Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Bachelorarbeit

Studienrichtung

Wirtschaftsinformatik

Simon Mayer
Digitalisierung der Vereinsorganisation mithilfe von Cloud Computing

Erstprüfer: Prof. Dr. Hubert Högl

Zweitprüfer: Prof. Dr. Phillip Heidegger

Abgabe der Arbeit am: 21.03.2022

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Verfasser der Bachelorarbeit:
Simon Mayer
Ravenspurgerstraße 34
86150 Augsburg
Telefon: +49 152 04814558
mailto.smayer@googlemail.com

Abstract

Die Entwicklung von Anwendungen unter Einsatz von Serverless Technologie wie Function as a Service stellt einen neuen Ansatz innerhalb der Softwareentwicklung dar und bietet neue Möglichkeiten, aber auch Herausforderungen. Die vorliegende Bachelorarbeit fokussiert sich auf diesen Bereich des Cloud Computing, indem eine Webanwendung in Form eines Vereinsportals entwickelt wird. Es bietet Vereinen u.a. eine zentrale Benutzerverwaltung und ein Modul für die Durchführung von Terminabstimmungen innerhalb der Vereinsorganisation.

Technisch wird diese praxisnahe Beispielanwendung im Frontend durch eine Angular Applikation umgesetzt. Die Benutzerverwaltung wird darin integriert, indem auf den Benutzerverwaltungsdienst von Amazon Web Services zugegriffen wird. Auch die Businesslogik für das Umfragemodul basiert auf einer Serverless Architektur bestehend aus AWS Lambda, API Gateway, DynamoDB, S3 etc. des gleichen Cloud Providers.

Neben der Beschreibung dieses Setups betrachtet die Arbeit grundlegende Schritte, um mit der Entwicklung von Anwendungen auf Grundlage von Cloud Technologien Fuß fassen zu können. So nimmt der Dienst AWS Identity and Access Management eine Schlüsselrolle ein und wird deshalb ausführlich beschrieben.

Die Technologie des Cloud Computing stellt insofern eine interessante Möglichkeit speziell für Vereine dar, da Elemente wie eine Benutzerverwaltung fertig implementiert für den Einsatz in Webanwendungen verwendet werden können und hierbei interessante Freikontingente bereitstehen. Das Prinzip der Serverless Architektur basiert zudem auf einem Pay-Per-Use Prinzip, wodurch Rechenzeit für Businesslogik innerhalb einer Anwendung nur bei Bedarf in Rechnung gestellt wird. Dadurch entstehen selbst für eine umfangreiche Anwendung nur geringe Kosten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Unternehmen wie Amazon, Google oder Microsoft sind weltweit bekannt. Alle drei gehören zu den Big Playern im Bereich des Cloud Computing, dessen Marktvolumen sich im Jahr 2021 auf 178 Milliarden US-Dollar belief und damit knapp 37% über dem Vorjahresniveau lag (Synergy Research Group, 2022).

Laut einer Umfrage unter ausgewählten Unternehmen verschiedener Branchen schätzen ca. 50% das Thema Cloud Computing als bedeutsam ein und knapp 30% sehen hierin auch in Zukunft ein wichtiges Feld. Gerade im Hinblick auf die voranschreitende Digitalisierung ist dieses Themengebiet ein wichtiger Baustein für viele Unternehmen (EHI Retail Institute, 2021).

Ziel dieser Arbeit ist eine praxisorientierte Umsetzung einer Webanwendung auf Basis der von Cloud Providern bereitgestellten Diensten und das Aufzeigen von Einsatzgebieten dieser Technologie innerhalb der Vereinsorganisation. Die Entwicklung folgt dabei methodisch dem klassischen Aufbau des Softwareentwicklungsprozesses. Besonderer Fokus wird zudem auf die Beschreibung der Grundlagen und den ersten Schritten für die Arbeit mit Cloud Technologien gelegt.

Die Bachelorarbeit ist dabei wie folgt aufgebaut: Einleitend wird der Digitalisierungsbegriff beschrieben und eine einheitliche Definition für den weiteren Verlauf der Arbeit aufgestellt. Das zweite Kapitel bietet Einblicke in den Ablauf der Softwareentwicklung, indem zuerst das allgemeine Konstrukt eines Projekts mit seinen Phasen erläutert und anhand zwei konkreter Vorgehensmodelle dargestellt wird. Daraufhin wird auf die Bezeichnung des Cloud Computing und wichtige Begrifflichkeiten innerhalb dieses Themengebiets eingegangen sowie dessen Einsatz in Vereinen aufgezeigt. Kapitel 4 greift darauf aufbauend die speziellen Dienste des Cloud Providers Amazon Web Services auf, welche innerhalb der praktischen Anwendung zum Einsatz kommen. In Anlehnung an die Projektphasen zur Softwareherstellung wird im letzten Kapitel die serverlose Implementierung einer Cloud Computing Anwendung beschrieben, welche beispielsweise in Sportvereinen eingesetzt werden kann.

1 Digitalisierung

Der Begriff „Digitalisierung“ ist zu einem der zentralen Schlagwörter im Zeitalter der vierten industriellen Revolution geworden, in der wir uns als Gesellschaft aktuell befinden (Becker, Ulrich, Schmid, & Feichtinger, 2020). Bei näherer Betrachtung besitzt dieser Begriff viele Facetten, jedoch soll die nachfolgende Beschreibung ein einheitliches Verständnis für den Rahmen dieser Arbeit liefern.

1.1 Bedeutung des Digitalisierungsbegriffs

Grundsätzlich handelt es sich hierbei in einem ersten Schritt um die Umwandlung von analogen Daten in für Maschinen lesbare Formate. Dadurch können Informationen gespeichert, in Folgeschritten weiterverarbeitet sowie digital übertragen werden (Mertens, Barbian, & Baier, 2017). Grundlage hierfür legte einerseits Konrad Zuse mit seinem Digitalrechner Z3, welcher im Jahr 1941 erstmalig vorgestellt wurde. Andererseits zählt auch die Einführung der digitalen Einheit für Informationen, den sogenannten „bits“ im Jahr 1948 von J. W. Tukey, als notwendiger Baustein für die heutige digitale Erfassung von Daten (Deckert, 2019). Die Digitalisierung besitzt jedoch noch viel weitreichendere Auswirkungen. So können ganze Prozesse mithilfe von Software abgebildet und hierdurch manuelle Tätigkeiten reduziert werden. Zudem entstehen durch die enorme Menge an gesammelten Daten neue Geschäftsmodelle und Produkte. Die Auswirkungen sind somit für jeden Einzelnen in der Gesellschaft durch beispielsweise digitale Kundenportale, Smartphone-Apps etc. spürbar (Abolhassan, 2016). Da es sich bei der Digitalisierung um ein vielschichtiges und komplexes Thema handelt, benötigen Unternehmen eine Vision, welche in Form einer digitalen Transformation einen Ist-Zustand in einen Soll-Zustand überführt. Ein Leitfaden ist notwendig und für jede Branche und jedes Unternehmen individuell zu erstellen. Technologien wie Cloud Computing oder Big Data dürfen hierbei nicht als Synonym für den Begriff der Digitalisierung verstanden werden. Vielmehr stellen diese beiden Themenfelder lediglich eine technische Lösung zur Umsetzung dar (Hanschke, 2018).

1.2 Digitalisierung in Unternehmen

Für Unternehmen ist die Digitalisierung und digitale Transformation entscheidend, um wettbewerbsfähig zu bleiben. Die Gründe, warum sich Betriebe hiermit auseinandersetzen müssen, können in die zwei Kategorien der Treiber und Enabler eingeordnet werden, die in der nachfolgenden Grafik dargestellt sind.

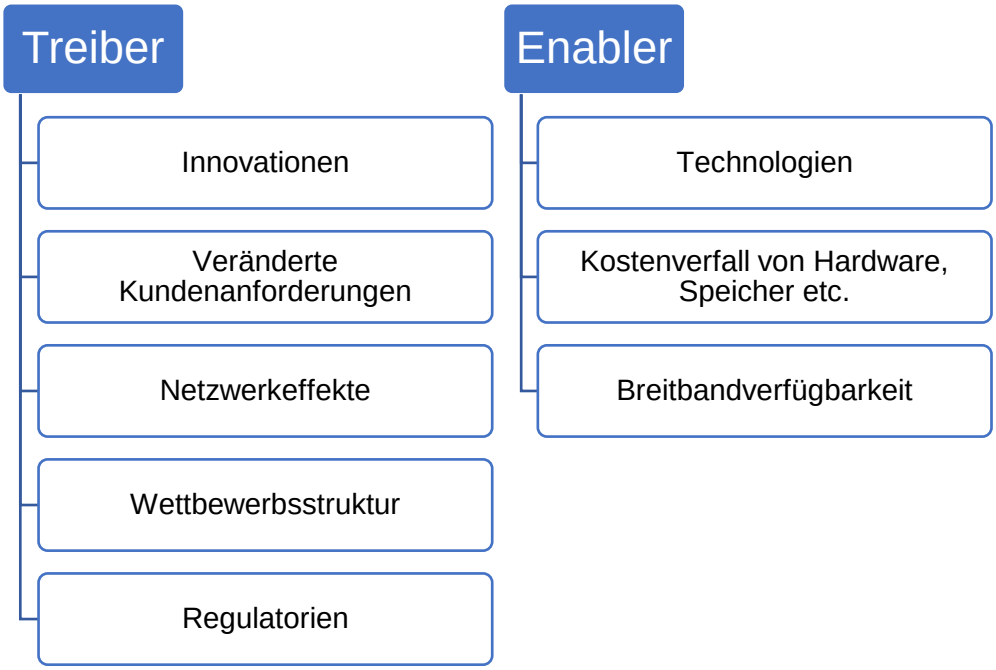


Abbildung 1: Treiber und Enabler der Digitalisierung.

Der Begriff „Treiber“ fasst Aspekte zusammen, weshalb sich jedes Unternehmen mit diesem Thema beschäftigen sollte. Der zweite Ausdruck „Enabler“ beschreibt, wie Innovationen überhaupt möglich gemacht werden. Anzumerken ist, dass es sich zwischen diesen Begriffen nicht um separierte Bereiche handelt, sondern durchaus eine Wechselwirkung besteht. So können beispielsweise neue digitale Technologien – aus dem Bereich der Enabler – als Auslöser für veränderte Kundenanforderungen gelten.
Schlussendlich entstehen aus diesen Veränderungen neue Produkte und Geschäftsmodelle sowie eine Verkürzung der Produktlebenszyklen. Dies hat einen wesentlichen Einfluss auf Unternehmen, denn um auch in Zukunft konkurrenzfähig zu bleiben, müssen sie sich mit dem Thema der Digitalisierung und einer fundamentalen Strategie hierzu auseinandersetzen (Hanschke, 2018).

1.3 Digitalisierung in gesellschaftlichen Einrichtungen

Der Trend der Digitalisierung ist jedoch nicht nur im beruflichen Umfeld und der Wirtschaft zu sehen. Auch das Privatleben wird zunehmend mit digitalen Bestandteilen bestückt. Ganz konkret besteht häufig auch Handlungsbedarf in gesellschaftlichen Einrichtungen wie beispielsweise Sportvereinen. Sie stehen meist vor der Herausforderung, ihren Mitgliedern sowohl digitale Services anzubieten als auch selbst Prozesse und Abläufe mithilfe von Software zu erneuern, um effizienter arbeiten zu können. Auch diese Einrichtungen können von den Enablern und Treibern der Digitalisierung profitieren,

wenn eine passende Strategie erarbeitet wird. Die Alexander Otto Sportstiftung setzt dieses Thema in den Mittelpunkt und stellt hierfür einen praktischen Ratgeber in Form eines Praxishandbuchs bereit. In Zusammenarbeit mit einem Sportverein wurden in einem mehrstufigen Projekt Handlungsfelder analysiert und Maßnahmen wie die Implementierung einer Vereinshomepage, eines Mitgliederportals und weiteren digitalen Produkten aufgestellt. Eine sukzessive Umsetzung dieser Punkte kann demnach zu einer ganzheitlichen Stärkung des Vereins und des Miteinanders führen, da sich Trainer und Vereinsfunktionäre mehr auf wesentlichen Aufgaben anstatt auf organisatorische Tätigkeiten konzentrieren können (Barmscheidt, Ebert, Hoppe, Kelmes, & Röthemeier, 2018).

2 Softwareentwicklung

Wie im Vorfeld kurz beschrieben, muss Digitalisierung im privaten als auch beruflichen Umfeld ganzheitlich betrachtet werden. Im Wesentlichen handelt es sich hierbei jedoch um den Einsatz von Software. Deshalb beleuchtet dieses Kapitel grundlegend, wie eine Software entsteht.

2.1 Projekt

Offiziell kann ein Vorhaben als Projekt bezeichnet werden, wenn die nachfolgenden Kriterien der DIN 69901 erfüllt sind:

- Definierter Zeitraum
- Projektziel
- Verantwortlichkeiten
- Einmaligkeitscharakter / Neuartigkeit des Vorhabens
- Begrenzte Ressourcen
- Interdisziplinäre Vorgehensweise
- Komplexität

Bei der Entwicklung von Software sind diese Aspekte meist vorhanden, weshalb sie innerhalb eines Rahmens, dem sogenannten Projekt entsteht (Meyer, 2018). Wichtige Pfeiler für ein Vorhaben sind dabei das Ziel, die Zeit und das Budget. In der Literatur wird im Zusammenhang mit diesen drei Begriffen vom magischen Dreieck, wie Abbildung 2 illustriert, im Projektmanagement gesprochen. Das liegt daran, dass sich

die Größen gegenseitig beeinflussen. So hat beispielsweise eine Veränderung des zeitlichen Rahmens einen direkten Einfluss auf das Budget (Kuster, et al., 2019).

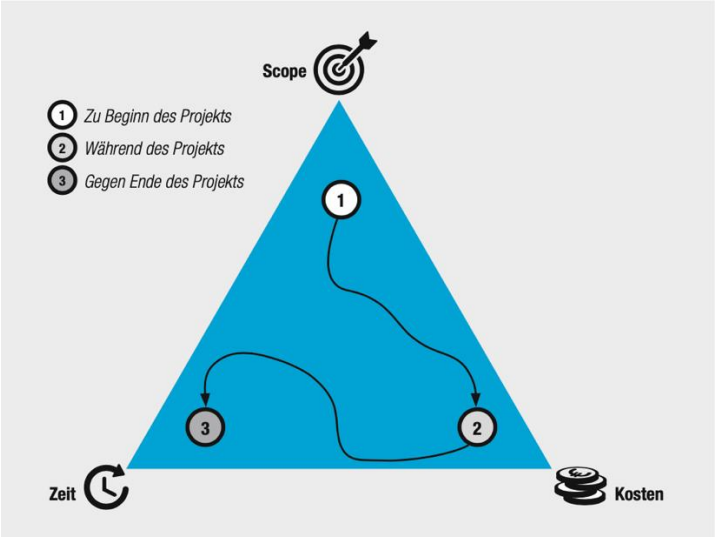


Abbildung 2: Magisches Dreieck. Quelle: (Kuster, et al., 2019)

In Unternehmen laufen meistens mehrere Projekte parallel ab. Zur Übersichtlichkeit können diese anhand der Projektdauer, Projektgröße oder der Projektart klassifiziert werden. Zeitlich gesehen können Vorhaben von wenigen Wochen bis hin zu mehreren Jahren reichen. Der Umfang eines Projekts lässt sich einerseits anhand der Kosten und andererseits anhand der beteiligten Personen vergleichen. Das letzte Unterscheidungsmerkmal gibt meist Auskunft über das Ziel eines Projekts. Zur Verdeutlichung seien an dieser Stelle das Forschungsprojekt zur Entdeckung neuer Geschäftsfelder und die Kategorie der Rationalisierungsprojekte zum Aufdecken von Einsparungspotentialen zu nennen (Peters & Schelter, 2021).

2.2 Team

Das vorherige Unterkapitel hat die Notwendigkeit von Projekten für die Einführung von Software verdeutlicht. Um diese auch erfolgreich umsetzen zu können, werden Teams benötigt. Dabei handelt es sich um die interdisziplinäre Zusammensetzung von Personen, damit die anfallenden Aufgaben innerhalb eines Vorhabens erreicht werden können. Personen werden dabei anhand der benötigten Funktionen, der Fähigkeiten bzw. des vorhandenen Wissens oder einer Rolle zu einem Projekt zugeteilt. Entscheidend für den Erfolg ist jedoch, dass für jede anfallende Aufgabe relevantes Know-how im Projektteam verfügbar ist. Bei sehr großen Vorhaben kann es erforderlich sein, dass mehrere Projektteams benötigt werden, welche horizontal (z.B. ein Team für die

Datenbank, eines für das Backend) oder vertikal (ein Team pro Produktfeature) an den anfallenden Aufgaben arbeiten. Gerade die Koordination und Versorgung mit Informationen von mehreren Teams stellt hierbei die größte Herausforderung dar. Abhilfe kann an dieser Stelle ein „Single Point of Information“ schaffen, also ein einziger Ort für Informationen (Meyer, 2018).

2.3 Phasen der Softwareentwicklung

Um ein strukturiertes Vorgehen bei der Erstellung von Software zu gewährleisten, wird ein Projekt in Phasen aufgeteilt, woraus Artefakte hervorgehen und in darauffolgenden Entwicklungsschritten weiterverarbeitet werden. Im Folgenden wird die allgemeine Unterteilung der Phasen sowie deren Notwendigkeit beschrieben (Kuster, et al., 2019).

2.3.1 Initialisierungsphase

Zu Beginn werden relevante Informationen bezüglich der Machbarkeit, den Risiken und Stakeholdern für das Vorhaben gesammelt, indem Machbarkeitsanalysen, Befragungen etc. stattfinden. Am Phasenende folgt das Zusammentragen aller Daten, woraufhin Risiken und Unsicherheiten abgewogen und das Einleiten oder Abbrechen des Projekts beschlossen wird. Dieser Schritt soll eine Fehlallokation von Ressourcen verhindern (Kuster, et al., 2019). Vier Schritte sind während dieser Phase durchzuführen. Die Ist-Aufnahme, welche den Zustand des aktuellen Systems dokumentiert, stellt den Startpunkt dar. Anhand dieser Aufzeichnungen erfolgt die Ist-Analyse, bei der Schwachstellen aufgenommen werden. Hierzu sind in Schritt drei zugleich potenzielle Varianten für Verbesserungen zu erarbeiten. Im letzten Schritt entsteht das Soll-Konzept, indem aus einem Pool an Alternativen die optimale Lösung herausgegriffen wird. Der ursprüngliche Stand der Software, sowie das Zielsystem werden schlussendlich in dem sogenannten Lastenheft festgehalten. Dieses Artefakt definiert, was im Rahmen des Projekts umgesetzt werden soll und ist dabei aus Sicht des Auftraggebers verfasst. Zur Einschätzung der zeitlichen und finanziellen Dimension wird das Lastenheft mit umzusetzenden Aufgaben als Grundlage für Schätzungen verwendet. Hierfür gibt es zahlreiche Methoden, welche die vorliegende Arbeit jedoch nicht aufgreift (Metzner, 2020).

2.3.2 Definitionsphase

In dieser zweiten Phase werden Anforderungen, auch bezeichnet als fachliche Lösung eines Problems, an ein Projekt herausgearbeitet. Neben schriftlichen Aufzeichnungen entstehen meist zahlreiche Diagramme, welche mithilfe der Unified Modeling Language (UML) Notation erstellt werden. Als Beispiel sei an dieser Stelle das Use-Case-

Diagramm zu nennen, welches im Praxisteil mithilfe eines Beispiels verdeutlicht wird. Als Ausgangspunkt für die Ermittlung von Anforderungen sind Anwenderwissen, eigene Beobachtungen der Prozesse, Dokumentationen, Lastenhefte etc. relevant. Die so entstandenen Ergebnisse werden anschließend analysiert, dokumentiert und auf Korrektheit überprüft.

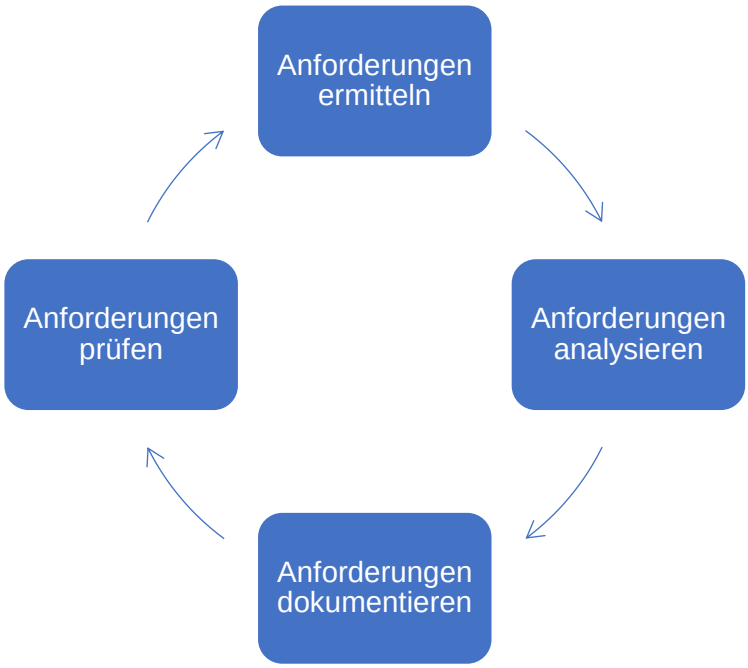


Abbildung 3: Zyklus der Anforderungsanalyse. Quelle: In Anlehnung an (Metzner, 2020)

Die Informationen werden in das Pflichtenheft übertragen und definieren, was mithilfe der Software umgesetzt werden soll. Die mit Abbildung 3 verbundenen Tätigkeiten werden meist unter dem Begriff der Anforderungsanalyse bzw. des Requirements Engineering zusammengefasst (Metzner, 2020).

2.3.3 Designphase

Nachdem die vorherigen Phasen ein gemeinsames Verständnis entwickelt haben, was umgesetzt werden soll, beschäftigt sich die Designphase mit der Frage, wie dies technisch möglich ist. Auch hier werden Modelle zur Unterstützung herangezogen und daraufhin ein Softwaredesign erstellt. Relevant sind dabei die Makroarchitektur und die Mikroarchitektur (Metzner, 2020). Erstere beschreibt die zu erstellende Software auf einem hohen Abstraktionsniveau. Dabei werden grundlegende Elemente und Strukturen bestimmt und anschließend mit höherer Genauigkeit dargestellt. Ab einem durch die Stakeholder und Softwareentwickler bestimmten Punkt führt eine weitere Verfeinerung

der Makroarchitektur zum Übergang in eine Mikroarchitektur. Sie beschreibt die Implementierung der Software sehr detailliert. Meist werden dabei Design Pattern angewendet, welche einen Rahmen wie z.B. der Programmcode zu implementieren ist, vorgeben. Architekturspezifische Elemente besitzen somit keine Relevanz mehr, da diese in der abstrakten Makroarchitektur spezifiziert sind (Vogel, Arnold, Chughtai, & Kehrer, 2011).

2.3.4 Implementierungsphase

Alle bisher erstellten Artefakte wie Modelle und schriftliche Aufzeichnungen werden an dieser Stelle vereint. Besonders entscheidend sind die Modelle aus der vorherigen Designphase. Sie dienen als wichtige Grundlage für die Programmierung der Software. Im Zuge dessen entstehen zugleich notwendige Dokumentationen und Handbücher für die spätere Wartung der Software (Brandt-Pook & Kollmeier, 2020).

2.3.5 Test- und Abnahmephase

Nach der Programmierung der Software folgt die Test- und Abnahmephase, welche dazu beiträgt, dass qualitativ hochwertige digitale Produkte entstehen. Das Testen sollte hierbei nicht allein den Entwicklern überlassen werden, da sonst Fehler übersehen werden könnten. Deshalb gibt es heutzutage meist eigene Berufe als Softwaretester und ganze Instanzen in Unternehmen, welche mit der Aufgabe betraut sind, die programmierten Anwendungen zu untersuchen. Zeitlich gesehen startet dieser Aufgabenbereich bereits während der Softwaredesign- und Implementierungsphase.
Damit Fehler jeglicher Art möglichst frühzeitig erkannt werden, sind verschiedene Teststufen vorhanden. Als kleinste, testbare Einheit und damit erste Teststufe ist hierbei der Modultest bzw. Unittest zu nennen. Dabei wird ein einziges funktionales Einzelteil, auch häufig als Modul bezeichnet, mithilfe von stichprobenhaften Testfällen durch den Entwickler selbst getestet. Da dabei der Programmcode ersichtlich ist und somit die Ausgabe nachvollzogen werden kann, ist hier die Rede von sogenannten „White-Box-Tests“. Auf den Unittest aufbauend folgt der Integrationstest. Hier werden abhängige Komponenten einer Software auf eine reibungslose Zusammenarbeit hin untersucht. Die Prüfung des Systems als Ganzes erfolgt im Systemtest. Neben den funktionalen sind zu diesem Zeitpunkt erstmalig nicht-funktionale Anforderungen wie die Validierung von Performance, Benutzbarkeit etc. im Test enthalten. Um aussagekräftige Ergebnisse zu erzielen, ist eine möglichst reale Testumgebung entscheidend. Die letzte Stufe stellt der Abnahmetest dar. Dieser wird von den Endanwendern absolviert. Ziel ist nun nicht mehr

Fehler zu finden, sondern die Kunden anhand wichtiger Alltagsabläufe mit dem System vertraut zu machen (Witte, 2019).

2.3.6 Wartungsphase

In der letzten Phase ist die Software produktiv im Einsatz. Schulungen und Supportmitarbeiter helfen den Anwendern Funktionalitäten besser verstehen und nutzen zu können. Trotz einer ausgiebigen Testphase sind Fehler im laufenden Betrieb möglich. Wünsche nach neuen Eigenschaften an der Software können im Laufe der Zeit auftreten. Diese werden in der Wartungsphase auf Grundlage eines Wartungsprozesses erfüllt (Metzner, 2020).

2.4 Vorgehensmodelle

Unterschiedliche Rahmenbedingungen erfordern verschiedene Herangehensweisen für die Erstellung einer Software. Deshalb gibt es eine Reihe von Vorgehensmodellen, welche grundlegend die Phasen, die im Abschnitt 2.3 beschrieben sind, enthalten. Modellspezifische Ausprägungen wie beispielsweise die Durchführung vieler Tests oder der Fokus auf eine enge Zusammenarbeit mit dem Kunden lassen jedoch eine Unterscheidung der Modelle zu. Die Relevanz von Vorgehensmodellen wird besonders bei großen Projekten deutlich, da nur durch deren Einsatz die Komplexität beherrschbar gemacht werden kann, indem Aufgaben in einzelne Schritte zerlegt werden. Ein Vorgehensmodell spricht demnach auch für die Qualität eines Softwareprodukts, da vor dem Handeln ausreichend Zeit und Wissen in einen Plan zur Umsetzung investiert wurde (Brandt-Pook & Kollmeier, 2020).

Heutzutage können Unternehmen und Entwickler auf eine Vielzahl an Projektmanagement-Methoden zurückgreifen. Eine Auswahl der am weitesten verbreiteten Modellen zeigt Abbildung 4.

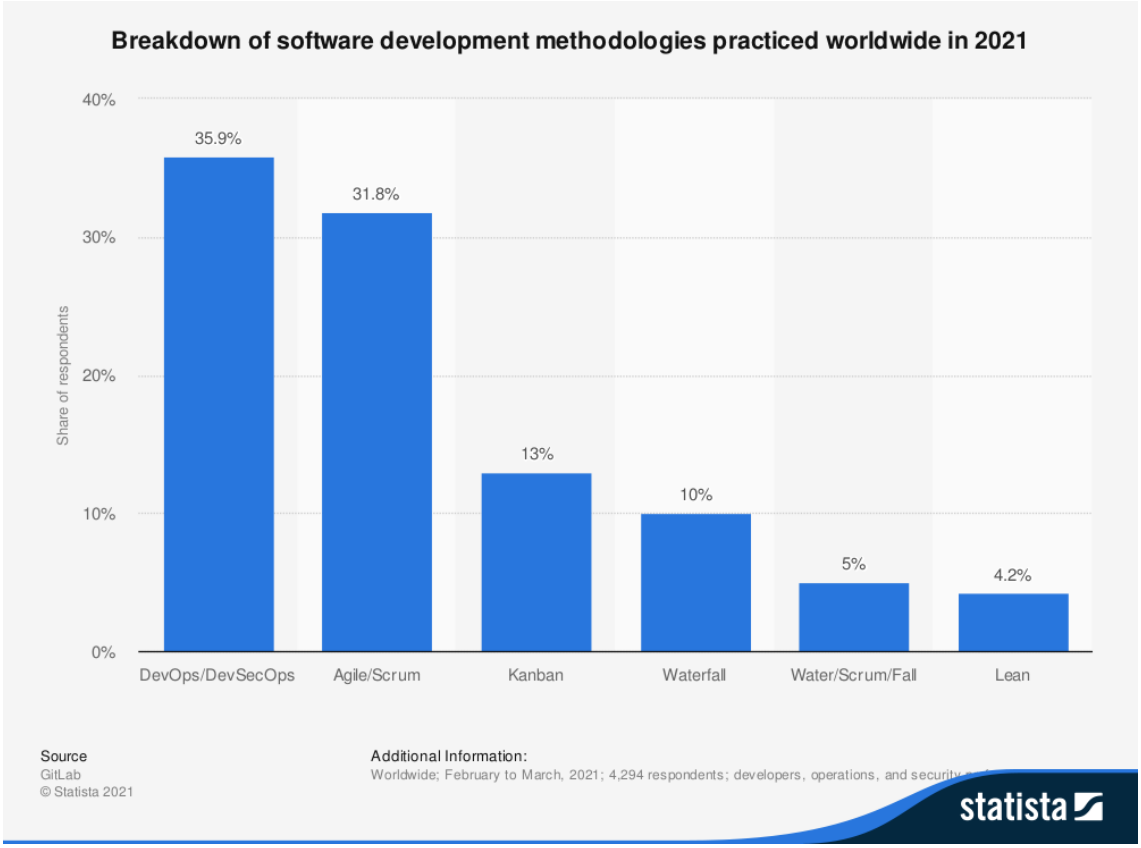


Abbildung 4: Verbreitung der Vorgehensmodelle im Jahr 2021. Quelle: (GitLab, 2021)

Alle bekannten Modelle lassen sich grundlegend in die Kategorien der klassischen oder agilen Ansätze einordnen. Exemplarisch hierfür wird im Folgenden auf das Wasserfallmodell und Scrum eingegangen.

2.4.1 Wasserfallmodell

Eines der ältesten Verfahren zur Erstellung von Software ist das Wasserfallmodell, welches sich in die Kategorie der klassischen Ansätze einordnen lässt. Die Phasen werden in diesem Modell einmal nacheinander durchlaufen und bauen aufeinander auf. Das zu erreichende Ziel ist bei dieser Art von Projekten bereits genau festgelegt und die beiden anderen Punkte des magischen Dreiecks, Zeit und Kosten, sind geschätzt. Mithilfe einer klaren Rollenverteilung entsteht in Teamarbeit ein Produkt, welches dem Kunden am Projektende präsentiert wird. Insbesondere der strikte Ablauf der Phasen führt dazu, dass das Wasserfallmodell schlicht und einfach zu verstehen ist. Gleichzeitig ist es jedoch unflexibel und lässt das Einfließen von Anforderungsänderungen nur schwer zu (Brandt-Pook & Kollmeier, 2020), (Hindel, Hörmann, Müller, & Schmied, 2009).

2.4.2 Scrum

Als Vertreter der agilen Methoden wird nachfolgend Scrum beschrieben. Das Agile Manifest sowie die Agilen Prinzipien sind hierbei zwei grundlegende Artefakte, auf denen dieses Vorgehensmodell beruht. Sie legen dar, auf welche Art und Weise die Softwareentwicklung betrieben wird und worauf der Fokus bei dieser Arbeit liegt. Die Software wird nicht in einem Stück ähnlich dem Wasserfallmodell, sondern mithilfe von Produktinkrementen erstellt. Diese entstehen innerhalb von aufeinanderfolgenden Iterationen, welche auch als Sprints bezeichnet werden und einen Zeitaufwand von zwei Wochen haben. Jede Iteration enthält für sich die gängigen Projektphasen aus Unterkapitel 2.3. Durch die ständige Wiederholung und Überarbeitung der entstandenen Software können Anforderungsänderungen von der Kundenseite gut integriert werden (Brandt-Pook & Kollmeier, 2020). Dieser Aspekt ist für agile Methoden charakteristisch, da sie sich nicht nur auf das zu entstehende Produkt fokussieren, sondern spätere Nutzer und weitere Stakeholder in den Mittelpunkt stellen. Dies wird insbesondere durch ausreichende und eindeutige Kommunikation während des gesamten Prozesses erreicht.
Die folgende Abbildung stellt den Ablauf von Scrum skizzenhaft dar. Auf die nähere Beschreibung der einzelnen Rollen wird an dieser Stelle verzichtet.

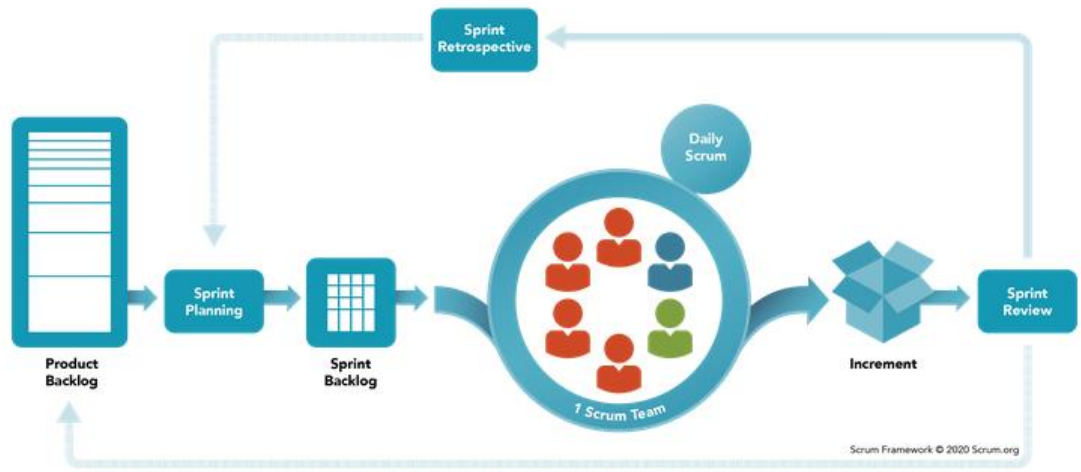


Abbildung 5: Der Scrum Prozess. Quelle: (scrum.org, o.D.)

Zu Beginn besteht eine Produktvision, welche durch den Product Owner (PO) verwaltet wird. Dieser allein oder alternativ das komplette Scrum Team bringt Ideen für neue Funktionalitäten, welche auch als User Stories bezeichnet werden, in das Product Backlog ein. Innerhalb von zwei Sprint Plannings wird das Ziel für den nächsten Sprint festgelegt. Zudem entscheidet das Team gemeinsam, welche User Stories sie aus dem Product Backlog abarbeiten möchten und ermitteln gleichzeitig die dafür notwendigen

Schritte, die schlussendlich im Sprint Backlog zusammengefasst werden. Innerhalb eines Sprints bearbeiten die Mitarbeiter anschließend die aufgestellten Tätigkeiten aus dem Sprint Backlog. Das Daily Scrum stellt während dieser Zeit einen festen Besprechungstermin dar, bei dem jedes Teammitglied kurz über den aktuellen Entwicklungsstand berichtet. Am Ende eines Iterationszyklus wurde somit ein Produktinkrement – ein Produkt mit neuen Funktionen – geschaffen, welches im Sprint Review demonstriert und in die Produktivumgebung eingebettet wird. Bei der Sprint Retrospektive wird die Zusammenarbeit während des Sprints untersucht und Verbesserungsvorschläge für den Nächsten erarbeitet (Gloger & Häusling, 2011).

3 Cloud Computing

Nachdem die vorherigen Abschnitte das Thema der Softwareentwicklung näher beleuchtet haben, wird nachfolgend das Themengebiet des Cloud Computing erklärt.

3.1 Definition

Cloud Computing bezeichnet das Konzept zur gemeinsamen Nutzung von Computerressourcen wie beispielsweise Netzwerke, Server, Speicherplatz, Anwendungen und Diensten, welche von sogenannten Cloud Providern zur Verfügung gestellt werden. Die Bereitstellung der Ressourcen kann dabei jederzeit und bedarfsgerecht über ein Netzwerk an die Kunden erfolgen und stellt somit einen geringen Aufwand sowohl für den Nutzer als auch den Cloud Provider dar (Mell & Grance, 2022). Da die Bereitstellung der Computerressourcen über das Internet erfolgt, werden in Grafiken zur Beschreibung von Cloud Architekturen meistens Wolken als Piktogramme verwendet. Daher stammt der Begriff „Cloud Computing“, denn jeder Rechner mit Internetanbindung besitzt somit die Möglichkeit, die große Anzahl an Ressourcen von Cloud Providern zu nutzen (McHaney, 2021).

3.2 Entstehung von Cloud Computing

Das Gebiet des Cloud Computing, wie es heute allgemein verstanden wird, hat sich über viele Jahrzehnte entwickelt und konnte nur entstehen, da sich die Technologien der Computer die letzten Jahre ständig verändert haben. Mainframes, welche auf die 1960er Jahre zurückgehen, werden im Allgemeinen als Grundlage für das heutige Gebiet des Cloud Computing angesehen. Als bekannteste Firma sei an dieser Stelle IBM zu nennen. Zu Beginn waren diese großen Rechner auf den Single-User Modus ausgelegt.

Mit der Einführung des Time-Sharing Konzeptes und der Multi-User Fähigkeit in den 1970er Jahren, konnte ein Mainframe als zentrale Recheneinheit für eine Vielzahl von virtuellen Maschinen dienen, welche per Thin-Client an die Nutzer ausgeliefert wurden. Berechnungen und Datenhaltung fanden dabei zentral am Mainframe statt und konnten von Benutzern durch eine Ein- und Ausgabemöglichkeit per Desktop und Tastatur genutzt werden. Gerade die Aspekte der zentralen Datenhaltung und Auslagerung von Rechenkapazität zeigen Parallelen zum heutigen Cloud Computing auf.

In den 1980er Jahren wurde verstärkt auf Personal Computer (PC) gesetzt, weshalb sich die Speicherung von Daten und Durchführung von Berechnungen auf die Desktop Systeme verlagerte. Üblich für die Zeit war das Zusammenschalten von mehreren Personal Computern, wodurch sich die verfügbare Rechenleistung aus der Leistung des individuellen und der verbundenen Geräte zusammensetzte. Diese Architektur ist auch heute noch besser als Client-Server Architektur bekannt. Ein PC innerhalb dieser Infrastruktur wird als Thick-Client bezeichnet, da dieser unabhängig von einem Server und dessen Hardwareressourcen betrieben werden kann (McHaney, 2021).

Mitte der 1990er Jahre entstand das World Wide Web, welches von Computern im Zusammenspiel mit dem TCP/IP Protokoll genutzt wurde, um auf entfernte Server zuzugreifen.

Das Jahr 2005 wird als Startpunkt des Cloud Computing Zeitalters angesehen. Rechenleistung wurde wieder zentralisiert von Maschinen vorgehalten und bei Bedarf an Nutzer und deren Geräte ausgeliefert.

Treiber für diese Entwicklung waren unter anderem die stark ansteigenden Zahlen an mobilen Endgeräten und die Entstehung von Social Media Plattformen, welche bis heute Anwendungen für viele Nutzer über das Internet bereitstellen.

Die wichtigsten Aspekte, um Cloud Computing zu ermöglichen, sind Veränderungen in der Hardware und Software. Ersteres ergibt sich aus der Leistungssteigerung von Computern (McHaney, 2021). Diese wird u.a. anhand des Mooreschen Gesetzes deutlich, welches besagt, dass sich die Rechenkapazität alle zwölf Monate verdoppelt (Moore, 1965). Zudem sind über die letzten Jahre hinweg die Kosten für Speicherkapazität stetig gesunken. Auf Seiten der Softwareebene sei einerseits die Virtualisierung und anderseits die sogenannte Service Orientated Architecture (SOA) als wichtige Bestandteile zu nennen. Die SOA ist ein Rahmenwerk und beschreibt, wie große Software organisiert und erstellt werden soll. Die Aufteilung von großen monolithischen Architekturen erfolgt dabei in kleinere Services, welche wiederverwendet und einfach erweitert werden können (McHaney, 2021). Abbildung 6 fasst die Entwicklung von Cloud Computing prägnant in einem Schaubild zusammen.

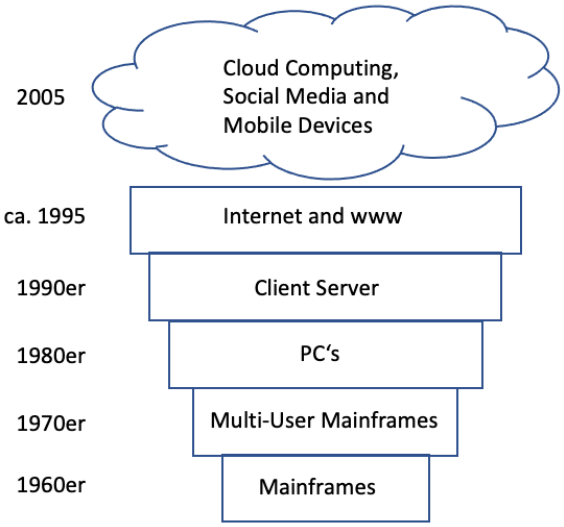


Abbildung 6: Schritte zur Entstehung von Cloud Computing. Quelle: In Anlehnung an (McHaney, 2021)

3.3 Services im Cloud Computing

Die Ressourcen, welche im Rahmen des Cloud Computing von Providern zur Verfügung gestellt werden, lassen sich in eine von vier aufeinander aufbauenden Stufen, welche in den folgenden Unterkapiteln beschrieben werden, einteilen. Abbildung 7 illustriert den Zusammenhang der verschiedenen Ebenen. Die Abkürzung für den Überbegriff aller Services wird häufig mit XaaS abgekürzt (RedHat, 2019).

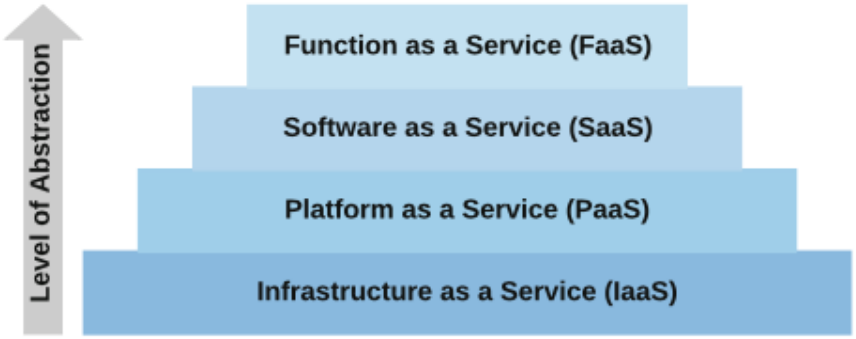


Abbildung 7: Services im Cloud Computing. Quelle: (Añel , Montes, & Iglesi, 2020)

3.3.1 IaaS

Der Begriff IaaS steht für Infrastructure as a Service. Wie der Name schon zu erkennen gibt, wird grundlegende virtuelle Computer Hardware für eine funktionierende Infrastruktur zur Verfügung gestellt. Zu dieser Kategorie zählen unter anderem virtuelle Maschinen, Datenbanken, Netzwerke oder komplette Server. Ein Beispiel hierfür ist Amazon EC2 (Stigler, 2018) (Añel , Montes, & Iglesi, 2020).

3.3.2 PaaS

Die Abkürzung steht für Platform as a Service und stellt eine komplette Softwareplattform inklusive Betriebssystem, Datenbank, Webserver etc. bereit. Nutzer erhalten somit die komplette notwendige Infrastruktur und Umgebung für einen reibungslosen Betrieb einer mobilen oder webbasierten Applikation. Die Anwendung Amazon Elastik Beanstalk kann hier als Beispiel genannt werden (Stigler, 2018). Besonders für Entwickler ist diese Form interessant, da sie sich hiermit komplett auf das Erstellen der Anwendung fokussieren können und sich nicht um den Aufbau, die Wartung und Bereitstellung dieser sorgen müssen (RedHat, 2019).

3.3.3 SaaS

Bei der Bereitstellung einer kompletten Softwareanwendung ohne die Notwendigkeit der Installation oder Wartung, wie z.B. Softwareupdates durch den Nutzer, handelt es sich um Software as a Service. Die Benutzung der gebuchten Anwendung, wie z.B. Salesforce oder Microsoft 365 erfolgt dabei meist über einen Browser (Stigler, 2018).

3.3.4 FaaS

Als letzte Kategorie sei Function as a Service zu nennen. Diese Ausprägung des Cloud Computing besitzt Ähnlichkeiten mit PaaS, unterscheidet sich jedoch darin, dass keine kontinuierliche Verwaltung und Vorhaltung der Ressourcen stattfinden. FaaS enthalten die Programmlogik für eine serverlose Anwendung und werden dabei ereignisgesteuert, beispielsweise durch einen Buttonklick des Nutzers, aufgerufen (RedHat, 2019). Daraufhin stellt der Cloud Provider die benötigte Programmlogik innerhalb eines zustandslosen virtuellen Containers bereit, führt diese aus und gibt das Ergebnis an die aufrufende Stelle zurück. Somit ist keine Konfiguration und Wartung von Servern mit entsprechender Laufzeitumgebung notwendig (Varshney & Sharma, 2021). Interessant ist dies insbesondere für Softwareentwickler, da Hardwareressourcen und Server von einem Cloud Provider verwaltet werden und der komplette Fokus auf die Programmlogik gelegt

werden kann. AWS Lambda stellt einen Service von Amazon dar und wird häufig für mobile und IoT Anwendungen genutzt (Stigler, 2018).

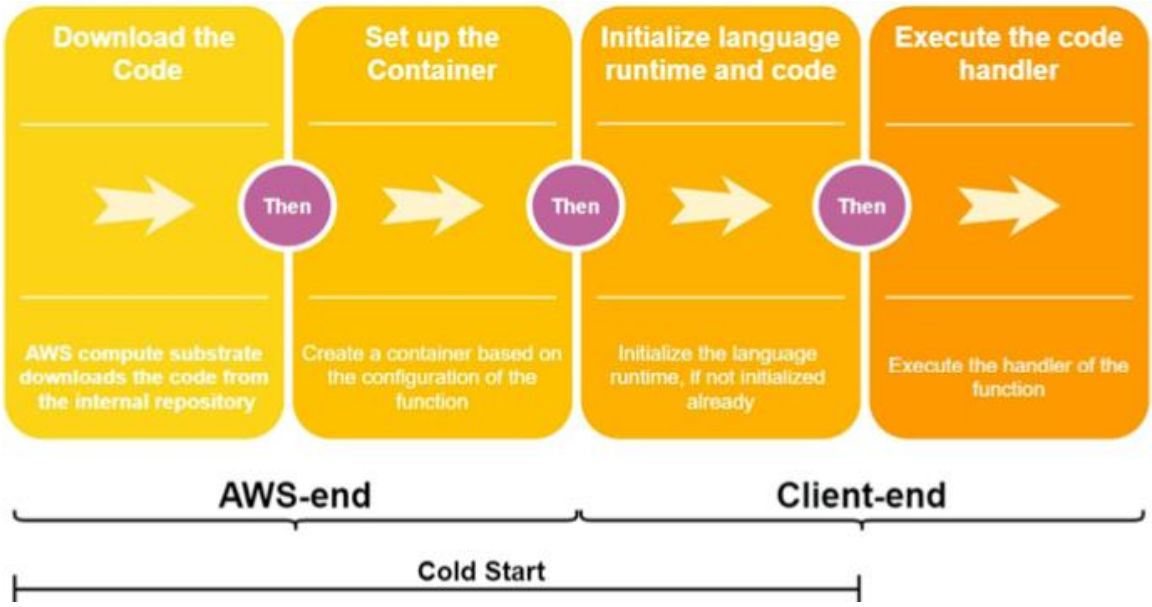


Abbildung 8: Schritte zur Ausführung einer FaaS. Quelle: (Varshney & Sharma, 2021)

Die vorangehende Abbildung zeigt auf, welche Schritte beim Ausführen einer Function as a Service im AWS Umfeld stattfinden. Zuerst greift der Cloud Provider auf den gespeicherten Programmcode zu und baut hierfür einen virtuellen Container auf. Für die Ausführung ist in Schritt drei die Initialisierung der Laufzeitumgebung notwendig. Danach kann die eigentliche Logik der Anwendung ausgeführt werden. Beim erstmaligen Aufruf einer serverlosen Funktion muss zunächst ein Container bereitgestellt werden, wodurch eine zeitliche Latenz entsteht. Dieser Vorgang wird daher auch als Cold Start bezeichnet. Zeitnahe Folgeaufrufe hingegen liefern aufgrund eines bestehenden Containers schneller die gewünschten Rückmeldungen und Ergebnisse (Varshney & Sharma, 2021)**.**

Häufig wird FaaS mit dem Serverless Computing Begriff in Verbindung gebracht. Als Synonyme können sie jedoch nicht verwendet werden, denn Serverless Computing ist vielmehr ein Ansatz zur Entwicklung von Anwendungen. FaaS stellt lediglich einen Teilbereich dar, solche serverlosen Applikationen zu erstellen (RedHat, 2019).

Entwickler bestimmen im Vorhinein welche Hardware benötigt wird. Um die Bereitstellung, Wartung und Überwachung der Prozesse im Hintergrund kümmert sich schlussendlich der Cloud Provider. Dadurch entfällt die Notwendigkeit eines fundierten Fachwissens zum Aufsetzen von Webservern etc. (Gabbrielli, et al., 2019).

FaaS stellt die flexibelste Art des Cloud Computing Ansatzes dar und ist zudem hoch skalierbar, da bei Bedarf weitere Ressourcen hinzugefügt werden können. Nur bei Beanspruchung der Hardware kommt es zu einer Bereitstellung und Berechnung der dafür angefallenen Kosten – Anwendung des sogenannten Pay-Per-Use Prinzips (Stigler, 2018) (Microsoft, o.D.). Die folgende Abbildung illustriert, welche Tätigkeiten bei den verschiedenen Cloud Services vom Kunden selbst oder dem Cloud Provider übernommen werden.

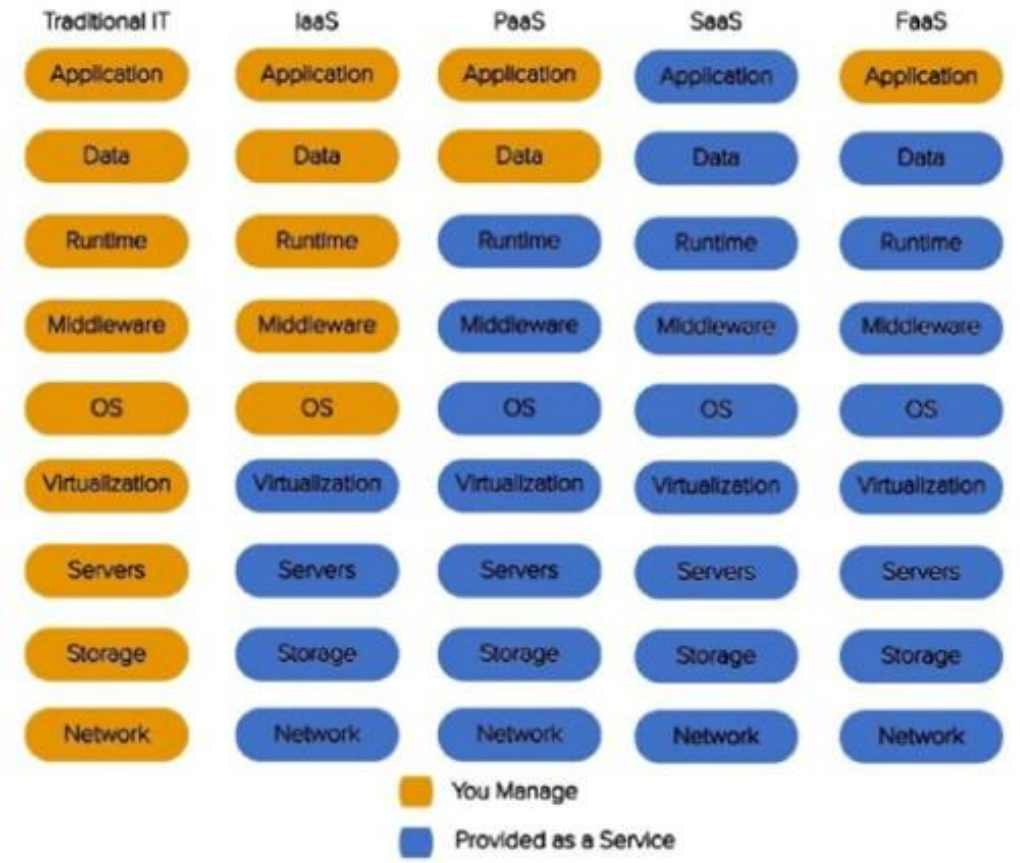


Abbildung 9: Zuständigkeit für die IT in Abhängigkeit des Cloud Computing Services. Quelle: (Stigler, 2018)

3.4 Bereitstellungstypen von Cloud Computing

Die enthaltenen Services aus dem vorherigen Abschnitt können auf unterschiedlichen Wegen bereitgestellt werden. Grundlegend lassen sich dabei die drei Arten der Bereitstellung via Public Cloud, Private Cloud und Hybrid Cloud unterscheiden. Bei Ersterem steht die Hardware z.B. Server in großen Rechenzentren und wird von Cloud Providern betrieben und gewartet. Zugriff auf Ressourcen haben Kunden ausschließlich über das

Internet. Aufgrund von Datenschutzbestimmungen oder geschäftskritischen Daten dürfen Public Clouds beispielsweise von Banken nicht verwendet werden. Hierfür soll die Private Cloud als Ersatz dienen. Die Hardware steht dabei in eigenen Räumen oder Rechenzentren. Alternativ können Server auch bei externen Anbietern angesiedelt sein. Dabei müssen jedoch Sicherheitsmaßnahmen für die Abgrenzung der Tenants auf den Servern gewährleistet werden. Die Verwaltung der Infrastruktur erfolgt hierbei in einem privaten Netzwerk. Eine Mischung der beiden eben genannten Ansätze stellt die Hybrid Cloud dar. Dabei werden meist geschäftskritische Daten in einer Private und restliche Daten in einer Public Cloud gespeichert (Añel , Montes, & Iglesi, 2020) (Microsoft, o.D.).

3.5 Cloud Provider

Dieser Absatz soll einen kurzen Überblick der bekanntesten kommerziellen und Open Source Anbieter von Cloud Produkten bieten. An erster Stelle sei hier Amazon Web Services (AWS) genannt. Diese von Amazon im Jahr 2006 gegründete Plattform gilt als erste Cloud Plattform am Markt. Nur zwei Jahre später folgte die Google Cloud Plattform (GCP), welche zu der Alphabet Tochter Google gehört. Zu den führenden Cloud Providern zählt zudem IBM mit ihrer IBM Cloud. Das Cloud Geschäft des Unternehmens erfuhr besonders durch den Zukauf von Red Hat einen Aufschwung. Neben diesen kommerziellen Public Cloud Anbietern stellen Open Source Projekte wie OpenNebula oder OpenStack eine Alternative für den Einsatz von Cloud Computing bereit. Hierbei handelt es sich meistens um einen Ansatz der Private Cloud, bei der Software unter einer freien Lizenz bereitgestellt und somit eigenständig in vorhandene Rechenzentren integriert werden kann (Añel , Montes, & Iglesi, 2020).

3.6 Vor- und Nachteile von Cloud Computing

Das abschließende Unterkapitel in die Einführung des Cloud Computing befasst sich mit Vor- und Nachteilen dieser Technologie.

Ein grundlegender Vorteil besteht in der Bereitstellung der benötigten Hardware und Software durch den Anbieter, sodass der Fokus direkt auf die zu erstellende Kernanwendung bzw. das Produkt gerichtet werden kann. Die Anschaffung teurer Hardware wird somit vermieden. Eine hoch skalierbare IT ermöglicht die Bereitstellung der grundlegenden Infrastruktur über das Internet, da Server oder Datenbanken je nach Auslastung erweitert oder reduziert werden können. Diese Skalierung der Ressourcen bedarf dabei keinem physischen Zugang zu Serverräumen, sondern ist ausschließlich von einer funktionierenden Internetverbindung abhängig. Eine flexible IT führt schlussendlich zu geringeren Kosten, da benötigte Cloud Computing Services selbstständig auf-

und abgebaut bzw. im Fall von FaaS auf einer Pay-Per-Use Basis abgerechnet werden. Darüber hinaus bieten viele Cloud Provider Freikontingente an, wodurch anfangs teilweise keine Kosten anfallen. Die Plattformen der Anbieter ermöglichen Unternehmen, aber auch Privatpersonen per Weboberfläche Zugang zu einer Vielzahl von IT-Dienstleistungen. Diese können dabei mit wenigen Mausklicks erstellt werden und sind durch die integrierten Sicherheitsstandards des Cloud Providers geschützt.

Neben den zahlreichen Vorteilen von Cloud Services darf eine kritische Betrachtung nicht fehlen. Für einzelne Unternehmen kann ein direkter Zugriff auf die Hardware des Servers relevant sein, was ihnen hier jedoch verwehrt bleibt. Durch die Nutzung der Hardware bzw. spezifischen Dienstleistungen eines Cloud Providers besteht ein potenzielles Risiko eines Vendor Lock-In, wodurch die IT nicht zu einem anderen Anbieter umgezogen werden kann. Das Thema Sicherheit stellt sowohl einen Vor- als auch einen Nachteil dar. Aufgrund der Größe der Cloud Provider sind sie ein attraktives Ziel für Hackerangriffe. Sollten sich Unternehmen dazu entscheiden kritische Daten in der Public Cloud zu speichern, könnte dies schwerwiegende Folgen haben (Sehgal & Bhatt, 2018) (McHaney, 2021).

3.7 Cloud Computing in Vereinen

Sportvereine sind Institutionen, in denen unter anderem organisatorische Tätigkeiten anfallen. Eine Auswahl zeigt die folgende Abbildung. So müssen beispielsweise Anträge für die Aufnahme oder das Ausscheiden von Mitgliedern bearbeitet bzw. Kontaktdaten gepflegt werden. Meist gibt es feste Termine für Weihnachtsfeiern, Vorstandssitzungen oder Aufführungen, welche koordiniert und mit dem Raumbelegungsplan in Einklang gebracht werden müssen. Diese Liste könnte leicht erweitert werden (Golinsky, 2020).

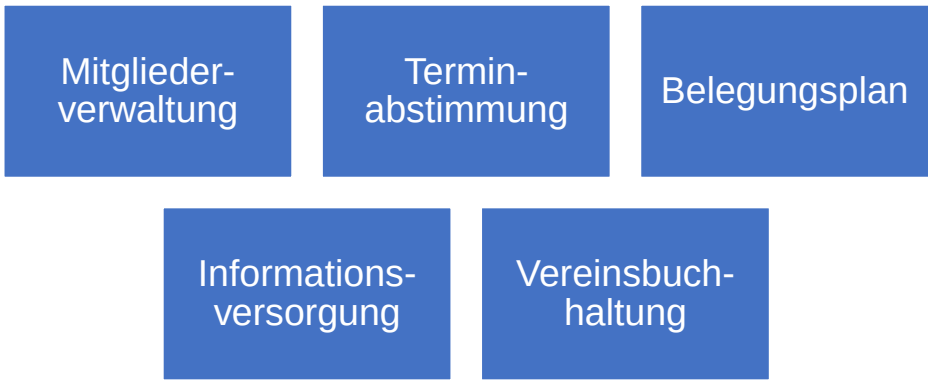


Abbildung 10: Organisatorische Tätigkeiten eines Sportvereins.

Auf den ersten Blick ist zu erkennen, dass viele dieser Tätigkeiten digitalisierbar sind. Beispielsweise könnte die Mitgliederverwaltung anstelle von Karteikarten durch ein System zur Verwaltung von Personendaten ersetzt werden. Auch Terminabstimmungen sind heutzutage mithilfe von online Umfragetools einfach und schnell zu erledigen.
Ein Verein muss in Form einer Strategie im Vorfeld definieren, welche Tätigkeiten mit IT unterstützt werden sollen. Falls lediglich Informationen über Kursangebote, Termine etc. für Mitglieder einsehbar sein sollen, so ist eine schlichte Website ausreichend. Liegt das Ziel in einer digitalen Mitgliederverwaltung sowie weiteren Funktionen, ist verstärkt über eine Webanwendung bzw. Software nachzudenken. Nachfolgend wird eine Sammlung von technischen Hilfsmitteln vorgestellt, welche in Vereinen zum Einsatz kommen könnten.
Für organisatorische Tätigkeiten wie Terminabstimmungen kann auf Online Tools wie doodle (doodle, o.D.) oder DFNTerminplaner (dfn, o.D.) zurückgegriffen werden. Sie bieten die Möglichkeit, Umfragen für Weihnachtsfeiern, Trainingszeiten und sonstige Veranstaltungen zu erstellen und meist mithilfe eines CSV-Exports in Tabellenkalkulationsprogrammen wie Microsoft Excel auszuwerten. Vorteilhaft ist, dass mit diesen Tools überwiegend kostenlos gearbeitet werden kann, es sich jedoch um eine alleinstehende Anwendung handelt und eine Vielzahl davon schnell zu Überforderung und Unübersichtlichkeit führt.
Anders sieht es mit einer integrierten Vereinsverwaltungssoftware wie Online-Vereinsverwaltung (online-vereinsverwaltung, o.D.), easyVerein (easyverein, o.D.) oder Vereinsplaner (vereinsplaner, o.D.) aus. Diese Komplettlösungen werden über das Internet den Mitgliedern des Vereins bereitgestellt und bieten mehrere Funktionen wie Mitgliederverwaltung, Terminplaner, Belegungsplan etc. in einer Software. Als Vorteil sticht hier vor allem die Integration von mehreren Funktionen in einer Anwendung sowie die Bereitstellung durch den Drittanbieter heraus. Dadurch entfällt das Verwalten von Soft- und Hardware durch Mitglieder des Vereins.
Ehrenamtliche Vereinsmitglieder können jedoch auch selbst für die notwendige IT sorgen. Um den Verein zu präsentieren und bestehende sowie neue Mitglieder zu informieren, ist meistens eine Website oder eine Webanwendung mit zusätzlichen Funktionen im Einsatz. Diese kann z.B. mit php oder einem JavaScript Framework wie React oder Next.js programmiert werden. Programmierkenntnisse sind hingegen bei der Arbeit mit sogenannten Websitebaukasten wie beispielsweise jimdo (JIMDO, o.D.), wix (wix, o.D.) oder wordpress (wordpress, o.D.) nicht notwendig. Sie erfreuen sich immer größerer Beliebtheit, da ganze Webanwendungen weitestgehend ohne Programmierkenntnisse zusammengeklickt werden können. Eine Vielzahl an Plugins bieten hier zudem Erweiterungen für einen Online-Shop, Benutzerverwaltung, Social Media

Integration etc. an. Hierdurch ergeben sich viele Möglichkeiten, Prozesse oder das Webdesign individuell anzupassen.
Eine allgemeingültige Aussage, wie und welche digitalen Elemente bevorzugt werden sollen, kann nicht getroffen werden. Für diese Entscheidung müssen der geplante Anwendungsfall und die zur Verfügung stehenden Ressourcen wie das Know-how der ehrenamtlichen Mitglieder, zeitlicher Einsatz, Finanzen etc. in Betracht gezogen werden. Für den weiteren Verlauf dieser Arbeit ist der Anwendungsfall entscheidend, wenn Vereinsmitglieder selbst eine Website bzw. Webanwendung entwickeln. Anfallende Tätigkeiten sind hier nicht nur in der Programmierung des Frontend und der Backendlogik zu finden. Für einen reibungslosen Betrieb ist das Anschaffen, Aufsetzen, Konfigurieren und Warten eines Servers und ggf. einer Datenbank inklusive Laufzeitumgebung für die Programmlogik notwendig. Allerdings bietet gerade hier Cloud Computing attraktive Lösungen, um die notwendige Infrastruktur einer Anwendung zu verwalten. Zusätzlich bieten Cloud Provider vordefinierte Softwaremodule z.B. eine Benutzerverwaltung an, welche mit minimalem Aufwand in eine eigene Anwendung integriert werden kann.

Variante 1

Mithilfe von Infrastructure as a Service kann die Anschaffung teurer Hardware wie Server umgangen werden. Stattdessen werden virtuelle Server, Datenbanken, Load Balancer etc. per Mausklick gebucht und durch den Cloud Provider bereitgestellt. Hierbei entfällt jedoch nur der monetäre Aufwand für den Kauf der Hardware. Verwaltung und Pflege muss weiterhin selbst betrieben werden.

Variante 2

Die Entwicklung einer serverlosen Anwendung mithilfe von FaaS wie in Abschnitt 3.3.4 beschrieben, ermöglicht die Fokussierung auf die Entwicklung des Softwareprodukts. Die notwendige Infrastruktur und Bereitstellung der fertigen Anwendung kann somit auf den Cloud Provider übertragen werden, wodurch die Notwendigkeit der Anschaffung und Wartung von Servern, Datenbanken etc. sowie deren Skalierung entfällt. Zudem wirkt sich die pay per use Basis positiv auf die finanziellen Ausgaben eines Vereins aus. Beispielsweise könnten Funktionen einer Anwendung nur saisonal bzw. zeitlich begrenzt innerhalb eines Jahres benutzt werden. Eine klassische Infrastruktur in Form eines Servers müsste trotzdem 365 Tage zur Verfügung stehen, außer diese wird extra für bestimmte Aufgaben eingeschaltet – was jedoch nicht sehr realistisch scheint.

Für die Mitgliederverwaltung sowie ein Online Tool zur Terminabstimmung wird in den folgenden Kapiteln anhand einer praktischen Beispielanwendung aufgezeigt, wie Elemente des Cloud Computing und somit Variante 2 einzusetzen sind.

4 AWS als Cloud Provider

Vor Beschreibung der praktischen Arbeit werden nachfolgend die in der Software verwendeten AWS Services aufgezählt und anhand einer kurzen Beschreibung erläutert. Genauere Informationen zur Implementierung und Konfiguration der Dienste erfolgen im nächsten Kapitel.

4.1 AWS IAM

Der Service Identity and Access Management (IAM) ist zentraler Bestandteil eines jeden Accounts. Die Sicherheitseinstellungen werden durch ein Rechtekonzept bestehend aus policies, roles, groups und users feingranular organisiert. Ein grundlegendes Verständnis dieser Begriffe ist notwendig, um andere AWS Services nutzen zu können (Stigler, 2018).

4.1.1 user, group und role

Diese Begriffe stehen für ihre deutschsprachigen Übersetzungen Rollen, Gruppen und (Be-)Nutzer und sind im Kontext von AWS IAM unter dem Begriff der Identities zusammengefasst. Eine identity ist notwendig, um sich bei der AWS Plattform anmelden und mit Services interagieren zu können. Die erste identity besitzt ein Kunde von Amazon Web Services nach dem erfolgreichen Erstellen eines Benutzerkontos, welcher auch als root Account bezeichnet wird. Dieses Konto ist notwendig, um weitere Identities zu erstellen. So kann damit ein Nutzerkonto aufgebaut und einer weiteren Person Zugriff auf die Konsole von AWS gegeben werden. Auf Dienste des Cloud Providers kann über die Weboberfläche per Command Line Interface (CLI) oder Application Programming Interface (API) zugegriffen werden. Die letzten beiden Optionen sind besonders für Entwickler relevant und werden durch das Bereitstellen von Access Keys erreicht.
Mehrere Nutzer können innerhalb einer Gruppe zusammengefasst werden. Hintergrund dabei ist die Zuweisung von Richtlinien zu einer Vielzahl von Nutzerkonten, um dies nicht individuell vornehmen zu müssen. Wozu Richtlinien benötigt werden, wird im nächsten Abschnitt erläutert. Eine Rolle ist ebenfalls eine eigenständige identity, welche jedoch von jeder Person angenommen werden kann, um temporäre Berechtigungen auf AWS Ressourcen zu erhalten (AWS, docs.aws.amazon.com/IAM, o.D.). Teilweise benötigen Dienste selbst Ausführungsrollen, wie dies beispielsweise bei AWS Lambda der Fall ist.

Dort wird beim Anlegen einer neuen Lambda Funktion nach einer sogenannten ExecutionRole verlangt, da nur durch diese Rolle, welche mit speziellen Berechtigungen verknüpft ist, ein Zugriff auf einen anderen AWS Service möglich ist (AWS, docs.aws.amazon.com/lambda, o.D.).

4.1.2 policies

Im Deutschen kann dieser Begriff auch als „Richtlinie“ verstanden werden. Es gibt zahlreiche vordefinierte policies von Amazon Web Services. Allerdings besteht die Möglichkeit, diese auch individuell zu erstellen. Die Richtlinien lassen sich, wie in Abbildung 11 illustriert, in sechs Arten einteilen.

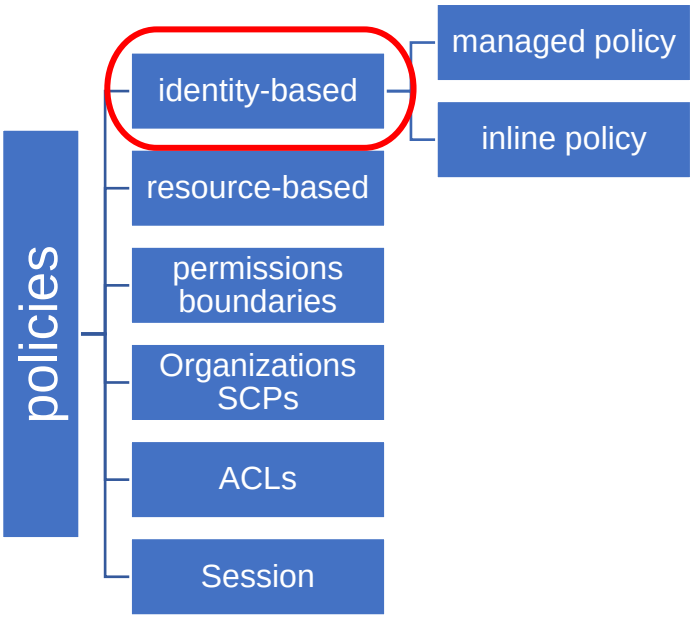


Abbildung 11: Die sechs policy Arten.

Für die folgenden Ausführungen sind die identity-based policies entscheidend, weshalb diese näher erläutert werden. Sie definieren, welche Rechte eine identity – user, group oder role – besitzt. Die Verzweigung in Abbildung 11 veranschaulicht die Unterteilung der identity-based policies. Eine inline policy wird direkt einer identity zugeordnet, wohingegen eine managed policy mehreren Identities zugewiesen wird. Grundsätzlich lässt sich festhalten, dass innerhalb einer Richtlinie definiert wird, welche Aktionen eine bestimmte identity an einem AWS Service tätigen darf (AWS, docs.aws.amazon.com/IAM, o.D.).
Einfache JavaScript Object Notation (JSON) Dateien dienen als Beschreibung einer Richtlinie (json.org, o.D.). Die grundlegende Struktur ist in Abbildung 12 verdeutlicht.

Abbildung 12: Struktur einer policy.

Die Version muss dabei immer angegeben werden und kann entweder „2008-10-17“ oder „2012-10-17“ lauten. Sie gibt an, welche Version und somit Syntax zur Verarbeitung der vorliegenden policy verwendet werden soll. Unter dem statement Abschnitt lassen sich eine oder mehrere Richtlinien definieren, die sich aus folgenden Bestandteilen zusammensetzen: Um eine optionale Beschreibung hinzuzufügen gibt es die statement ID (Sid), welche sowohl aus Buchstaben als auch aus Zahlen bestehen kann. Innerhalb eines JSON Arrays werden alle Rechte in der Form „service:action“ aufgelistet, welche die vorliegende policy adressieren soll. Ein weiterer wichtiger Bestandteil ist effect in Zeile sechs, welcher angibt, ob die aufgeführten actions genehmigt oder verweigert werden sollen. Als Ressource wird in AWS ein Service wie beispielsweise eine DynamoDB Datenbank bezeichnet und besitzt einen eindeutigen Amazon Resource Name (ARN). In Zusammenhang mit policies kann diese verwendet werden, um die im Action Teil spezifizierten Tätigkeiten auf ganz gezielte Ressourcen zu beziehen. Alternativ kann mit einem „*“ diese Einschränkung verhindert werden. Optional besteht im Abschnitt der condition die Möglichkeit, eine logische Unterscheidung einzufügen, in welcher Situation der statement Block angewendet werden soll (AWS, docs.aws.amazon.com/IAM, o.D.).

4.2 AWS Lambda

In Unterkapitel 3.3 wurden die unterschiedlichen Services im Bereich Cloud Computing erläutert. Innerhalb AWS ist die Kategorie der FaaS durch Lambda vertreten. Eine serverlose Applikation besteht meist aus einer Vielzahl dieser Funktionen und beinhaltet

die Programmlogik. Ausgeführt wird dieser Code nur, wenn ein Nutzer diesen anfordert. In diesem Fall stellt AWS die notwendige Infrastruktur und Ressourcen eines Servers bereit und berechnet daraufhin die angefallenen Kosten. Entwickler müssen sich demnach nicht um die Konfiguration von Servern und der notwendigen Umgebung für einen reibungslosen Ablauf der Funktion, sondern einzig um die Businesslogik kümmern. Um die Größe der einzelnen Lambda Funktionen zu reduzieren, eine schnellere Bereitstellung zu ermöglichen sowie Programmcode zwischen Funktionen wiederverwendbar zu machen, gibt es das Konzept der Lambda Layers. Diese beinhalten meist einen gemeinsam verwendeten Code, Abhängigkeiten oder Bibliotheken, welche bei Erstellung mithilfe einer .zip Datei in einen S3 bucket gespeichert werden. Sobald eine Funktion aufgerufen und die notwendige Laufzeitumgebung, Umgebungsvariablen etc. zum Ausführen aufgebaut werden, wird automatisch der Programmcode aus dem S3 Speicherort für die Verwendung innerhalb der Lambda Funktion bereitgestellt (AWS, docs.aws.amazon.com/lambda, o.D.).

4.3 AWS DynamoDB

Amazon Web Services stellt eine serverlose No-SQL Datenbank in Form von AWS DynamoDB zur Verfügung. Für die Speicherung von Daten wird auf key-value Paare gesetzt. Daten werden dabei automatisch verschlüsselt in Tabellen abgelegt. Wie bei allen anderen serverlosen Services kümmert sich AWS komplett um die Verwaltung, Skalierung etc. der Datenbanken innerhalb des verteilten Netzwerkes (AWS, docs.aws.amazon.com/amazondynamodb, o.D.).

4.4 AWS API Gateway

Daten, welche in einer DynamoDB Tabelle gespeichert und mithilfe von Lambda Funktionen verarbeitet werden, müssen beispielsweise für Frontend Applikationen erreichbar sein. Hierbei hilft die Schnittstelle des sogenannten API Gateway, welche für die Erstellung von skalierbaren REST, HTTP oder WebSocket APIs genutzt werden kann. AWS stellt verschiedene Ansätze bereit, um den Zugriff auf REST API’s zu steuern. Eine Möglichkeit sind eigens geschriebene Lambda Funktionen, welche den Zugriff steuern und deshalb auch als Lambda authorizer bezeichnet werden. Im nächsten Absatz wird auf den Dienst Cognito eingegangen. Falls dieser in der Anwendung verwendet wird, können die Endpunkte der API auch mithilfe eines von AWS verfügbaren Cognito authorizer abgesichert werden (AWS, docs.aws.amazon.com/apigateway, o.D.).

4.5 AWS Cognito

Cognito ist ein Service zur mobilen Benutzerverwaltung. Ein zentrales Element ist der sogenannte user pool, welcher mehrere Funktionen besitzt. Zum einen bietet er Benutzern von mobilen und Webanwendungen die Möglichkeit der Registrierung und Autorisierung. In einem user pool werden somit alle Nutzerdaten gespeichert. Zum anderen stellen user pools eine eigene identity im Sinne einer AWS Anwendung dar und übernehmen dabei die Aufgabe des Token Management mit dem Frontend. Innerhalb eines user pools besteht die Möglichkeit, Gruppen zu definieren und Benutzer einzuordnen, wodurch beispielsweise Privilegien für Administratoren und normale Nutzer unterschieden werden können. Ein weiterer Bestandteil von Cognito ist das Konzept der identity pools, welches den Zugriff auf AWS Ressourcen steuert. Die Idee besteht darin, dass sich ein Benutzer am user pool einloggt und einem identity pool den generierten Token übergibt. Daraufhin erhält dieser Berechtigungen, um auf vordefinierte Dienste von Amazon Web Services zuzugreifen. User pools werden in Cognito immer benötigt, wohingegen identity pools nicht zwingend notwendig sind. Der Zugriff auf die Benutzerverwaltung aus dem Frontend heraus wird über das Erstellen eines sogenannten App Client erreicht, welcher berechtigt ist, die Funktionen zur Registrierung, Login etc. für den entsprechenden user pool aufzurufen. Durch die Verwendung einer Client ID und eines Client Secret innerhalb der Webanwendung können somit die entsprechenden Operationen aufgerufen werden (AWS, docs.aws.amazon.com/cognito, o.D.).

4.6 AWS S3

Simple Storage Service – kurz S3 – kann zur Speicherung von Objekten jeglicher Art wie beispielsweise Archiven, Backups, Webseiten etc. verwendet werden. Hierfür muss lediglich ein sogenannter bucket, also ein Container für Daten in einem initialen Schritt erstellt werden (AWS, docs.aws.amazon.com/AmazonS3, o.D.).

4.7 AWS CloudFormation

Mit der Hilfe von DynamoDB, Lambda etc. entfällt der Kauf und die Einrichtung von physischer Hardware, wie beispielsweise einem Server. Allerdings ist bei der Entwicklung einer Anwendung die Verknüpfung einzelner AWS Services notwendig. Diese Aufgabe kann mithilfe von Terminalbefehlen oder der Weboberfläche der Amazon Plattform vorgenommen werden. Bei großen und sich verändernden IT-Infrastrukturen mit vielen Ressourcen kann dies schnell zu Unübersichtlichkeit führen. AWS CloudFormation bietet hier Abhilfe, indem die notwendigen Ressourcen für eine serverlose Anwendung in einem einzigen JSON oder YAML Template definiert werden,

wie die nachfolgende Abbildung auf der linken Seite illustriert. Dieses Konzept ist auch besser bekannt als Infrastructure as Code.

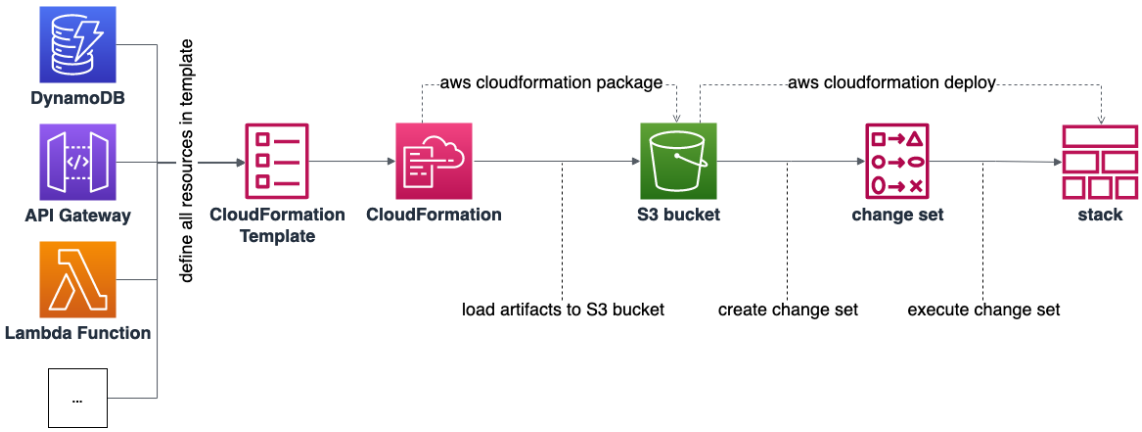


Abbildung 13: Aufbau eines Stacks mit CloudFormation.

Nachdem alle Ressourcen innerhalb des CloudFormation Template beschrieben wurden, entsteht durch das Ausführen von wenigen Kommandozeilenoperationen ein sogenannter Stack, welcher aus den beschriebenen AWS Ressourcen auf der linken Seite der Abbildung 13 besteht. CloudFormation übernimmt dabei den Aufbau und die Verknüpfung der Dienste untereinander und achtet auf deren Abhängigkeiten während der Erstellung (AWS, docs.aws.amazon.com/AWSCloudFormation, o.D.). Die notwendigen Terminalkommandos sehen hierbei folgendermaßen aus:

```
1  aws cloudformation package \
2      --template-file <template name> \
3      --output-template-file template.packaged.yml \
4      --s3-bucket <bucket name>
```

Abbildung 14: package Befehl für CloudFormation. Quelle: In Anlehnung an (AWS, docs.aws.amazon.com/cli, o.D.)

Durch das Ausführen des Kommandos in Abbildung 14 werden alle im Template definierten Artefakte wie Code für Lambda Funktionen und benötigte Ressourcen gebündelt und in einem S3 bucket abgelegt. (AWS, docs.aws.amazon.com/cli, o.D.)

```
aws cloudformation deploy \
    --template-file template.packaged.yml \
    --stack-name <stack name> \
    --parameter-overrides <optional environment variables>
```

Abbildung 15: deploy Befehl für CloudFormation. Quelle: In Anlehnung an (AWS, docs.aws.amazon.com/cli, o.D.)

Anschließend folgt der Befehl in Abbildung 15, wodurch in einem ersten Schritt ein sogenanntes change set erstellt wird. Dabei gleicht CloudFormation den bestehenden Stack, falls vorhanden, mit dem neu zu erstellenden Stack ab und teilt Informationen über Änderungen mit. Schlussendlich werden die Änderungen durchgeführt, indem AWS Ressourcen erstellt, löscht oder abändert. Damit dies reibungslos abläuft, müssen den Benutzern die entsprechenden Berechtigungen innerhalb der policy über IAM erteilt werden (AWS, docs.aws.amazon.com/cli, o.D.).

4.8 AWS SAM

Wie in Unterkapitel 4.7 beschrieben, bietet AWS CloudFormation die Möglichkeit, mit einem Template zu arbeiten und darin alle benötigten Ressourcen zu definieren. Diese Datei kann aufgrund von umfangreichen Spezifikationen für die Beschreibung und Konfiguration der Ressourcen schnell unübersichtlich werden. Deshalb wurde das Open Source Framework namens Serverless Application Model (SAM) entwickelt. Hiermit lassen sich serverlose Anwendungen ebenfalls mithilfe einer JSON oder YAML Datei erstellen. Da es sich bei SAM um eine Erweiterung von AWS CloudFormation handelt, können sowohl AWS SAM als auch CloudFormation spezifische Ressourcen in der JSON oder YAML Datei verwenden. Auf die Inhalte des Templates wird in einem späteren Kapitel genauer eingegangen. Um jedoch den Hintergrund zu verdeutlichen, folgendes Beispiel: Innerhalb einer CloudFormation YAML Datei muss zum Erstellen einer Lambda Funktion der Typ „AWS::Lambda::Function“ angegeben werden. Dasselbe Resultat kann innerhalb einer SAM Datei mit dem Typ „AWS::Serverless::Function“ erreicht werden. Da es sich bei SAM um eine Erweiterung von CloudFormation handelt, wird im Hintergrund aus „AWS::Serverless::Function“ eine „AWS::Lambda::Function“ generiert. Die folgende Abbildung 16 ähnelt der aus dem vorangegangenen Abschnitt. Da SAM allerdings als Abstraktionsschicht für CloudFormation angesehen werden kann, ist ein zusätzlicher Schritt in Form des sam build Kommandos notwendig.

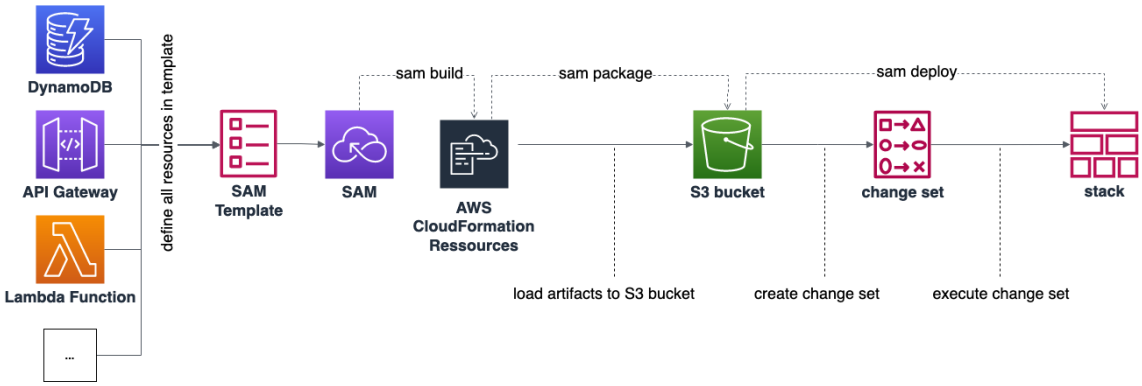


Abbildung 16: Aufbau eines Stacks mit SAM.

Das eben vorgestellte Kommando ist in nachfolgender Grafik aufgeführt und bewirkt die Umwandlung der im SAM Template definierten Ressourcen zu AWS Services, welche CloudFormation kennt. Zusätzlich werden weitere Artefakte wie Programmcode für die nächsten Schritte vorbereitet.

```
1    sam build --template-file <template name>
```

Abbildung 17: build Befehl für AWS SAM. Quelle: In Anlehnung an (AWS, docs.aws.amazon.com/serverless-application-model, o.D.)

Ähnlich dem package Befehl in CloudFormation verpackt das Kommando in Abbildung 18 den Programmcode sowie die Abhängigkeiten der einzelnen Lambda Funktionen in einzelne .zip Dateien und legt diese in einem S3 bucket ab.

```
1  sam package \
2    --template-file <template name> \
3    --output-template-file template.packaged.yaml \
4    --s3-bucket <bucket name>
```

Abbildung 18: package Befehl für AWS SAM. Quelle: In Anlehnung an (AWS, docs.aws.amazon.com/serverless-application-model, o.D.)

Abschließend wird eine serverlose Applikation mithilfe der deploy Anweisung in Abbildung 19 erstellt und der Aufbau bzw. das Update eines CloudFormation Stacks initiiert.

```
sam deploy \
  --template template.packaged.yaml \
  --stack-name <stack name> \
  --capabilities CAPABILITY_IAM \
  --s3-bucket <bucket name> \
  --parameter-overrides <optional environment variables>
```

Abbildung 19: deploy Befehl für AWS SAM. Quelle: In Anlehnung an (AWS, docs.aws.amazon.com/serverless-application-model, o.D.)

Besonders für Entwickler ist AWS SAM von Vorteil, da benötigte Cloud Computing Dienste in einer zentralen Datei gebündelt beschrieben sowie konfiguriert werden und im Gegensatz zu CloudFormation die lokale Entwicklung möglich ist (AWS, docs.aws.amazon.com/serverless-application-model, o.D.).

4.9 AWS Services und deren Kosten

Bei erstmaliger Erstellung eines AWS Accounts startet eine einjährige Phase, welche mehrere kostenlose Services enthält. Erst nach Ablauf dieser Zeit kann es zu anfallenden Kosten für den weiteren Gebrauch kommen. Nachfolgende Tabelle zeigt den Zusammenhang zwischen Freikontingenten und anfallenden Kosten für die Dienste der vorherigen Unterkapitel auf.

Service	Freikontingent pro Monat	kostenlos	Kosten
Lambda	Aufrufe: 1 Millionen Rechenzeit: Bis zu 3,2 Millionen Sekunden	immer	*Aufrufe:* 0,20 USD pro 1 Mio. Anforderungen* *Rechenzeit:* 0,0000000021 USD pro 1 ms**
API Gateway	1 Millionen Aufrufe	12 Monate	*Für 333 Millionen Aufrufe:* 3,70 USD pro Monat
DynamoDB	25 GB Speicher	immer	*Schreibanforderungseinheiten:* 1,525 USD pro Million *Leseanforderungseinheiten:* 0,305 USD pro Million *Speicher:* 0,306 USD pro GB-Monat
Cognito	50 000 MAUs***	immer	50 001–100 000 MAUs: 0,0055 USD
S3	Speicher: 5 GB Put-Anforderungen: 2 000 Get-Anforderungen: 20 000	12 Monate	*Standard Storage:* Erste 50 TB/Monat: 0,0245 USD pro GB *PUT-, COPY-, POST-, LIST-Anforderungen:* 0,0054 USD pro 1.000 Anforderungen *GET-, SELECT- und alle anderen Anforderungen:* 0,00043 USD pro 1.000 Anforderungen
CloudFormation	1 000 Handler-Vorgänge	immer	über 1 000 pro Monat: 0,0009 USD pro Handler-Vorgang

* Berechnung auf einer x86 Architektur
** Berechnung auf Basis eines 128 MB Arbeitsspeichers
*** MAU = Monthly Active Users
**** bei Überschreitung des Freikontingents oder nach Ablauf der 12 Monate
(Region der Infrastruktur: Europa - Frankfurt)

Abbildung 20: Aufstellung ausgewählter AWS Dienste und deren Kosten.

Die Tabelle verdeutlicht, dass viele der Dienste, welche im Rahmen dieser Arbeit verwendet werden, dauerhaft oder zumindest für die ersten 12 Monate kostenlos sind. Vorausgesetzt, das Freikontingent wird nicht überschritten (AWS, aws.amazon.com/de/free, o.D.). Um einen Eindruck bzw. einen Referenzwert für den

Verbrauch zu erhalten, zeigt die nachfolgende Abbildung einen Auszug aus den Verbrauchswerten eines Monats (MTD) (Stand: 13.02.2022) während der Entwicklung der Anwendung.

Service	AWS Free Tier usage limit	MTD actual usage %
Amazon Simple Storage Service	5 GB of Amazon S3 standard storage	0.95%
Amazon Simple Storage Service	2,000 Put, Copy, Post or List Requests of Amazon S3	0.65%
Amazon Simple Storage Service	20,000 Get Requests of Amazon S3	0.46%
Amazon API Gateway	1 Million API Calls per month of Amazon API Gateway	0.05%
AWS Lambda	1,000,000 free requests per month for AWS Lambda	0.03%
Amazon Cognito	50,000 MAUs each month for Amazon Cognito	0.02%
AmazonCloudWatch	5 GB of Log Data Archive for Amazon Cloudwatch	0.01%
AWS Lambda	400,000 seconds of compute time per month for AWS Lambda	0.00%
AmazonCloudWatch	5 GB of Log Data Ingestion for Amazon Cloudwatch	0.00%
Amazon DynamoDB	25 GB of Storage for Amazon DynamoDB (EUC1)	0.00%

Abbildung 21: Verbrauch ausgewählter AWS Dienste innerhalb eines Monats für diese Anwendung.

5 Praktische Umsetzung

Angelehnt an die bereits vorgestellten Softwareentwicklungsphasen werden in diesem Kapitel die Tätigkeiten beschrieben, welche für die Entwicklung der Anwendung innerhalb dieser Arbeit notwendig waren. Aufgrund persönlicher Erfahrungen als Betreuer der Jugend innerhalb eines örtlichen Tennisvereins werden für die Beschreibung der Phasen hin zur Implementierung eigene Erfahrungen verwendet. Diese Erkenntnisse waren zudem ausschlaggebend für die Entwicklung der Bachelorarbeit. Die Test- sowie Wartungsphase deckt der Rahmen dieser Bachelorarbeit nicht ab, weshalb hierzu keine eigenen Kapitel bestehen.

5.1 Initialisierungsphase

Vor der Überlegung zu einem Zielsystem oder dessen Umsetzung wird an dieser Stelle die Ist-Aufnahme durchgeführt. Zur Planung von Tennisspielen während einer Saison,

den Jugendvereinsmeisterschaften oder auch Abfragen von Trainingszeiten wurde in den letzten Jahren auf online Umfrageanwendungen wie Doodle oder DFNTerminplaner zurückgegriffen. Nach dem Erstellen einer Umfrage wurde ein entsprechender Link zur Teilnahme per E-Mail an die betroffenen Personen versand. Aufgrund von Bedenken bezüglich des Datenschutzes wurde das Einsehen aller Umfrageergebnisse für die Teilnehmer gesperrt.

Bei Analyse dieses Ist-Zustands sind zahlreiche Aspekte aufgefallen, welche den reibungslosen Ablauf stören. Hierzu zählen z.B. Eintragungen ohne Nachnamen, wodurch eine Zuordnung der Teilnehmer und die damit verbundene Einteilung zu Teams, Gruppen etc. erschwert wird. Durch den Versand eines Links für eine Umfrage bestand für Teilnehmende zudem keine Notwendigkeit einen Account bei den online Anwendungen zu besitzen. Nachteil hierbei ist, dass nach einer einmaligen Teilnahme keine Abänderung der Daten möglich, sondern eine Neuabstimmung notwendig ist. Eine weitere Möglichkeit der Abänderung des Umfrageergebnisses bestand in der Information der Jugendbetreuer zum Beispiel per E-Mail, Telefon, SMS.

Diese und weitere Aspekte führen zwangsläufig zu Verbesserungsvorschlägen, welche an dieser Stelle nicht ausführlich beschrieben, sondern in Form eines Sollkonzepts vorgestellt werden. Dieses besteht in der Entwicklung einer mobilen Applikation, welche als Plattform dient und in einem ersten Schritt zur Durchführung von Umfragen verwendet werden kann. Eine angebundene Benutzerverwaltung im Hintergrund stellt hierzu eine Registrierungs- und Anmeldemöglichkeit und somit zentrale Datenhaltung bereit. Eine Erweiterung der Plattform ist dabei jederzeit möglich.

5.2 Definitionsphase

Das in Unterkapitel 5.1 angestrebte Szenario wird nun in einzelne, überschaubare Abschnitte zerlegt und mithilfe von UML Use-Case-Diagrammen grafisch dargestellt.

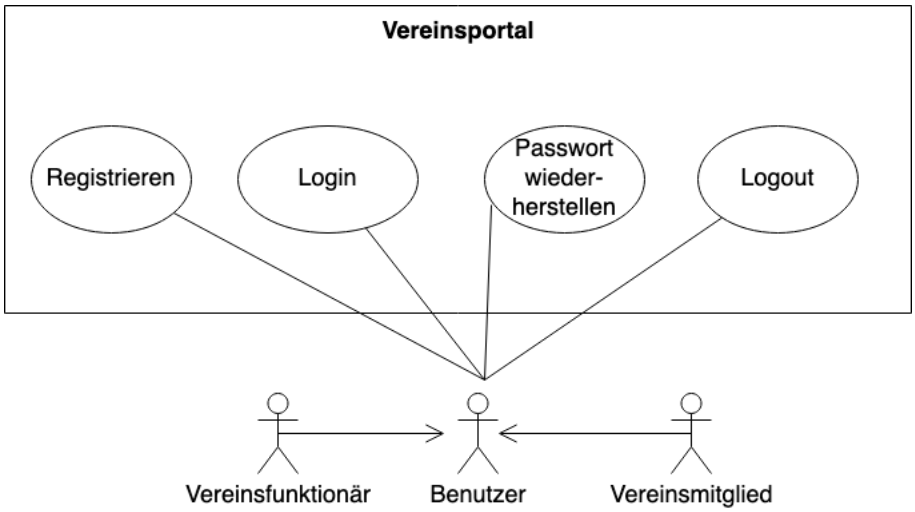


Abbildung 22: Use-Case-Diagramm des Vereinsportals.

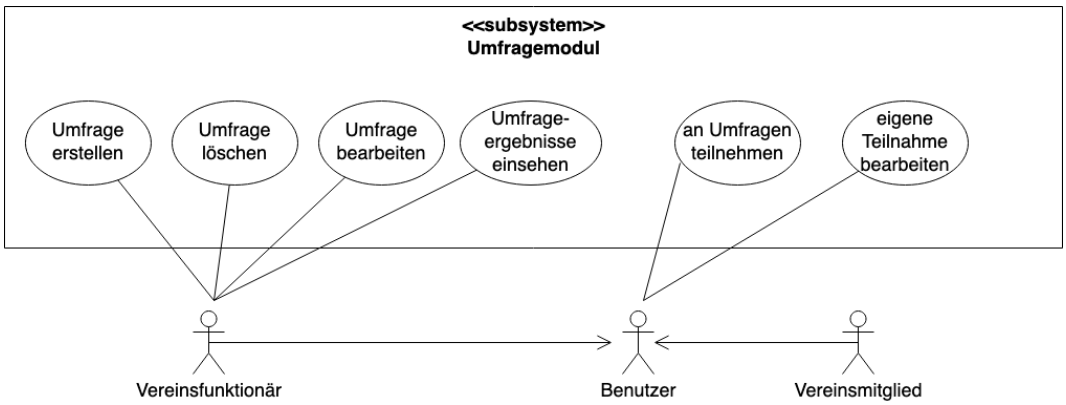


Abbildung 23: Use-Case-Diagramm des Umfragemoduls.

Jede Funktionalität wurde demnach in den kreisförmigen Formen der Abbildung 22 und Abbildung 23 erfasst. Bei den Benutzern des Systems wird eine Unterteilung in ein normales Vereinsmitglied und einen Vereinsfunktionär vorgenommen. Die Trennung ist notwendig, da nur Personen mit organisatorischen Tätigkeiten innerhalb des Vereins Zugriff auf zusätzliche Funktionen wie das Erstellen, Löschen, Bearbeiten und Auswerten von Umfragen besitzen sollen. Alle anderen Mitglieder haben hingegen die Möglichkeit an Umfragen teilzunehmen und ihre Teilnahme bei Bedarf zu bearbeiten.

5.3 Designphase

Nachdem ein allgemeingültiges Verständnis – sowohl textuell, also auch grafisch – für die zu erstellende Anwendung geschaffen wurde, rückt nun stärker die technische Umsetzung in den Mittelpunkt. Hierzu wurden wiederum aussagekräftige Grafiken angefertigt, um die Zielarchitektur für das Frontend der Angular Applikation, das Backend aus den AWS Services und deren Verknüpfung zu verdeutlichen.

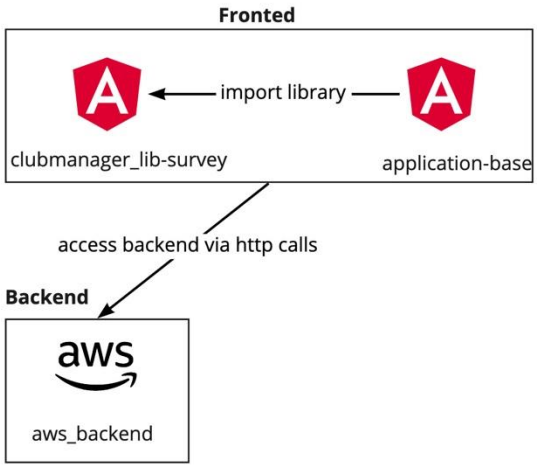


Abbildung 24: Makroarchitektur der kompletten Anwendung.

Grundlegend lässt sich die Zielarchitektur in das Frontend und Backend einteilen, wie im vorangestellten Bild zu erkennen ist. Ersteres besteht aus einem Angular Projekt, welches den Rahmen in Form von Registrierung, Login etc. und somit die Verwaltung der Nutzer bietet und als application-base bezeichnet wird. Der ausgesuchte Anwendungsfall zur Unterstützung von Terminabstimmungen ist mithilfe einer selbst erstellten Angular Bibliothek als Modul in das Rahmenprojekt integriert und wird im Folgenden als clubmanager_lib-survey bezeichnet. Die Datenhaltung etc. im Hintergrund wird durch ein AWS Backend realisiert und mithilfe von HTTP Aufrufen angesteuert.

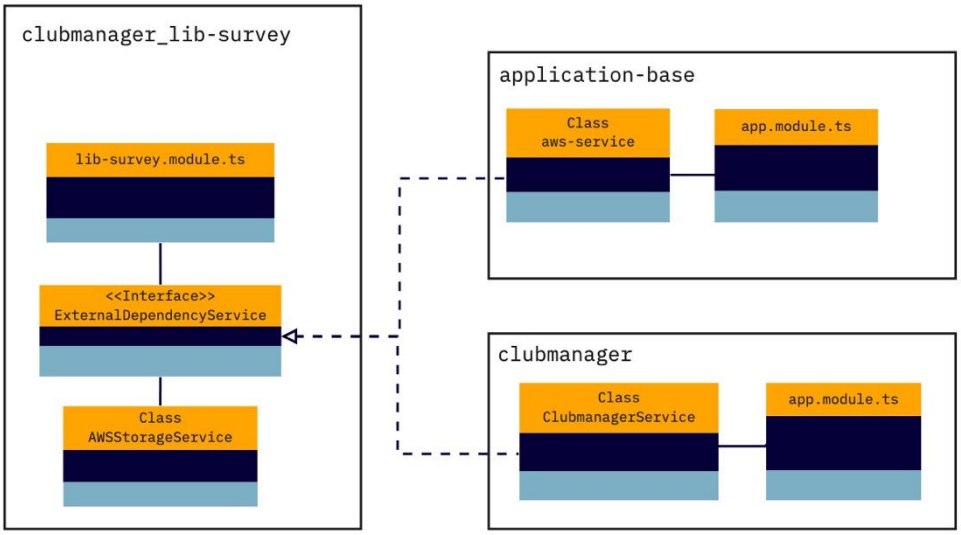


Abbildung 25: Mikroarchitektur für die Einbindung der Bibliothek in das Rahmenprojekt.

Die Integration der Bibliothek mit dem Umfragemodul in das Rahmenprojekt und somit die Mikroarchitekturebene stellt Abbildung 25 dar. Das clubmanager_lib-survey Projekt stellt das Interface ExternalDependencyService *(Anlage: Nr. 1)* nach außen bereit und kann somit von einer weiteren Angular Anwendung implementiert werden. Diese Schnittstelle ist zugleich für die Verknüpfung des Frontend und Backend zuständig, da hieraus die HTTP Aufrufe an AWS getätigt werden. Die Integration des Interfaces ist anhand des application-base Projekts angedeutet, jedoch ist dies auch mit anderen Angular basierten Programmen wie beispielsweise dem Online Vereinsverwaltungstool namens clubmanager (online-vereinsverwaltung, o.D.) möglich.

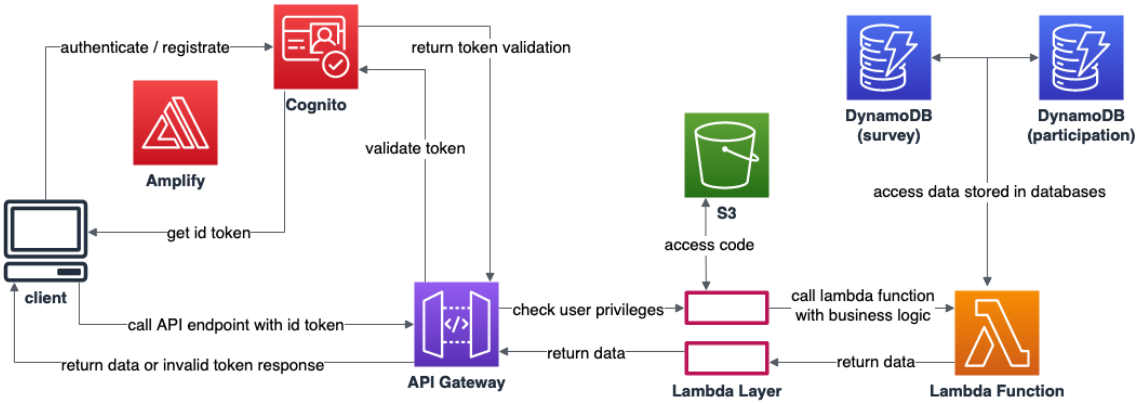


Abbildung 26: Makroarchitektur des AWS Backend.

Die vorliegende Abbildung 26 zeigt die Makroarchitektur des angestrebten AWS Backend. Zu erkennen sind mehrere miteinander verknüpfte Dienste des Cloud Providers. Der Client stellt den Einstiegspunkt dar und registriert bzw. authentifiziert sich bei dem Cognito Benutzerverwaltungsdienst, woraufhin mehrere Token generiert und an den Nutzer zurückgeliefert werden. Das abgebildete Amplify Logo verdeutlicht, dass dieses Framework als Schnittstelle zwischen dem Frontend und Cognito agiert. Mit einem der erhaltenen Token wird im Anschluss, beispielsweise durch einen Klick auf einen Knopf in der Oberfläche, ein Endpunkt des API Gateway aufgerufen. Bevor diese Anfrage weiterverarbeitet wird, prüft ein sogenannter authorizer der API den Token auf Validität, indem mit dem Cognito Service kommuniziert wird. Sollte dies nicht der Fall sein, so wird eine Fehlermeldung über einen unberechtigten Zugriff zurückgegeben. Im anderen Fall handelt es sich um einen authentifizierten Nutzer. Die Anfrage wird im nächsten Schritt weiter an die in der Anwendung integrierten Lambda layers weitergegeben, in denen unter anderem die Privilegien des Benutzers anhand der Gruppenzugehörigkeit aus dem Cognito user pool herausgefiltert werden. Im letzten Schritt wird die mit der Anfrage verknüpfte Lambda Funktion und somit die Businesslogik aufgerufen, welche auf die in den DynamoDB Datenbanken gespeicherten Inhalte zugreift und die Ergebnisse dem Nutzer bereitstellt.

5.4 Implementierungsphase

Alle bisher gesammelten Informationen werden nun zusammengetragen und zur Implementierung der Zielarchitektur herangezogen. Zuerst wird auf die ersten Schritte zur Vorbereitung eines AWS Accounts eingegangen. Darauf aufbauend folgen relevante Schritte für die Erstellung des serverlosen Backend. Abschließend wird das technische Design des Frontend erläutert und wie dieses mit den AWS Ressourcen verknüpft ist.

5.4.1 Aufsetzen und Vorbereiten des AWS Accounts

Um AWS Services nutzen zu können, ist es notwendig, einen AWS Account aufzusetzen. Hierfür ist eine Registrierung mit einem Passwort und Kontaktdaten inklusive Kreditkarteninformationen erforderlich. Nach erfolgreichem Abschluss dieses fünfstufigen Prozesses besteht ein Account bei AWS, welcher durch eine eindeutige Account-ID identifiziert werden kann. Zudem ist ein root user vorhanden, welcher die erste AWS identity darstellt und vollen Zugriff auf diesen Account besitzt. Eine Anmeldung mit diesem user ist nur unter Angaben der Account-ID, auch als Kontonummer bezeichnet, der E-Mail-Adresse und des selbst vergebenen Passworts möglich (AWS, docs.aws.amazon.com/IAM, o.D.). Aufgrund der umfangreichen Rechte und des

damit verbundenen Sicherheitsrisikos sollte dieser initiale Account nicht für tägliche, administrative oder anderweitige Tätigkeiten verwendet werden. Die Rechte des root user können nicht begrenzt werden, weshalb verschiedene Accounts aufgesetzt werden sollten. Um sich für administrative Tätigkeiten nicht als Account owner anmelden zu müssen, wird empfohlen, einen admin user zu erstellen. Auch dieser sollte nicht für den täglichen Bedarf verwendet, sondern hierfür ein normaler user in IAM angelegt werden. Die Zugangsdaten des root Accounts sollten hingegen nur im Notfall verwendet und deshalb sicher verwahrt werden. Für einen erhöhten Schutz des AWS Accounts ist es gute Praxis, den neu angelegten Nutzern Passwortregeln aufzuerlegen, sodass Kennwörter sicher zu gestalten sind. Ebenso kann eine Zwei-Faktor-Authentifizierung mithilfe einer mobilen App wie beispielsweise authy aktiviert werden, sodass beim Anmelden mit Benutzername und Passwort zusätzlich ein sechsstelliger Zahlencode angegeben werden muss. Im Allgemeinen folgt der interne Aufbau der Benutzer, Gruppen und Rollen dem in Abschnitt 4.1 beschriebenen AWS IAM Konzepts. Die Rechte der neu angelegten Benutzer sollten anhand des „least privilege“ Konzepts vergeben werden, sodass stets die minimal notwendigen Befugnisse bestehen. Anstatt einem Admin oder User allerdings direkt Rechte zuzuweisen, werden an dieser Stelle zwei Gruppen – mit der Bezeichnung admin und user – erstellt, in welche die Nutzer eingeordnet werden. Der admin Gruppe wird daraufhin die AWS managed policy AdministratorAccess zugeordnet und somit weitreichende Befugnisse erteilt. Allen normalen Benutzern der user Gruppe sollen restriktivere Zugriffsrechte besitzen, weshalb eine oder mehrere customer managed policies notwendig sind. Um die Übersichtlichkeit der Zugriffsrechte für die einzelnen Services zu wahren, wurden diese in mehrere Richtlinien aufgeteilt. Exemplarisch zeigt die folgende Abbildung die Rechte eines user in Zusammenhang mit AWS DynamoDB auf.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:createTable",
                "dynamodb:deleteTable",
                "dynamodb:describeTable",
                "dynamodb:listTables",
                "dynamodb:updateTable",
                "dynamodb:query",
                "dynamodb:scan",
                "dynamodb:describeLimits",
                "dynamodb:batchGetItem",
                "dynamodb:batchWriteItem",
                "dynamodb:conditionCheckItem",
                "dynamodb:deleteItem",
                "dynamodb:getItem",
                "dynamodb:putItem",
                "dynamodb:query",
                "dynamodb:scan",
                "dynamodb:updateItem"
            ],
            "Resource": "*"
        }
    ]
}
```

Abbildung 27: Richtlinie für DynamoDB.

Abbildung 27 *(Anlage: Nr. 2)* ist zu entnehmen, dass ein user grundlegende Tabellenoperationen als auch Rechte für die Arbeit mit einzelnen Tabelleneinträgen besitzt. Ähnlich wurde mit dem Aufbau der Rechte für die weiteren, in Kapitel 4 beschriebenen Services verfahren. Als letzten Schritt im Zusammenhang mit policies muss eine ExecutionRole für Lambda Funktionen erstellt werden. Im vorliegenden Fall setzte sich diese aus der AWS managed policy AWSLambdaBasicExecutionRole sowie zwei customer managed policies *(Anlagen: Nr. 3, 4)* zusammen. Diese waren für die von AWS bereitgestellte Richtlinie für Operationen an den Datenbanken sowie dem Cognito user pool notwendig.

Für die Vorbereitung der Entwicklung auf dem eigenen Computer war zudem noch die Installation der AWS CLI sowie der AWS SAM CLI erforderlich. Aufgrund von gut dokumentierten Schritt-für-Schritt Anleitungen wird auf die Beschreibung dieser Tätigkeiten verzichtet (AWS, docs.aws.amazon.com/de_de/cli, o.D.), (AWS, docs.aws.amazon.com/serverless-application-model, o.D.).

5.4.2 Aufbau des AWS Backend

Nachdem im Unterkapitel 5.4.1 alle grundlegenden Schritte erklärt wurden, um mit AWS sicher arbeiten zu können, beschreibt dieser Teil der Arbeit, wie das Backend für die Anwendung umgesetzt, welche Services eingesetzt und verknüpft wurden. Ein Überblick bietet die Abbildung 26 in Abschnitt 5.3. Für eine effizientere Entwicklung wurde das Open Source Framework AWS SAM für die Beschreibung der notwendigen AWS Services innerhalb der Anwendung verwendet. Die Dateistruktur für den Aufbau des Backend sieht dabei wie in Abbildung 28 aus.

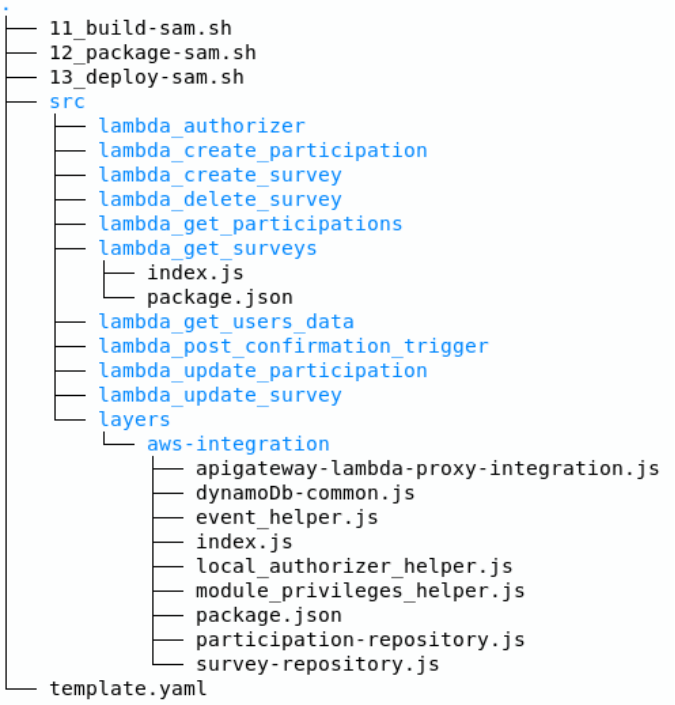


Abbildung 28: Dateistruktur des Programmcodes für das AWS Backend.

Die template.yaml Datei *(Anlage: Nr. 5)* bildet die Basis und wird mithilfe der Shellskripte, welche die Kommandos aus Unterkapitel 4.8 enthalten, zu einem Cloudformation Stack umgewandelt *(Anlagen: Nr. 6, 7, 8)*. Das /src Verzeichnis beinhaltet für jede Lambda Funktion einen Unterordner mit einer index.js Datei, welche die Businesslogik enthält sowie eine package.json Datei für das Auflisten der Abhängigkeiten. Um den Programmcode zu strukturieren, wurde das /layers/aws-integration Verzeichnis angelegt. Dort befinden sich Hilfsfunktionen wie beispielsweise das Entpacken von events *(Anlage: Nr. 9)* oder auch die Implementierung für die Datenbankoperationen *(Anlagen: Nr. 10, 11)*.

Der komplette Aufbau des AWS SAM Template kann im Rahmen dieser Arbeit nicht verdeutlicht werden, weshalb sich die nachfolgenden Aufführungen auf den Aufbau der

Cognito Benutzerverwaltung, einer DynamoDB, eines API Gateway und einer Lambda Funktion mithilfe von AWS SAM beschränken. Dadurch wird das Konzept der serverlosen Entwicklung deutlich und lässt sich auf umfangreichere Applikationen übertragen.

AWS DynamoDB

Die Beispielanwendung soll die Möglichkeit bieten, Terminabstimmungen digital durchzuführen. Deshalb ist die Speicherung von Umfragen und deren Teilnahmen innerhalb einer Datenbank notwendig. Eine AWS DynamoDB Ressource kann innerhalb des SAM Template mithilfe der folgenden Zeilen in Abbildung 29 *(Anlage: Nr. 5)* erreicht werden.

template.yaml

```
37    SurveysTable:
38      Type: AWS::DynamoDB::Table
39      Properties:
40        AttributeDefinitions:
41          -
42            AttributeName: "id"
43            AttributeType: "S"
44        KeySchema:
45          -
46            AttributeName: "id"
47            KeyType: "HASH"
48        BillingMode: "PAY_PER_REQUEST"
```

Abbildung 29: Definition einer DynamoDB Ressource innerhalb des SAM Template.

Wichtig ist an dieser Stelle die Angabe des Type Attributes, wodurch das SAM Framework erkennt, um welchen AWS Services es sich handelt. Unter den properties können anschließend Attribute der zu speichernden Objekte definiert werden. Da es sich bei DynamoDB um eine NoSQL Datenbank handelt, muss an dieser Stelle nur ein primary key angegeben werden, selbst wenn zu einem Umfrageobjekt noch weitere Attribute gespeichert werden sollen.

API Gateway

Um die erstellten Ressourcen und das Backend später über beispielsweise eine Angular Frontend Applikation ansprechen zu können, wird eine API für die Bereitstellung von Endpunkten benötigt. Diese ist in Form des API Gateway implementiert und wird im SAM Template, wie in Abbildung 30 *(Anlage: Nr. 5)* zu sehen, erstellt.

Abbildung 30: Definition eines API Gateway innerhalb eines SAM Template.

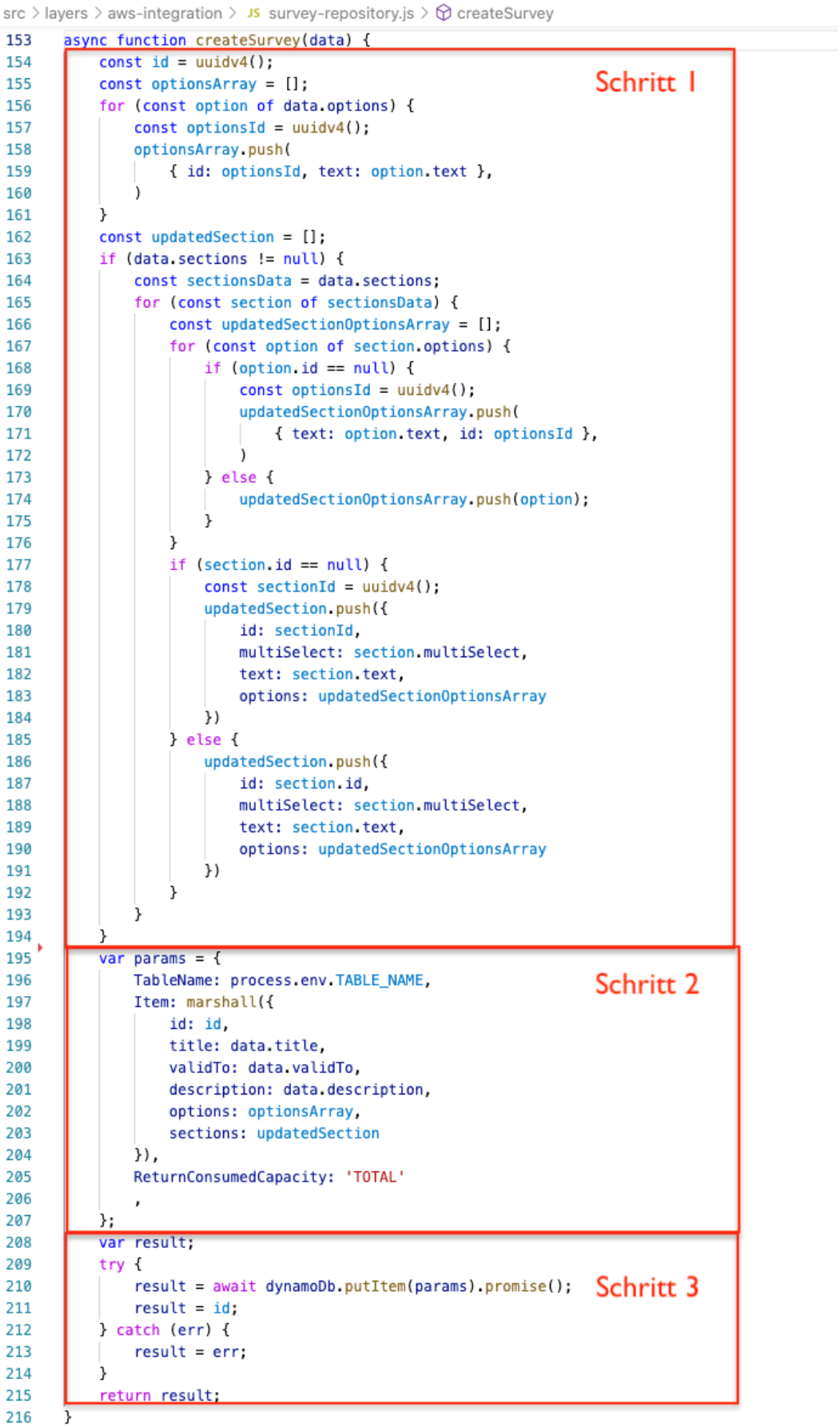
Auch hier muss der entsprechende Typ angegeben und unter dem properties Attribut die weitere Konfiguration vorgenommen werden. Wichtig ist hier der Auth Abschnitt, in dem definiert wird, wie die Endpunkte der API abgesichert werden sollen. Da die vorliegende Anwendung eine Cognito Benutzerverwaltung verwendet, kann an dieser Stelle auf den AWS spezifischen CogntioAuthorizer zurückgegriffen werden. Somit muss keine eigene Logik für das Prüfen von berechtigten Nutzern implementiert werden.

AWS Lambda

Den Resultaten der Definitionsphase ist zu entnehmen, dass Nutzer mit Admin-Rechten die Möglichkeit besitzen sollen, Umfragen zu erstellen. Hieraus lässt sich die Notwendigkeit einer Lambda Funktion ableiten, welche an dieser Stelle mit einer Laufzeitumgebung von node.js implementiert wurde. Die notwendigen Artefakte für die Funktion lassen sich unter /src/lambda_create_survey *(Anlage: Nr. 12)* sowie /layer/aws-integration finden (*Anlage: Nr. 13)*. Im ersten Ordner befindet sich die index.js Datei (*Anlage: Nr. 14)*, welche Abbildung 31 zeigt und den Einstiegspunkt der Lambda Funktion für das Erstellen einer Umfrage darstellt.

Abbildung 31: Programmcode der Lambda Funktion zum Erstellen einer Umfrage.

Charakteristisch ist hierbei der Export einer asynchronen Funktion, welche zum Schluss das Ergebnis an die aufrufende Stelle zurückgibt und als Parameter ein event, context und callback Objekt übergeben bekommt. Der restliche Programmcode kann individuell gestaltet werden, um diesen jedoch nicht unkommentiert zu lassen, wird dieser kurz erläutert. Funktion zwei wird verwendet, um das eintreffende event aus dem Frontend in eine standardisierte Form zu bringen. Hierfür werden Hilfsfunktionen aus dem /layers/aws-integration Verzeichnis *(Anlage: Nr. 9)* aufgerufen. Funktion drei beinhaltet den Verweis auf die Methode, welche sich auf das Erstellen der Umfrage bezieht. Bevor diese jedoch aufgerufen wird, ist eine Überprüfung der Nutzerrechte notwendig. Hierzu wird wiederum eine Methode aus dem /layers/aws-integration Ordner angesprochen *(Anlage: Nr. 15)*, um zu prüfen, ob der Nutzer ein normaler User oder Administrator ist. Diese Information wird der Abfrage der Gruppenzugehörigkeit eines Benutzers innerhalb des Cognito user pool entnommen. Wird die entsprechende Berechtigung bestätigt, so kann die createSurvey() Methode, welche in Abbildung 32 dargestellt ist, aus dem survey-repository *(Anlage: Nr. 11)* aufgerufen werden.




```
async function createSurvey(data) {
    const id = uuidv4();
    const optionsArray = [];
    for (const option of data.options) {
        const optionsId = uuidv4();
        optionsArray.push(
            { id: optionsId, text: option.text },
        )
    }
    const updatedSection = [];
    if (data.sections != null) {
        const sectionsData = data.sections;
        for (const section of sectionsData) {
            const updatedSectionOptionsArray = [];
            for (const option of section.options) {
                if (option.id == null) {
                    const optionsId = uuidv4();
                    updatedSectionOptionsArray.push(
                        { text: option.text, id: optionsId },
                    )
                } else {
                    updatedSectionOptionsArray.push(option);
                }
            }
            if (section.id == null) {
                const sectionId = uuidv4();
                updatedSection.push({
                    id: sectionId,
                    multiSelect: section.multiSelect,
                    text: section.text,
                    options: updatedSectionOptionsArray
                })
            } else {
                updatedSection.push({
                    id: section.id,
                    multiSelect: section.multiSelect,
                    text: section.text,
                    options: updatedSectionOptionsArray
                })
            }
        }
    }
    var params = {
        TableName: process.env.TABLE_NAME,
        Item: marshall({
            id: id,
            title: data.title,
            validTo: data.validTo,
            description: data.description,
            options: optionsArray,
            sections: updatedSection
        }),
        ReturnConsumedCapacity: 'TOTAL'
        ,
    };
    var result;
    try {
        result = await dynamoDb.putItem(params).promise();
        result = id;
    } catch (err) {
        result = err;
    }
    return result;
}
```

Abbildung 32: Programmcode zur Speicherung einer Umfrage in eine DynamoDB Tabelle.

In Schritt eins werden die eintreffenden Daten vorbereitet, indem z.B. eine eindeutige UUID für die Umfrage sowie deren Optionen generiert wird. Schritt zwei fasst notwendige Parameter wie den Tabellennamen und die zu speichernden Daten für die nächsten Schritte in einem Objekt zusammen. Die marshall() Methode wandelt dabei die Daten in für DynamoDB lesbare Strukturen um. Der abschließende Schritt zeigt schlussendlich, wie mithilfe des AWS SDK's die putItem() Methode der DynamoDB Instanz aufgerufen sowie die in Schritt zwei zusammengefassten Daten übergeben werden.
Nachdem zum einen erläutert wurde, wie eine Lambda Funktion bezüglich des Programmcodes aufgebaut werden kann und zum anderen verdeutlicht wurde, wo sich dieser innerhalb des konkreten Beispiels bzw. der Dateistruktur befindet, wird folgend auf den Aufbau dieser Ressource mithilfe von AWS SAM eingegangen.

Abbildung 33: Definition einer Lambda Funktion innerhalb eines SAM Template.

Abbildung 33 zeigt den relevanten Ausschnitt aus der template.yaml Datei *(Anlage: Nr. 5)*. Neben dem SAM spezifischen AWS::Serverless::Function Typ sind die Angaben zur Runtime, den Dateipfaden zum Lambda Programmcode und der layers relevant. Für einen reibungslosen Zugriff der Lambda Funktion auf andere Services ist zudem die Angabe der in Abschnitt 5.4.1 erstellten LambdaExecutionRole erforderlich. Optional ist hingegen der Block mit den environment Variablen, wo beispielsweise Tabellennamen referenziert werden können. Da diese Funktion nur ein Bestandteil aus einer größeren Anwendung darstellt und über eine API erreichbar sein soll, werden unter dem event

Abschnitt die Referenzen zum API Gateway hergestellt und so die Verknüpfung dieser beiden Services erreicht. Die restlichen Lambda Funktionen beinhalten abweichende Programmlogik, jedoch ist der Aufbau gleich strukturiert.

AWS Cognito

Eine Benutzerverwaltung ist notwendig, damit nur registrierte und angemeldete Nutzer auf das Umfragemodul des Portals Zugriff besitzen. Mithilfe weniger Konfigurationsschritte lässt sich der von AWS bereitgestellte Cognito Service erstellen.

Abbildung 34: Auszug aus der Konfiguration eines Cognito user pool innerhalb des SAM Template.

Vorangehende Grafik stellt einen Auszug aus dem Programmcode zum Aufbau eines Cognito user pool innerhalb der zu erstellenden Anwendung dar *(Anlage: Nr. 5)*. Hier wird unter anderem definiert, ob die Passwortwiederherstellung per E-Mail oder SMS erfolgen soll, ob eine Multifaktor Authentifizierung stattfindet, welche Passwortregeln gelten und welche Nutzerdaten während der Registrierung anzugeben sind.

Abbildung 35: Definition des Cognito user pool App Client sowie einer user pool Gruppe.

Um das Cognito Backend innerhalb der Frontend Anwendung benutzen zu können, wird in Zeile 391 bis 402 der Abbildung 35 ein App Client aufgebaut, dessen Client ID und Client Secret schlussendlich innerhalb des Amplify Frameworks referenziert werden. Ebenso zeigt diese Grafik exemplarisch den Aufbau einer Gruppe innerhalb des user pool.

Nachdem auf diese Art und Weise die Logik der Lambda Funktionen und die restlichen Ressourcen für die Anwendung in der template.yaml Datei definiert wurden, können die Shellskripte für den Aufbau eines Stacks bei AWS verwendet werden. Hierfür findet nacheinander die Ausführung der Befehle sam build, sam package und sam deploy statt. Das Terminal gibt Statusmeldungen aus, gleichzeitig beginnt das Framework die Umwandlung der Datei in ein CloudFormation Template und initiiert damit die Bereitstellung und Verknüpfung der Services. Neben den Ausgaben in der Konsole ist die Erstellung des Stacks auch unter dem CloudFormation Bereich der Weboberfläche zu sehen, wie dies Abbildung 36 zeigt.

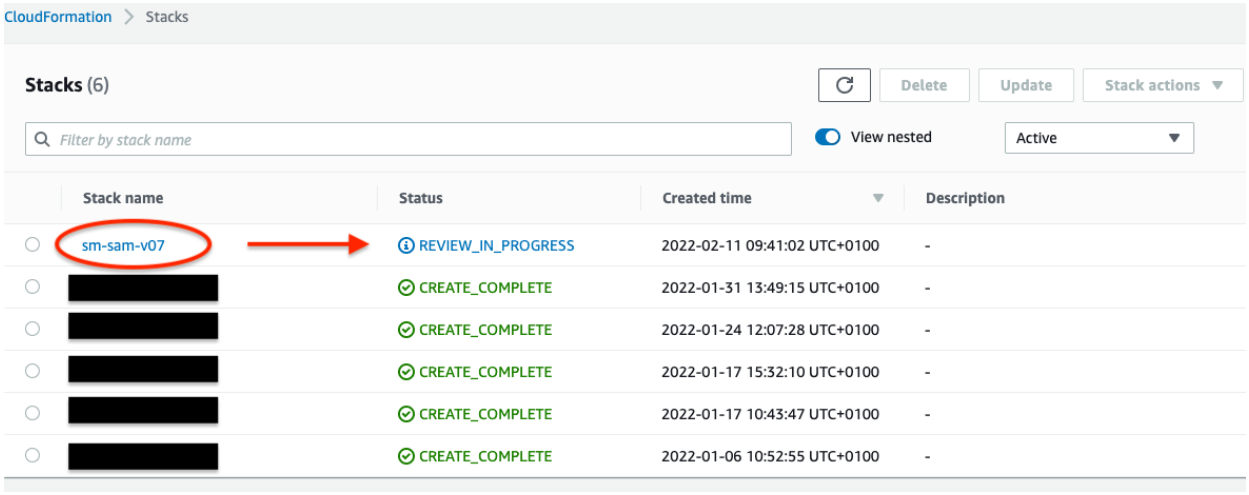


Abbildung 36: AWS Console zeigt Aufbau des Stacks.

Nach erfolgreichem Abschluss können die aufgebauten Ressourcen unter dem „Resource-Tab“ des jeweiligen CloudFormation Stacks eingesehen werden. Abbildung 37 zeigt dies anschaulich.

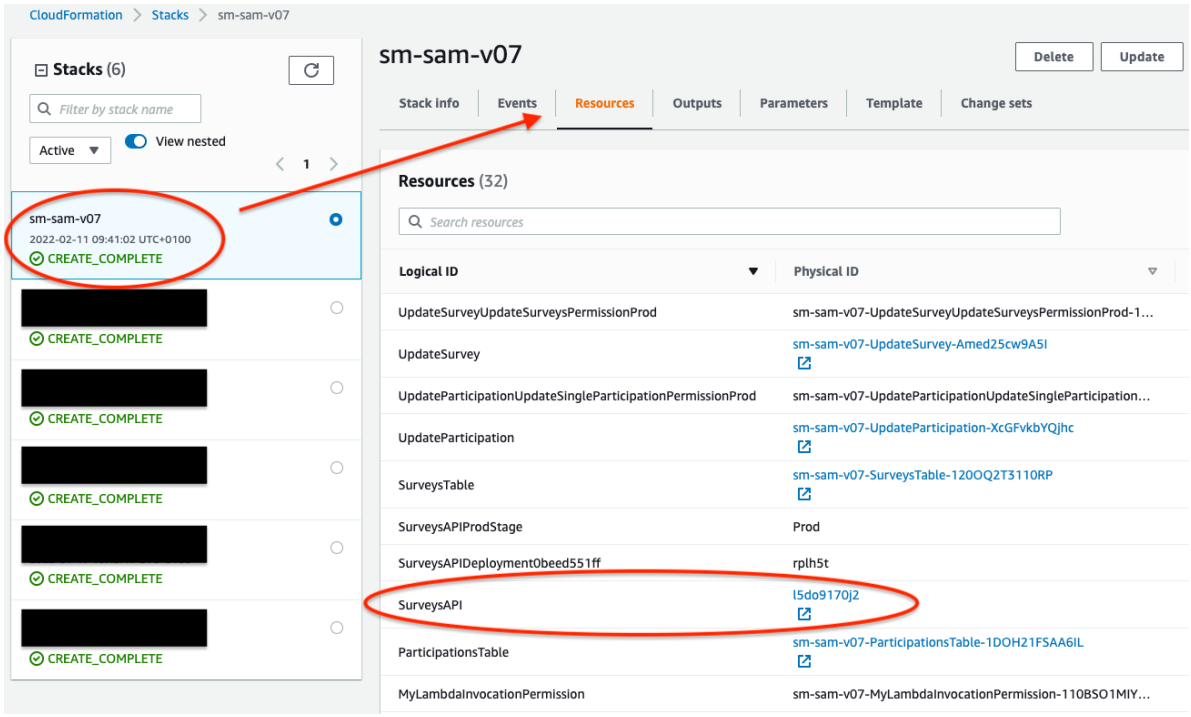


Abbildung 37: Auszug aus den Ressourcen des Stacks.

Aus dieser Ansicht heraus ist es beispielsweise möglich, das mit diesem Stack verknüpfte API Gateway via der Physical ID aufzurufen, woraufhin alle Endpunkte und die im template.yaml vorgenommen Konfigurationen eingetragen sind.

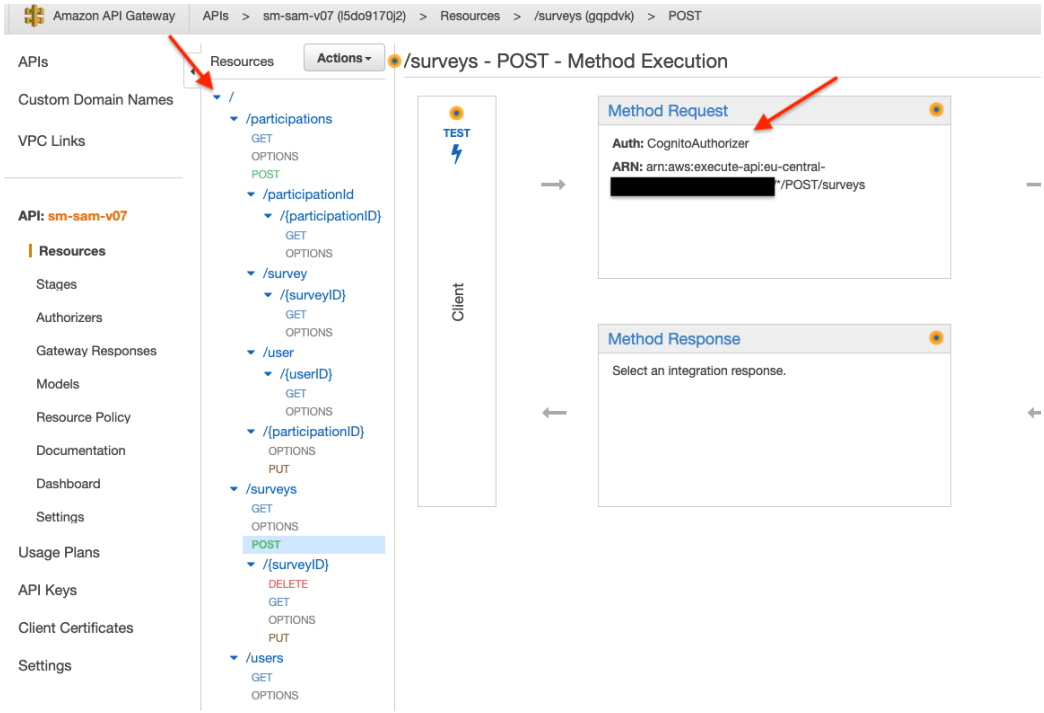


Abbildung 38: Aufgebautes API Gateway mit Verknüpfung zum Stack.

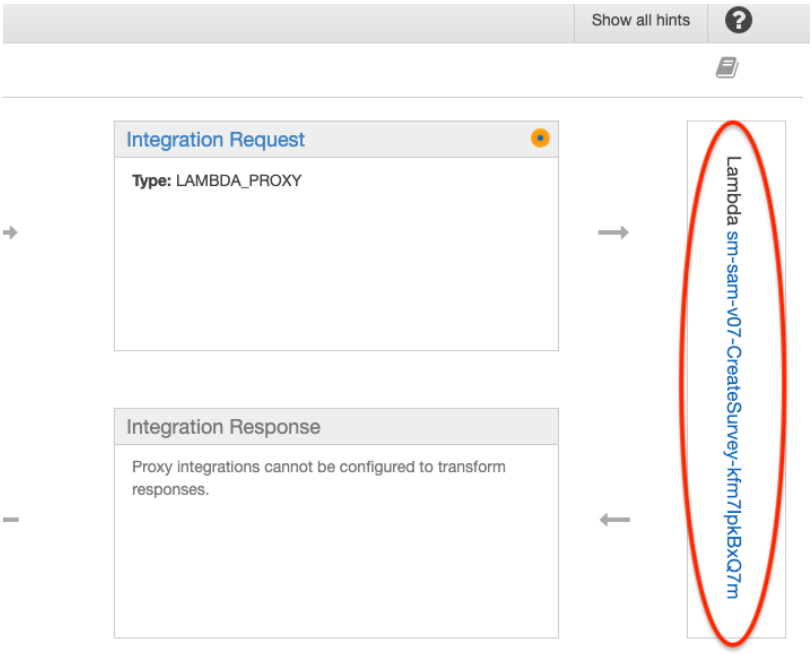


Abbildung 39: Aufgebautes API Gateway mit Verknüpfung zur Lambda Funktion.

Abbildung 38 zeigt die verfügbaren Endpunkte der API, welche im nächsten Abschnitt innerhalb der Angular Anwendung für Abfragen verwendet werden können. Zudem ist aus Abbildung 39 ersichtlich, dass unter dem /survey POST Endpunkt die Lambda

CreateSurvey Funktion referenziert ist. Die beiden Abbildungen sind zusammenhängend und wurden für eine übersichtlichere Darstellung getrennt.
In ähnlicher Weise sind die restlichen Lambda Funktionen, Tabellen sowie Bestandteile für die Cognito Benutzerverwaltung innerhalb der template.yaml Datei strukturiert.

5.4.3 Aufbau des Angular Frontend

Da der Fokus dieser Bachelorarbeit verstärkt auf dem Thema Cloud Computing liegt, werden in diesem Bereich nur die wichtigsten Punkte der Frontend Implementierung behandelt. Die grafische Oberfläche, welche mit dem Framework AngularJS umgesetzt wurde, soll Nutzern die Möglichkeit bieten, über einen Webbrowser auf das AWS Backend zuzugreifen und dieses nutzen zu können. Die Anwendung wurde in zwei Angular Projekte aufgeteilt. Projekt 1 beinhaltet die Implementierung des Umfragemoduls und Nummer 2 stellt den Rahmen für die Registrierung, Anmeldung etc. und somit die Verbindung zum Cognito Backend dar. Folglich wird die Einbettung des Angular Projekts 1 in andere Benutzerverwaltungssysteme ermöglicht.

Angular Projekt 1

Das Umfragemodul besteht aus vier Komponenten. Eines stellt dabei eine Übersicht bereit, um alle vorhandenen Umfragen anzeigen zu lassen *(Anlagen: Nr. 16, 17)*. Für das Erstellen neuer sowie für das Bearbeiten bestehender Umfragen steht eine Detailseite zur Verfügung *(Anlagen: Nr. 18, 19).* Teilnehmern einer Umfrage werden in einer dritten Komponente die Umfragen Inhalte angezeigt, wodurch eine neue oder die Bearbeitung der bestehenden Teilnahme möglich ist *(Anlagen: Nr. 20, 21).* Abschließend haben Vereinsfunktionäre die Option, alle Teilnahmen innerhalb einer Tabelle einzusehen und diese als CSV Datei zu exportieren *(Anlagen: Nr. 22, 23).* Die Ansicht eines Endbenutzers über den Browser ist in nachfolgender Abbildung 40 festgehalten. Ein Buttonklick durch den Benutzer im Browser initiiert einen HTTP Aufruf und somit die Ausführung einer referenzierten Lambda Funktion innerhalb des API Gateway aus dem AWS Backend.

Abbildung 40: Funktionen eines Administrators für das Umfragemodul.

Angular Projekt 2

Ein zweites Angular Projekt bildet den Rahmen, um das Modul der Umfragen nutzen zu können. Über eine Startseite gelangen Nutzer zu einer Registrierungsseite oder falls schon geschehen zu einer Loginmaske, wie in Abbildung 41 und Abbildung 42 zu sehen ist.

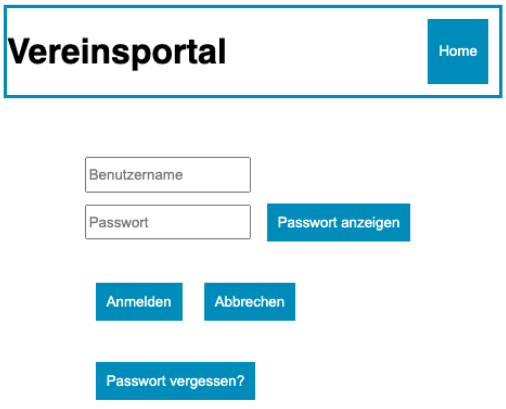


Abbildung 41: Loginmaske für das Vereinsportal.

Abbildung 42: Auszug aus dem Registrierungsprozess innerhalb der Angular Anwendung.

Die Verknüpfung der Registrierung bzw. Authentifizierung zwischen Frontend und Backend wird über das Amplify Framework hergestellt. Dazu muss ein npm Packet innerhalb der Angular Applikation installiert und die Daten des im AWS CloudFormation Stack referenzierten Cognito Backend eingetragen werden, wie Abbildung 43 *(Anlage: Nr. 24)* zeigt.

src > TS main.ts > ...

```ts
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

import Amplify from 'aws-amplify';


if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));

  Amplify.configure({
    Auth: {
      region: environment.region,
      userPoolId: environment.userPoolId,
      userPoolWebClientId: environment.userPoolWebClientId
    }
  })
```

Abbildung 43: Verknüpfung der Angular Frontend Applikation mit dem Cognito Backend.

Die notwendigen Daten sind dabei die Region, UserPoolID sowie UserPoolWebClienID, welche der Console in der AWS Weboberfläche entnommen werden können und auf dem Cognito App Client basieren. Abbildung 44 zeigt den Cognito Dienst innerhalb der AWS Console im Browser mit den gespeicherten Daten aus dem Registrierungsprozess.

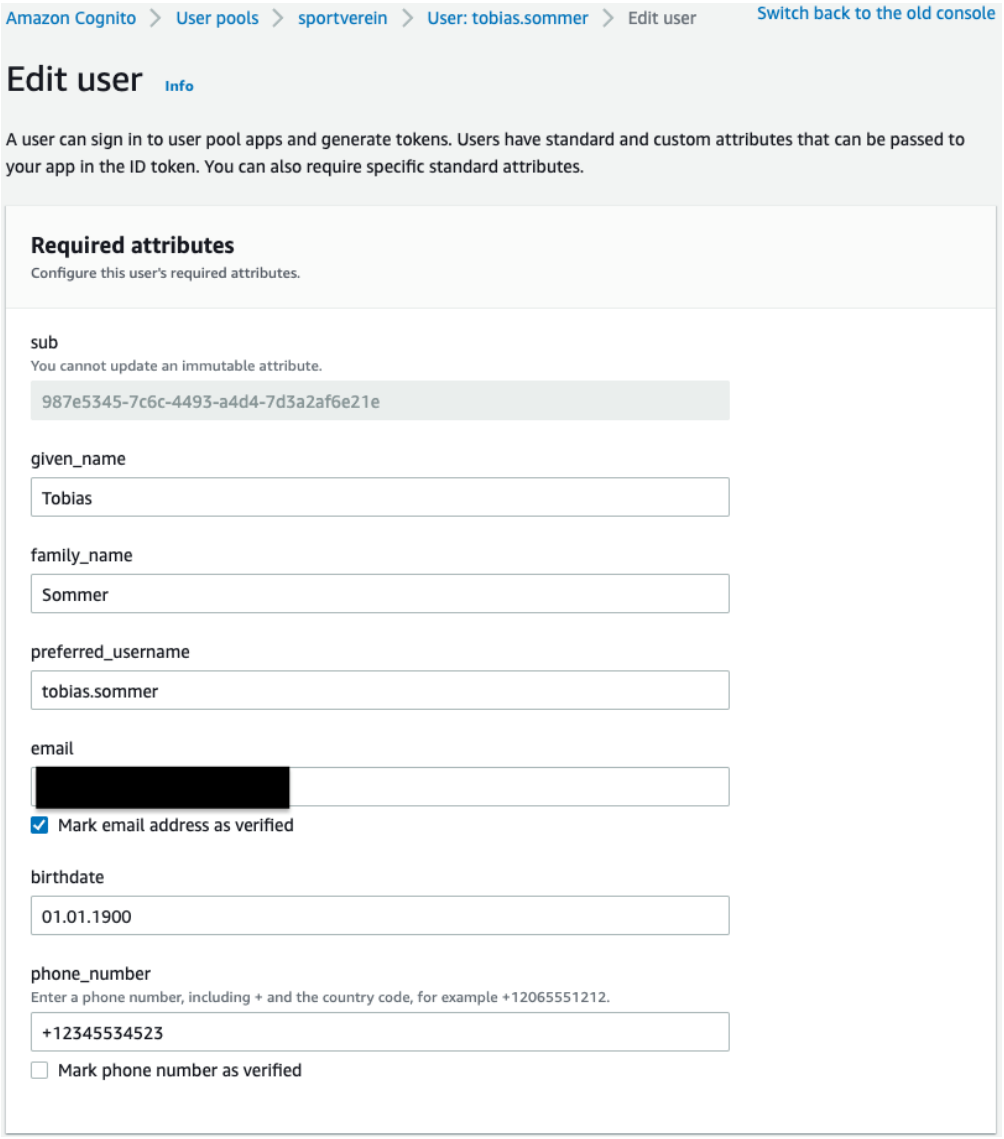


Abbildung 44: Gespeicherte Nutzerdaten in AWS Cognito.

Nach erfolgreicher Authentifizierung befindet sich der Benutzer auf einer Übersichtsseite, über welche der Logout oder das Umfragemodul aufgerufen werden kann.

6 Fazit

Die vorliegende Arbeit hatte zum Ziel, eine praxisnahe Anwendung auf Basis von Cloud Technologie zu entwickeln. Die durchgeführten Tätigkeiten wurden dabei auf das Rahmenwerk des Softwareentwicklungsprozesses gestützt, wodurch der Zusammenhang vom Beginn einer Idee bis hin zur fertigen Anwendung gut verdeutlicht werden konnte. Durch detaillierte Grafiken wurde zudem die Verknüpfung zwischen der Makro- und Mikroarchitektur aufgezeigt.

Anhand der Implementierung des serverlosen Backend konnte sowohl eine funktionsfähige Anwendung für die Vereinsorganisation erstellt als auch eine Hilfestellung für erste Berührungspunkte mit Cloud Technologie erzielt werden. Dazu wurde der Fokus auf die Dienste des AWS Cloud Provider gelegt, da es sich hierbei um einen der größten Anbieter handelt und aufgebautes Wissen in diesem Bereich für das spätere Berufsleben großen Mehrwert bieten kann.

Auch wenn Cloud Provider zahlreiche Dienste über ihre Plattformen der Öffentlichkeit zur freien Verwendung bereitstellen, ist die Einstiegshürde in dieses Themengebiet hoch. Es Bedarf an grundlegendem IT Wissen und ausreichend Einarbeitungszeit, um die Struktur der AWS Dokumentation sowie das zu Grunde liegende AWS IAM Konzept zu verstehen, bevor sicher mit der Entwicklung von Anwendungen begonnen werden kann. Gerade deshalb legt diese Arbeit besondere Aufmerksamkeit auf die Begriffe und Zusammenhänge der user, group, policy und role.

Potential für Vereine zur Digitalisierung ihrer Organisation besteht besonders dann, wenn ehrenamtliche Mitglieder grundlegende IT-Kenntnisse sowie eigenes Interesse an neuen Technologien mitbringen. Durch den Einsatz der AWS spezifischen Benutzerverwaltung Cognito steht eine out of the box Lösung bereit, welche mit wenig Aufwand mit einer Frontend Anwendung verknüpft werden kann. Besonders im Hinblick auf das freie Kontingent von 50 000 Monthly Active Users ist dies eine attraktive und kostengünstige Lösung. Die erstellte Anwendung im Rahmen dieser Bachelorarbeit hat während der Entwicklung keine Kosten verursacht und bleibt sogar noch weit hinter dem kostenlosen Kontingent zurück. Für die Zukunft wäre es daher interessant, den Umfang eines solchen Systems im produktiven Betrieb zu beobachten und die entstehende Kostenstruktur über einen längeren Zeitraum zu analysieren. Für Vereine bietet Cloud Computing eine interessante Möglichkeit, ihre Kosten für IT zu reduzieren, da entweder Infrastruktur in Form von IaaS gemietet oder in Form von Serverless Anwendungen auf einer Pay-Per-Use Basis genutzt werden kann. Bei letzterem entfällt zudem die Wartung und Konfiguration von Servern etc. für einen reibungslosen und produktiven Betrieb.

Aus Entwicklerperspektive ist besonders der technische Aspekt des Serverless Computing interessant, da keine eigenen Server für den Betrieb einer Anwendung notwendig sind. Das Serverless Application Model ist hierfür sehr zu empfehlen, da es die notwendige Infrastruktur in einem Dokument zentralisiert und diese mithilfe weniger Terminalbefehle automatisch bereitstellt.

7 Abkürzungsverzeichnis

DIN		Deutsches Institut für Normung
UML		Unified Modeling Language
PO		Product Owner
IaaS		Infrastructure as a Service
PaaS		Platform as a Service
SaaS		Software as a Service
FaaS		Function as a Service
AWS		Amazon Web Services
GCP		Google Cloud Platform
HTTP		Hypertext Transfer Protocol
IAM		Identity and Access Management
CLI		Command Line Interface
API		Application Programming Interface
JSON		JavaScript Object Notation
YAML		YAML Ain’t Markup Language
ARN		Amazon Resource Name
REST		Representational State Transfer
SAM		Serverless Application Model
z.B.		zum Beispiel
bzw.		Beziehungsweise
npm		Node Packet Manager
IT		Informationstechnik
ggf.		gegebenenfalls
IoT		Internet of Things
u.a.		unter anderem
etc.		et cetera
PC		Personal Computer
UUID		Universally Unique Identifier
Nr.		Nummer

8 Literaturverzeichnis

Abolhassan, F. (2016). *Was treibt die Digitalisierung? - Warum an der Cloud kein Weg vorbeiführt.* Wiesbaden: Springer Gabler.

Añel , J., Montes, D., & Iglesi, J. (2020). *Cloud and Serverless Computing for Scientists.* Cham: Springer.

AWS. (o.D. o.D. o.D.). *aws.amazon.com/de/free*. Abgerufen am 13. 02. 2022 von aws.amazon.com/de/free: https://aws.amazon.com/de/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/amazondynamodb*. Abgerufen am 12. 01. 2022 von docs.aws.amazon.com/amazondynamodb: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/AmazonS3*. Abgerufen am 13. 01. 2022 von docs.aws.amazon.com/AmazonS3: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/apigateway*. Abgerufen am 12. 01. 2022 von docs.aws.amazon.com/apigateway: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/AWSCloudFormation*. Abgerufen am 13. 01. 2022 von docs.aws.amazon.com/AWSCloudFormation: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/cli*. Abgerufen am 13. 01. 2022 von docs.aws.amazon.com/cli: https://docs.aws.amazon.com/cli/latest/reference/cloudformation/

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/cognito*. Abgerufen am 12. 01. 2022 von docs.aws.amazon.com/cognito: https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/de_de/cli*. Abgerufen am 13. 01. 2022 von docs.aws.amazon.com/de_de/cli: https://docs.aws.amazon.com/de_de/cli/latest/userguide/cli-chap-welcome.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/IAM*. Abgerufen am 22. 02. 2022 von docs.aws.amazon.com/IAM: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/lambda*. Abgerufen am 12. 01. 2022 von docs.aws.amazon.com/lambda: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

AWS. (o.D. o.D. o.D.). *docs.aws.amazon.com/serverless-application-model*. Abgerufen am 13. 01. 2022 von docs.aws.amazon.com/serverless-application-model: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html

Barmscheidt, S., Ebert, M., Hoppe, P., Kelmes, S., & Röthemeier, J. (o.D. 10. 2018). *alexander-otto-sportstiftung.de.* Abgerufen am 26. 01. 2022 von alexander-otto-sportstiftung.de: http://www.alexander-otto-sportstiftung.de/handbuecher-der-alexander-otto-sportstiftung/

Becker, W., Ulrich, P., Schmid, O., & Feichtinger, C. (2020). *Industrielle Digitalisierung - Entwicklungen und Strategien für mittelständische Unternehmen.* Wiesbaden: Springer Gabler.

Brandt-Pook, H., & Kollmeier, R. (2020). *Softwareentwicklung kompakt und verständlich - Wie Softwaresysteme entstehen.* Wiesbaden: Springer Vieweg.

Deckert, R. (2019). *Digitalisierung und Industrie 4.0 - Technologischer Wandel und individuelle Weiterentwicklung.* Wiesbaden: Springer Gabler.

dfn. (o.D. o.D. o.D.). *dfn.de*. Abgerufen am 07. 02. 2022 von dfn.de: https://www.dfn.de/dienstleistungen/dfnterminplaner/

doodle. (o.D. o.D. o.D.). *doodle.com/de/*. Abgerufen am 07. 02. 2022 von doodle.com/de/: https://doodle.com/de/

easyverein. (o.D. o.D. o.D.). *easyverein.com*. Abgerufen am 07. 02. 2022 von easyverein.com/: https://easyverein.com/

EHI Retail Institute. (15. 03. 2021). *de-statista-com*. Abgerufen am 14. 02. 2022 von de-statista-com: https://de-statista-com.ezproxy.hs-augsburg.de/statistik/daten/studie/1221723/umfrage/umfrage-zur-bedeutung-cloud-basierter-anwendungen-im-handel/

Gabbrielli, M., Giallorenzo, S., Lanese, I., Montesi, F., Peressotti, M., & Zingaro, S. (2019). No More, No Less - A Formal Model for Serverless Computing. In H. Nielson, & E. Tuosto, *Coordination Models and Languages* (S. 148 - 157). Cham: Springer.

GitLab. (03. 05. 2021). *Breakdown of software development methodologies practiced worldwide in 2021 [Graph].* Abgerufen am 03. 01. 2022 von https://www-statista-

com.ezproxy.hs-augsburg.de/statistics/1233917/software-development-methodologies-practiced/: https://www-statista-com.ezproxy.hs-augsburg.de/statistics/1233917/software-development-methodologies-practiced/

Gloger, B., & Häusling, A. (2011). *Erfolgreich mit Scrum – Einflussfaktor Personalmanagement.* München: Carl Hanser Verlag GmbH & Co. KG.

Golinsky, F. (2020). *Moderne Vereinsorganisation - Vereinsmanagement leicht gemacht.* Berlin: Springer Gabler.

Hanschke, I. (2018). *Digitalisierung und Industrie 4.0 - einfach und effektiv.* München: Carl Hanser Verlag GmbH & Co. KG.

Hindel, B., Hörmann, K., Müller, M., & Schmied, J. (2009). *Basiswissen Software-Projektmanagement.* Heidelberg: dpunkt.verlag.

JIMDO. (o.D. o.D. o.D.). *jimdo.com/de*. Abgerufen am 07. 02. 2022 von jimdo.com/de: https://www.jimdo.com/de/?_ncr=true&referrer=https://duckduckgo.com/

json.org. (o.D. o.D. o.D.). *json.org*. Abgerufen am 16. 01. 2022 von json.org: https://www.json.org/json-de.html

Kuster, J., Bachmann, C., Huber, E., Hubmann, M., Lippmann, R., Schneider, E., . . . Wüst, R. (2019). *Handbuch Projektmanagement - Agil – Klassisch – Hybrid.* Berlin, Heidelberg: Springer Gabler.

McHaney, R. (2021). *Cloud Technologies - An Overview of Cloud Computing Technologies for Managers.* Hoboken: John Wiley & Sons, Ltd.

Mell, P., & Grance, T. (07. 01. 2022). The NIST Definition of Cloud Computing. Gaithersburg, MD, USA.

Mertens, P., Barbian, D., & Baier, S. (2017). *Digitalisierung und Industrie 4.0 – eine Relativierung.* Wiesbaden: Springer Vieweg.

Metzner, A. (2020). *Software - Engineering - kompakt.* München: Carl Hanser Verlag.

Meyer, A. (2018). *Softwareentwicklung - Ein Kompass für die Praxis.* Berlin, Boston: De Gruyter Oldenbourg.

Microsoft. (o.D. o.D. o.D.). *Azure Microsoft*. Abgerufen am 07. 01. 2022 von Azure Microsoft: https://azure.microsoft.com/de-de/overview/what-is-cloud-computing/?cdn=disable#cloud-computing-models

Moore, G. (19. 04. 1965). *newsroom.intel.com.* Abgerufen am 26. 01. 2022 von newsroom.intel.com: https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/moores-law-electronics.pdf

online-vereinsverwaltung. (o.D. o.D. o.D.). *online-vereinsverwaltung.net*. Abgerufen am 10. 02. 2022 von online-vereinsverwaltung.net: https://online-vereinsverwaltung.net/

Peters, T., & Schelter, N. (2021). *Kompakte Einführung in das Projektmanagement - Mit vielen praxisnahen Beispielen und modernen didaktischen Instrumenten.* Wiesbaden: Springer Gabler.

RedHat. (11. 12. 2019). *redhat.com*. Abgerufen am 09. 02. 2022 von redhat.com: https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services

scrum.org. (o.D. o.D. o.D.). Abgerufen am 07. 01. 2022 von scrum.org: https://www.scrum.org/resources/what-is-scrum

Sehgal, N., & Bhatt, P. (2018). *Cloud Computing - Concepts and Practices.* Cham: Springer.

Stigler, M. (2018). *Beginning Serverless Computing - Developing with Amazon Web Services, Microsoft Azure, and Google Cloud.* Richmond, Virginia: Apress, Berkeley, CA.

Synergy Research Group. (03. 02. 2022). *srgresearch.com*. Abgerufen am 14. 02. 2022 von www.srgresearch.com: https://www.srgresearch.com/articles/as-quarterly-cloud-spending-jumps-to-over-50b-microsoft-looms-larger-in-amazons-rear-mirror

Varshney, R. P., & Sharma, D. K. (2021). Cold Start in Function as a Service: A Systematic Study, Analysis and Evaluation. In P. K. Singh, G. Veselov, V. Vyatkin, A. Pljonkin, J. M. Dodero, & Y. Kumar, *Futuristic Trends in Network and Communication Technologies* (S. 337–349). Singapore: Springer.

vereinsplaner. (o.D. o.D. o.D.). *vereinsplaner.de*. Abgerufen am 07. 02. 2022 von vereinsplaner.de: https://vereinsplaner.de/

Vogel, O., Arnold, I., Chughtai, A., & Kehrer, T. (2011). *Software Architecture - A Comprehensive Framework and Guide for Practitioners.* Heidelberg: Springer.

Witte, F. (2019). *Testmanagement und Softwaretest - Theoretische Grundlagen und praktische Umsetzung.* Wiesbaden: Springer Vieweg.

wix. (o.D. o.D. o.D.). *wix.com*. Abgerufen am 07. 02. 2022 von wix.com: https://www.wix.com/

wordpress. (o.D. o.D. o.D.). *wordpress.org*. Abgerufen am 07. 02. 2022 von wordpress.org: https://de.wordpress.org/

9 Anlagen

Nachfolgend sind die im Text referenzierten Anlagen aufgeführt.

Nr.	Dateiname	Dateipfad
1	external-dependency-service.ts	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/
2	SM_dynamodb.json	aws_backend/aws-iam/policies/
3	SM_AdditionalLambdaDynamodbExecutionRole.json	aws_backend/aws-iam/policies/
4	SM_LambdaCognitoExecutionRole.json	aws_backend/aws-iam/policies/
5	template.yaml	aws_backend/
6	11_build-sam.sh	aws_backend/
7	12_package-sam.sh	aws_backend/
8	13_deploy-sam.sh	aws_backend/
9	event_helper.js	aws_backend/src/layers/aws-integration/
10	participation-repository.js	aws_backend/src/layers/aws-integration/
11	survey-repository.js	aws_backend/src/layers/aws-integration/
12	lambda_create_survey	aws_backend/src/
13	aws-integration	aws_backend/src/layers/
14	index.js	aws_backend/src/lambda_create_survey/
15	module_privileges_helper.js	aws_backend/src/layers/aws-integration/

16	survey-overview.component.ts	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-overview/
17	survey-overview.component.html	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-overview/
18	survey-details.component.ts	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-details/
19	survey-details.component.html	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-details/
20	survey-participate.component.ts	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-participate/
21	survey-participate.component.html	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-participate/
22	survey-participations.component.ts	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-participations/
23	survey-participations.component.html	clubmanager_lib-survey/projects/clubmanager/lib-survey/src/lib/survey-participations/
24	main.ts	application-base/src/

Hochschule
Augsburg University of
Applied Sciences

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

Augsburg, den 24.02.2022 Simon Mayer

Ort, Datum Unterschrift des/der Studierenden