche an den AES gesendet werden sollen, befinden sich im JSON-Format, weshalb der Header der Request mit dem *Content-Type application/json* versehen wird. Die Liste an s3-Schlüsseln wird innerhalb des *json*-Parameters als Value einem Dictionary mit dem Schlüssel *key* hinzugefügt. Wird diese Request an den AES gesendet, so wird gemäß Kapitel 2.1 eine Response erwartet. Jedoch ist die Laufzeit des AES abhängig von der Menge und Größe der einzelnen Audiodateien. Somit könnte aufgrund einer zu hohen Wartezeit eine Timeout-Exception seitens dieser Funktion geworfen werden.[115] Um dieses Szenario zu vermeiden, wird der *request.post*-Funktion ein *timout*-Parameter mitgegeben, welcher den Wert von 0,2 Sekunden erhält. Dadurch wird lediglich 0,2 Sekunden auf eine Antwort des AES gewartet. Die daran anschließende Timeout-Exception wird mittels des *pass*-Arguments im *except*-Block abgefangen. Zuletzt folgt auch die *return*-Anweisung, durch welche die Funktion beendet wird (siehe Abbildung 21).

```
def send_to_aes(keys):
    headers = {
        "Content-Type": "application/json",
    }
    try:
        re-sponse = requests.post(
        "http://aes.aes.k8.aflr.io/eval",
        headers=headers,
        json={"keys": keys},
        timeout=0.2)
    except requests.exceptions.ReadTimeout:
        pass
    return True
```

**Abbildung 21: Quellcode der sent_to_aes-Funktion[116]**

Die letzte Funktion erhält ebenfalls eine Liste an s3-Schlüsseln. Jedoch gilt es, das *sentToAes*-Attribut des korrespondierenden Elements der DynamoDB-Tabelle *aes-feedback-table* upzudaten. Dafür wird für jeden s3-Schlüssel anhand des Organisationsnamens (*orgId*), des Usernamens (*userId*) sowie des Dateinamens (*filename*) der Primärschlüssel gebildet. Daran anschließend aktualisiert die *update_item*-Funktion das entsprechende Element der DynamoDB-Tabelle mittels des Primärschlüssels und

[115] Vgl. Python Documentation (o. D.)
[116] Eigener Quellcode

deklariert das *setToAES*-Attribut als *True*. Die *return*-Anweisung folgt zuletzt und beendet die *update_keys*-Funktion (siehe Abbildung 22).

```
def update_keys(keys):
    for key in keys:
        key_values = key.rstrip("/").split("/")
        table.update_item(
            Key={
                "pk": f"orgId|{key_values[0]}|userId|{key_values[1]}",
                "sk": f"filename|{key_values[-1]}",
            },
            UpdateExpression="set sentToAes = :bool",
            ExpressionAttributeValues={":bool": True,},
        )
    return True
```

**Abbildung 22: Quellcode der update_keys-Funktion[117]**

Um die Funktionalität des aes-audiofile-queue-forwarder-Lambdas zu erfüllen, werden innerhalb der Handler-Funktion alle Funktionen nacheinander aufgerufen. Die *create_new_file_item*-Funktion erhält das Event als Funktionsparameter, erstellt ein Dictionary, welches der Struktur der Elemente in der DynamoDB-Tabelle entspricht und gibt dieses zurück. Der Partitionsschlüssel dieses Elements wird anschließend zwecks einer Abfrage der DynamoDB-Tabelle *aes-feedback-table* verwendet, wobei nach Elementen mit gleichem Partitionsschlüssel gesucht wird, deren *sentToAes*-Attribut dem Wert *False* entspricht. Diese Operation wird mittels der *query*-Funktion durchgeführt. Die Antwort der Abfrage beinhaltet unter anderem einen *Count*-Parameter, welcher die Anzahl gefundener Elemente in der Tabelle beschreibt. Entspricht dieser Parameter dem Wert *9*, so befinden sich zehn s3-Schlüssel nicht evaluierter Audiodateien in der Handler-Funktion. In diesem Fall wird der Schlüssel des neu erstellten Elements einer Liste hinzugefügt, welcher die neun zurückgegebenen Schlüssel der Query-Operation hinzuaddiert werden. Mit dieser Liste als Funktionsparameter wird im Anschluss daran sowohl die *send_to_aes*-Funktion als auch die *update_keys*-Funktion aufgerufen, um die Schlüssel zum AES zu schicken und anschließend das *sentToAes*-Attribut upzudaten. Vollständigkeitshalber muss vor dem *return*-Statement noch das neue Element (*new_file*) der DynamoDB-Tabelle hinzugefügt werden, wobei der *sentToAES*-Parameter ebenfalls auf *True* gesetzt wird, sollte eine Anfrage zum

[117] Eigener Quellcode

AES erstellt worden sein. Die Lambdafunktion des *aes-audiofile-queue-forwarder*-Lambdas wird in der Abbildung 23 zusammengefasst.

```python
import json
import requests

import boto3
from boto3.dynamodb.conditions import Key

database = boto3.resource("dynamodb")
table = "feedback-table"

def handler(event, context):
    try:
        new_file = create_new_file_item(event)
    except Exception as e:
        print(e)
        return
    try:
        resp = table.query(
            KeyConditionExpression=Key("pk").eq(new_file.get("pk"))
            & Key("sk").begins_with("filename|"),
            FilterExpression="sentToAes = :sentToAes",
            ExpressionAttributeValues={":sentToAes": False}
        )
        items = resp.get("Count")
    except Exception as e:
        print(e)
        return

    # if 9 files + new one -> forward to AES
    if resp.get("Count") == 9:
        keys = list()
        keys.append(new_file.get("key"))

        # append keys with 9 items
        for item in resp.get("Items"):
            keys.append(item["key"])
        try:
            send_to_cluster(keys)
            update_keys(keys)
        except Exception as e:
            print(e)
            return
        new_file["sentToAes"] = True

    response = table.put_item(Item=new_file)
    return
```

**Abbildung 23: Quellcode der Lambdafunktion des aes-audiofile-queue-forwarder-Lambdas[118]**

[118] Eigener Quellcode

### (5) Das aes-feedback-interpreter-Lambda

Auch das *aes-feedback-interpreter*-Lambda wird von jeder neuen Nachricht innerhalb einer SQS-Queue aufgerufen. Demnach ähnelt sich die Erweiterung der serverless.yml-Datei (siehe Abbildung 24) stark Ergänzung des *aes-audiofile-queue-forwarder*-Lambdas in der Abbildung 19.

```
aes_feedback_interpreter:
    handler: aes_feedback_interpreter.handler
    events:
      - sqs:
          arn:
            Fn::GetAtt:
              - AesEvalScoreQueue
              - Arn
```

**Abbildung 24: serverless.yml-Datei, Ergänzung um das aes-feedback-interpreter-Lambda**[119]

Dieses Lambda wird jedoch von der SQS-Queue *AesEvalScoreQueue* ausgelöst und führt die Handler-Funktion der Datei *aes_feedback_interpreter.py* aus. Diese erhält die Evaluierungsergebnisse des AES und ermittelt zunächst die URL der SQS-Queue *AesDeleteAudiofileQueue*, um dieser Queue Nachrichten hinzufügen zu können. Auch hier wird eine Exception geworfen, falls die Queue nicht existieren sollte. Um die Evaluierungsergebnisse des AES zu interpretieren, wird die Liste an Ergebnissen auf dem eingehenden Event extrahiert und in einer Variablen gespeichert (*scores*). Anschließend wird über jedes Ergebnis mithilfe einer Schleife iteriert, wobei das Ergebnis auf den *invalid*-Parameter überprüft wird. Entspricht dessen Wert *True* so wird der Schlüssel dieses s3-Objekts an die SQS-Queue weitergeleitet und der automatische Löschprozess beginnt (siehe Abbildung 25).

[119] In Anlehnung an Serverless Framework Documentation (o. D. -h)

```python
import os
import json

import boto3

database = boto3.resource("dynamodb")
table = "feedbackTable"
sqs = boto3.client("sqs")
queue_name = "AesDeleteAudiofileQueue"


def handler(event, context):
    try:
        response = sqs.get_queue_url(QueueName=queue_name)
        queue_url = response["QueueUrl"]
    except sqs.exceptions.QueueDoesNotExist as e:
        raise e

    scores = json.loads(event["Records"][0]["body"])["scores"]

    for dic in scores:
        if dic["invalid"]: #if that is True
            response = sqs.send_message(
                QueueUrl=queue_url,
                MessageBody=json.dumps({"key": dic["Key"]}),
            )
    try:
        update_table(scores)
    except Exception as e:
        print(e)
        return
```

**Abbildung 25: Quellcode der Lambdafunktion des aes-feedback-interpreter-Lambdas[120]**

Wie im unteren Try-Block der Abbildung 25 zu erkennen, ist eine weitere Aufgabe des Lambdas das Updaten der überprüfen Elemente in der DynamoDB-Tabelle, wobei das *evaluationCode*-Attribut mit dem jeweiligen Evaluierungsergebnis in Form eines Dictionary upgedatet wird. Für diesen Zweck wurde die *update_table*-Funktion konzipiert, welche von der Handler-Funktion aufgerufen wird und die Liste an Evaluierungsergebnissen als Funktionsparameter erhält. Innerhalb der Funktion wird anschließend mit einer Schleife über jedes einzelne Evaluierungsergebnis iteriert, wobei anhand des Evaluierungsergebnisses interpretiert wird, ob eine ungültige Datei vorliegt (invalid = True). Anschließend wird der jeweilige Primärschlüssel gebildet und das

[120] Eigener Quellcode

Evaluierungsergebnis des jeweiligen Elements herausgesucht. Nach dem Updaten der beiden Attribute *evaluationCode* und *deleted* für jedes Element innerhalb des Evaluationsergebnisses, folgt die *return*-Anweisung, welche die Funktion *update_table* beendet (siehe Abbildung 26). Falls dieser Prozess des Aktualisierens jedoch fehlschlägt, wird innerhalb der Handler-Funktion eine weitere Exception geworfen. Andernfalls beendet das *return*-Argument die Lambdafunktion (siehe Abbildung 25). Der vollständige Code der *update-table*-Funktion wird in Abbildung 26 visualisiert.

```
def update_table(scores):
    for item in scores:
        invalid = True if item["invalid"] else False
        key_values = item.get("key").rstrip("/").split("/")
        table.update_item(
             Key={
                "pk": f"orgId|{key_values[0]}|userId|{key_values[1]}",
                "sk": f"filename|{key_values[-1]}",
            },
            UpdateExpression="SET evaluationCode = :dict, deleted = :invalid",
            ExpressionAttributeValues={
                        ":dict": {"score": scores["item"]},
                        ":invalid":invalid
                    }
        )
    return True
```

**Abbildung 26: Quellcode der update_table-Funktion[121]**

### (6) Das aes-audiofile-deleter-Lambda

Ähnlich den bisher vorgestellten Lambdas wird auch dieses mithilfe einer SQS-Queue ausgelöst. Dafür wird die serverless.yml-Datei des MS-File erweitert, da sich die zu löschenden Audiodateien innerhalb dieses MS befinden. Die nachfolgende Abbildung 27 veranschaulicht die Struktur, die der serverless.yml-Datei des MS-File hinzugefügt wurde.

[121] Eigener Quellcode

```
aes_audiofile_deleter:
    handler: aes_audiofile_deleter.handler
    events:
      - sqs:
          arn:
            Fn::GetAtt:
              - AesDeleteAudiofileQueue
              - Arn
```

**Abbildung 27: serverless.yml-Datei, Ergänzung um das aes-audiofile-deleter-Lambda**[122]

Nach dem Erstellen der Python-Datei *aes_audiofile*_deleter.py sowie der *handler*-Funktion ist die erste Aufgabe des Lambdas das Extrahieren des s3-Schlüssels, welcher sich im eingehenden Event befindet. Da der übergebene s3-Schlüssel auf eine Audiodatei innerhalb des s3-Buckets *voice-cloning* referenziert, wird dieser innerhalb der *delete_object*-Funktion als Wert des Funktionsparameters *Key* mitgegeben. Der Bucketname wird der Funktion mittels dem Funktionsparameter *Bucket* und dem Wert *voice-cloning* übergeben. Die *delete_object*-Funktion wird dabei von einem *try*-Block umschlossen, um im Falle eines Fehlschlags der Operation das Crashen des Handlers mit dem *except*-Block abzufangen. Die Handler-Funktion wird anschließend mit der *return*-Anweisung beendet, wodurch auch das Lambda beendet wird, wie der Abbildung 28 zu entnehmen ist.

[122] In Anlehnung an Serverless Framework Documentation (o. D. -h)

```python
import os
import json

import boto3


s3 = boto3.client("s3")


def handler(event, context):
    key = json.loads(event.get("Records")[0].get("body")).get("key")
    try:
        response = s3.delete_object(
            Bucket="voice-cloning",
            Key=key,
        )
    except Exception as e:
        return e

    return True
```

**Abbildung 28: Quellcode der Lambdafunktion des aes-audiofile-deleter-Lambdas[123]**

### (7) Das aes-feedback-provider-Lambda

Im Gegensatz zu den bisher implementierten Lambdas, wird dieses mittels einer HTTP-Request aufgerufen. Somit wird innerhalb der *serverless.yml*-Datei des MS-AES der *events*-Parameter des Lambdas als - *http:* definiert. Da das Frontend der Voice Cloning App die Daten erhalten soll, ist die Aufrufmethode *get*. Der Pfad auf welchem das Lambda eine GET-Request erwartet kann frei gewählt werden (siehe Abbildung 29).

```yaml
aes_feedback_provider:
  handler: aes_feedback_provider.handler
  timeout: 30
  events:
    - http:
        path: /aes/feedback
        method: get
```

**Abbildung 29: serverless.yml-Datei, Ergänzung um das aes-feedback-provider-Lambda[124]**

Die Handler-Funktion - welche der *feedback_provider.py*-Datei hinzugefügt wird - soll die DynamoDB-Tabelle nach Inhalten abfragen. Dabei soll die Tabelle nur Daten

[123] Eigener Quellcode
[124] In Anlehnung an Serverless Framework Documentation (o. D. -h)

zurücksenden, welche dem jeweiligen User noch nicht dargestellt wurden (sentToFrontend=False). Zwecks dieser Abfrage muss zunächst der Primärschlüssel definiert werden. Somit wird der Organisationsname (*orgId*) mittels des Lambda-Decorators *@get_org_id* sowie der Username aus dem URL-Parameter (*userId*) ermittelt. Für die daran anschließende Datenbankabfrage wird zusätzlich bestimmt, dass der Sortierschlüssel mit dem String *filename|* beginnt (*begins_with*), um alle Audiodateien zurückzuerhalten, die dem User zuordenbar sind. Des Weiteren werden nur Elemente zurückgegeben, deren *sentToFrontend*-Attribut *False* entspricht. Auch das *evalutionCode*-Attribut soll ungleich *empty* sein, um nur die Ergebnisse bereits evaluierter Audiodateien an das Frontend zurückzuschicken. Diese Bedingungen wurden innerhalb der *FilterExpression* formuliert.

Die erhaltenden Elemente der Tabelle werden schließlich innerhalb einer Schleife upgedatet, wobei *das sentToFrontend*-Attribut auf *True* gesetzt wird. Zusätzlich wird der *evaluationCode* jedes einzelnen Elements einer der *feedback*-Liste hinzugefügt, welche schlussendlich als Response samt dem *statusCode 200* an das Frontend zurückgeschickt wird. Der Abbildung 30 ist der gesamte Quellcode der Lambdafunktion *aes-feedback-provider* zu entnehmen.

```
import boto3
from boto3.dynamodb.conditions import Key
from aflr_decorators import get_org_id
from lambda_decorators import dump_json_body

database = boto3.resource("dynamodb")
table = database.Table("feedback-table")

@get_org_id
@dump_json_body
def handler(event, context):
    orgId = event["orgId"]
    try:
        userId = event["queryStringParameters"]["userId"]
    except Exception as e:
        return {
            "statusCode": 400,
            "body": {
                "message": "Error. Please specify a userId."
            },
        }
    try:
        response = table.query(
            KeyConditionExpres-sion=Key("pk").eq(f"orgId|{or-
gId}|userId|{userId}")
            & Key("sk").begins_with("filename|"),
            FilterExpression="sentToFrontend = :FilterExpression and evalua-
tionCode <> :evaluationCode"
            ExpressionAttributeValues={":FilterExpression": False,
                                       ":evaluationCode": "empty"}
        )
    except Exception as e:
        return {
            "statusCode": 400,
            "body": {"message": "Error. Could not connect to DynamoDB."}
        }
    feedback = dict()
    for item in response.get("Items"):
        resp = table.update_item(
            Key={"pk": item.get("pk"), "sk": item.get("sk"),},
            UpdateExpression="set sentToFrontend = :bool",
            ExpressionAttributeValues={":bool": True},
        )
        feedback[item.get("sk").replace("filename|", "")] = {
            "evaluationCode": item.get("evaluationCode").get("score")
        }
    return {
        "statusCode": 200,
        "body": {"newAudioFeedback": feedback},
        }
```

**Abbildung 30: Quellcode der Lambdafunktion des aes-feedback-provider-Lambdas[125]**

# 6 Schlussbetrachtung

Das Ziel dieser Bachelorthesis war die Modellierung und Implementierung eines Microservices, der eine automatisierte Darstellung von Userfeedback in Aflorithmics Voice Cloning App, mittels eines Audio Feedback Systems, ermöglicht.

Innerhalb des Grundlagen-Kapitels wurde auf das HTTP (2.1), auf ausgewählte Architekturansätze (2.2) sowie auf einige grundlegende AWS Services (2.3) eingegangen, welche den Modellierungs- und Implementierungsprozess theoretisch vorbereiteten. Das dritte Kapitel erläuterte das Unternehmensmodell Aflorithmics und ordnete die Voice Cloning App in selbiges ein. Das vierte Kapitel erläuterte, nach einer kurzen Vorstellung der Voice Cloning App in 4.1, einige ausgewählte Prozesse innerhalb der App. Neben dem Registrierungs- und Login-Prozess (4.2) wurde die Darstellung der vorzulesenden Texte (4.3) sowie das Hochladen und die Verwaltung der Audiodateien (4.4) veranschaulicht. Daran anknüpfend wurde sich innerhalb des fünften Kapitels mit dem Audio-Evaluierungs-Service (5.1) auseinandergesetzt. Angesichts der in Kapitel 5.2 dargestellten Anforderungen an den Microservice wurde ebendieser in Kapitel 5.3 schrittweise modelliert und das erstellte Modell anschließend implementiert (5.4).

Nach Abschluss der Implementierung und einem letzten Deploy der Microservices MS-AES und MS-File steht das Backend der Voice Cloning App für den automatisierten Evaluierungsprozess bereit. Alle Audioaufnahmen werden innerhalb des MS-AES registriert und einmalig evaluiert. Im Falle einer ungültigen Audiodatei wird ein Löschvorgang für die dementsprechende Audiodatei aktiviert. Zudem steht eine URL bereit, die angefragt werden kann, um die Evaluierungsergebnisse eines bestimmten Users zu erfragen. Diese URL muss innerhalb des Frontends lediglich einer neuen HTTP-Request hinzugefügt werden und bietet die Ermöglichung der clientseitigen Darstellung der Evaluierungsergebnisse an. Backendseitig wurde der MS-AES als neuer Microservice implementiert sowie der MS-File um einige AWS Ressourcen und damit einhergehende neue Funktionalitäten erweitert. Für die bereits bestehende Backend-Infrastruktur waren keinerlei Codeänderungen vonnöten.

[125] Eigener Quellcode

Die Verwendung der REST-API hat sich im Rahmen dieser Bachelorthesis als schnelle und einfache Möglichkeit der Modellierung und Implementierung erwiesen. Jedoch könnte diese API zukünftig durch die Verwendung der WebSocket API ersetzt werden. Der Vorteil würde darin liegen, dass seitens des Frontends keine HTTP-Request an das Backend geschickt werden muss, um die Evaluierungsergebnisse zu erhalten. Stattdessen wird eine direkte Verbindung zwischen dem Frontend und dem Backend aufgebaut. Diese kann verwendet werden, um Nachrichten, wie die Evaluierungsergebnisse, direkt zwischen Client und Server auszutauschen. Um eine solche Verbindung aufzubauen, müsste das Frontend jedoch auch weitgehender modifiziert werden.[126]
Zudem sollte die Lese- und Schreibkapazität der DynamoDB-Tabelle *feedback-table* festgelegt werden, um die Kosten der in Kapitel 5.4.2 beschriebenen automatischen Verwaltung der Tabellen-Kapazitäten zu sparen. Dafür wird innerhalb der serverless.yml-Datei der *BillingMode*-Parameter der DynamoDB-Tabelle durch den *ProvisionedThroughput*-Parameter ersetzt, wie De Brie veranschaulicht.[127]

Abschließend kann festgehalten werden, dass der neue Microservice aufgrund einer logischen und schrittweisen Modellierung unter der Berücksichtigung der gegebenen Anforderungen problemlos in die zuvor existierende Infrastruktur implementiert werden konnte. Auch nach der Laufzeit von mehreren Monaten mussten keine weiteren Änderungen vorgenommen oder Bugs beseitigt werden.

[126] Vgl. AWS Documentation (o. D. -e), S. 690
[127] Vgl. DeBrie (2018b)

**Literaturverzeichnis**

**Aflorithmic (o. D. -a)**: Voice Cloning, URL: https://www.aflorithmic.ai/solutions#voice-cloning (letzter Abruf am 09.09.2021)

**Aflorithmic (o. D. -b)**: Pricing, URL: https://www.aflorithmic.ai/pricing (letzter Abruf am 21.09.2021)

**Api.Audio Documentation (o. D. -a)**: Introduction, URL: https://docs.api.audio/docs (letzter Abruf am 21.09.2021)

**Api.Audio Documentation (o. D. -b)**: The basics, URL: https://docs.api.audio/docs/the-basics (letzter Abruf am 21.09.2021)

**AWS (2020a):** Amazon Polly launches Brand Voice, *About AWS, What's New Feed,* URL: https://aws.amazon.com/about-aws/whats-new/2020/02/amazon-polly-launches-brand-voice/ (letzter Abruf am 08.09.2021)

**AWS (2020b):** Build a unique Brand Voice with Amazon Polly, *Blog, machine learning*, URL: https://aws.amazon.com/blogs/machine-learning/build-a-unique-brand-voice-with-amazon-polly/ (letzter Abruf am 09.09.2021)

**AWS (2020c)**: Amazon Polly Overview, URL: https://aws.amazon.com/polly/ (letzter Abruf am 10.09.2021)

**AWS Documentation (2006)**: Amazon Simple Storage Service: User Guide, URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-userguide.pdf (letzter Abruf am 17.09.2021)

**AWS Documentation (2010)**: AWS CloudFormation: Benutzerhandbuch, URL: https://docs.aws.amazon.com/de_de/AWSCloudFormation/latest/UserGuide/cfn-ug.pdf#page=266&zoom=100,96,841 (letzter Abruf am 21.09.2021)

**AWS Documentation (2012)**: Amazon DynamoDB: Developer Guide, URL: https://docs.aws.amazon.com/en_us/amazondynamodb/latest/developerguide/dynamodb-dg.pdf (letzter Abruf am 17.09.2021)

**AWS Documentation (o. D. -a)**: Lambda: Developer Guide, URL: https://docs.aws.amazon.com/us_en/lambda/latest/dg/lambda-dg.pdf (letzter Abruf am 05.10.2021)

**AWS Documentation (o. D. -b)**: Working with Queries in DynamoDB, URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html#FilteringResults (letzter Abruf am 19.10.2021)

**AWS Documentation (o. D. -c)**: Amazon Simple Queue Service: Developer Guide , URL: https://docs.aws.amazon.com/us_en/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dg.pdf#sqs-configure-dead-letter-queue (letzter Abruf am 05.10.2021)

**AWS Documentation (o. D. -d)**: Amazon DynamoDB: API Reference, URL: https://docs.aws.amazon.com/de_de/amazondynamodb/latest/APIReference/dynamodb-api.pdf (letzter Abruf am 05.10.2021)

**AWS Documentation (o. D. -e)**: Amazon API Gateway Developer Guide, URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-dg.pdf (letzter Abruf am 07.11.2021)

**Bedford-Strohm (2017)**: Voice First? Eine Analyse des Potentials von intelligenten Sprachassistenten am Beispiel Amazon Alexa *Cummunicatio Socialis*, 50. Aufl., Heft 4, S. 485-494

**Boto3 Documentation (o. D. -a)**: Boto3 Documentation, URL: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html (letzter Abruf am 14.09.2021)

**Boto3 Documentation (o. D. -b)**: DynamoDB, URL: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html (letzter Abruf am 17.09.2021)

**Boto3 Documentation (o. D. -c)**: SQS, create_queue(**kwargs), URL: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sqs.html#SQS.Client.create_queue (letzter Abruf am 05.10.2021)

**Boto3 Documentation (o. D. -d)**: A sample tutorial, URL: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html (letzter Abruf am 05.10.2021)

**Bouré (2021)**: Understanding the DynamoDB Sort Key Order, URL: https://benoitboure.com/understanding-the-dynamodb-sort-key-order (letzter Abruf am 17.09.2021)

**Chaudhry (2017)**: What is an API?, URL: https://medium.com/@aditi.chaudhry92/what-is-an-api-234e949af15c (letzter Abruf am 10.09.2021)

**Chen/Tan/Li/Liu/Qin/Zhae/Liu (2021)**: AdaSpeech: Adaptive Text to Speech for Custom Voice, URL: https://arxiv.org/abs/2103.00993v1 (letzter Abruf am 06.09.2021)

**Codecademy (o. D.)**: What is REST?, URL: https://www.codecademy.com/articles/what-is-rest (letzter Abruf am 10.09.2021)

**Coyle (o. D.)**: API.audio Is Here!, URL: https://www.aflorithmic.ai/post/api-audio-is-here (letzter Abruf am 21.09.2021)

**De Alwis/Barros/Fidge/Polyvyanyy (2019)**: Availability and Scalability Optimized Microservice Discovery from Enterprise Systems, *On the Move to Meaningful Internet Systems,* Lecture Notes in Computer Science 11877, S. 496-514, DOI: https://doi.org/10.1007/978-3-030-33246-4

**DeBrie (2018a)**: Using SQS with AWS Lambda and Serverless, URL: https://www.serverless.com/blog/aws-lambda-sqs-serverless-integration (letzter Abruf am 12.10.2021)

**DeBrie (2018b)**: DynamoDB On-Demand: When, why and how to use it in your serverless applications, URL: https://www.serverless.com/blog/dynamodb-on-demand-serverless (letzter Abruf am 05.10.2021)

**Desai (2021)**: What is API explained in easy way?, URL: https://dev.to/vrajdesai78/what-is-api-explained-in-easy-way-5aih (letzter Abruf 10.09.2021)

**Drilling/Augsten (2017)**: Konzept, Aufbau und Funktionsweisen von REST, URL: https://www.dev-insider.de/konzept-aufbau-und-funktionsweise-von-rest-a-603152/ (letzter Abruf am 10.09.2021)

**Dutoit (1997):** An Introduction to Text-to-Speech Synthesis, URL: https://d1wqtxts1xzle7.cloudfront.net/3436743/12-Short-Intro-to-TTS-with-cover-page-v2.pdf?Expires=1631202421&Signature=NZPCNDO4wKIl-rEeRVp62IL9DqNFe4hRtmUqvGkf7z-tHkMcB-COxPhs8fu1XvU3YfIAz2DJUe-wOakZluxTv6KIM0bnBTrn7eC~xi84W~QEfC2ymThSogUPFpLu-ZQkT3BumUEQyeO2Sj6uVrC2GwvaAh2YwUHuPuhM-Cabb52OMF4uAYk8CVr1nUtBx1IGmpGIi0AP-FhUKjOuG9AK4t1vyXLI7xd8rpZCrCSe7joJ6KjerDkTZ0ioY6qc8VD~G-nw0N8nf63qsG-4O0vS0S19ie-OvuemvsU982S9hn4HQtdX4Gci7VzK2ST5tfThEgSHXYCcJN-RA-s6MMVq9tK1ZQQQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA (letzter Abruf am 09.09.2021)

**Ekblom (2011)**: Applied Representational State Transfer, URL: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A393777&dswid=-9366 (letzer Abruf am 10.09.2021)

**Fielding (2000)**: Architectural Styles and the Design of Network-based Software Architectures, URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (letzter Abruf am 10.09.2021)

**Fielding/Irvine/Gettys/Mogul/Frystyk/Leach/Masinter/Berners-Lee (1999)**: Hypertext Transfer Protocol -- HTTP/1.1, URL: https://dl.acm.org/doi/pdf/10.17487/RFC2616 (letzter Abruf am 11.09.2021)

**Google Support (o. D.)**: URL-Parameter, URL: https://support.google.com/google-ads/answer/6277564?hl=de (letzter Abruf am 12.09.2021)

**Hörner (2019)**: Marketing mit Sprachassistenten, DOI: https://doi.org/10.1007/978-3-658-25650-0

**Juniper Research (2020)**: Number of digital voice assistants in use worldwide 2019-2024. Zitiert nach de.statista.com, URL: https://www.statista.com/statistics/973815/worldwide-digital-voice-assistant-in-use/ (letzter Abruf am 07.09.2021)

**Kandwal (2020)**: REST API Tutorial – REST Client, REST Service, and API Calls Explained With Code Examples, URL: https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/ (letzter Abruf am 11.09.2021)

**Lewis/Fowler (2014)**: Microservices, URL: https://www.martinfowler.com/articles/microservices.html#footnote-etymology (letzter Abruf am 12.09.2021)

**Mathew/Varia (2014)**: Overview of Amazon Web Services, URL: https://www.sysfore.com/Assets/PDF/aws-overview.pdf (letzter Abruf am 13.09.2021)

**Microsoft Azure Documentation (2021a)**: What is Custom Neural Voice, *Cognitive Services, Speech Service* URL: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/custom-neural-voice (letzter Abruf am 08.09.2021)

**Microsoft Azure Documentation (2021b)**: Get started with Custom Neural Voice, *Cognitive Services, Speech Services,* URL: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-custom-voice (letzter Abruf am 09.09.2021)

**Microsoft Azure Documentation (2020)**: How to record voice samples, *Cognitive Services, Speech Service,* URL: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/record-custom-voice-samples (letzter Abruf am 09.09.2021)

**Monperrus/Eichberg/Tekes/Mezini (2012)**: What should developers be aware of? An empirical study on the directives of API documentation, DOI: 10.1007/s10664-011-9186-4

**Postman Blog (2020)**: Intro to APIs: What Is an API?, URL: https://blog.postman.com/intro-to-apis-what-is-an-api/ (letzer Abruf am 09.09.2021)

**Python Documentation (o. D.)**: Quickstart, URL: https://docs.python-requests.org/en/latest/user/quickstart/#errors-and-exceptions (letzter Abruf am 05.10.2021)

**Robillard/Chhetri (2014)**: Recommending reference API documentation, DOI: 10.1007/s10664-014-9323-y

**Sisman/Yamagishi/King/Li (2020)**: An Overview of Voice Conversion and its Challenges: From Statistical Modeling to Deep Learning, URL: https://arxiv.org/abs/2008.03648 (letzter Abruf am 21.09.2021)

**stackoverflow (2020):** In general terms: what is the difference between a handler and a function, URL: https://stackoverflow.com/questions/44281644/in-general-terms-what-is-the-difference-between-a-handler-and-a-function (letzter Abruf am 05.10.2021)

**Supreme court of the United States (2020)**: Syllabus, Google LLC v. Oracle America, inc., URL: https://www.supremecourt.gov/opinions/20pdf/18-956_d18f.pdf (letzter Abruf am 21.09.2021)

**Serverless Framework Documentation (o. D. -a)**: s3, URL. https://www.serverless.com/framework/docs/providers/aws/events/s3 (letzter Abruf am 05.10.2021)

**Serverless Framework Documentation (o. D. -b)**: Serverless Framework Documentation, URL: https://www.serverless.com/framework/docs (letzter Abruf am 05.10.2021)

**Serverless Framework Documentation (o. D. -c)**: Credentials, URL: https://www.serverless.com/framework/docs/providers/aws/guide/credentials/ (letzter Abruf am 05.10.2021)

**Serverless Framework Documentation (o. D. -d)**: Services, URL: https://www.serverless.com/framework/docs/providers/aws/guide/services (letzter Abruf am 05.10.2021)

**Serverless Framework Documentation (o. D. -e)**: deploy, URL: https://www.serverless.com/framework/docs/providers/aws/cli-reference/deploy (letzter Abruf am 05.10.2021)

**Serverless Framework Documentation (o. D. -f)**: Variables, URL: https://www.serverless.com/framework/docs/providers/aws/guide/variables (letzter Abruf am 05.10.2021)

**Serverless Framework Documentation (o. D. -g)**: SNS, URL: https://www.serverless.com/framework/docs/providers/aws/events/sns (letzter Abruf am 12.10.2021)

**Serverless Framework Documentation (o. D. -h)**: SQS Queues, URL: https://www.serverless.com/framework/docs/providers/aws/events/sqs (letzter Abruf am 12.10.2021)

**Tan/Qin/Soong/Liu (2021)**: A Survey on Neural Speech Synthesis, URL: https://arxiv.org/abs/2106.15561v3 (letzter Abruf am 07.09.2021)

**Wang/Keivanloo/Zou (2014)**: How Do Developers React to RESTful API Evolution?, *Service-Oriented Computing*, Lecture Notes in Computer Science 8831, S. 245-259, DOI: 10.1007/978-3-662-45391-9

**Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis mit dem Titel

„Microservice-Modellierung und -Implementierung. Darstellungsermöglichung von Userfeedback in Aflorithmics Voice Cloning App“

selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

_____

Trier, 30. November 2021