Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Rebecca **Kietzer**

Technologische Antworten auf soziale Herausforderungen im Homeoffice: Entwurf und Implementierung einer Slack-App zur Verbesserung der sozialen Standortpflege in einem IT-Beratungsunternehmen

Technological Responses to Social Challenges in the Home Office: Design and Implementation of a Slack App to enhance the Social Location Management of an IT Consulting Company

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Rebecca **Kietzer**

Technologische Antworten auf soziale Herausforderungen im Homeoffice: Entwurf und Implementierung einer Slack-App zur Verbesserung der sozialen Standortpflege in einem IT-Beratungsunternehmen

Technological Responses to Social Challenges in the Home Office: Design and Implementation of a Slack App to enhance the Social Location Management of an IT Consulting Company

Bearbeitungszeitraum: von 1. März 2024
bis 15. Juli 2024

1. Prüfer: Prof. Dr.-Ing. Christoph P. Neumann

2. Prüfer: Prof. Dipl.-Des. Martin Frey

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Selbstständigkeitserklärung

Name und Vorname
der Studentin/des Studenten: **Kietzer, Rebecca**

Studiengang: **Medieninformatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Technologische Antworten auf soziale Herausforderungen im Homeoffice: Entwurf und Implementierung einer Slack-App zur Verbesserung der sozialen Standortpflege in einem IT-Beratungsunternehmen

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 30. Juni 2024

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin (Name, Vorname):	**Kietzer, Rebecca**
Studiengang:	Medieninformatik
Aufgabensteller, Professor:	Prof. Dr.-Ing. Christoph P. Neumann
Durchgeführt in (Firma):	codecentric AG
Betreuer in Firma:	David Schwarzmann
Ausgabedatum:	1. März 2024
Abgabedatum:	15. Juli 2024

Titel:

Technologische Antworten auf soziale Herausforderungen im Homeoffice: Entwurf und Implementierung einer Slack-App zur Verbesserung der sozialen Standortpflege in einem IT-Beratungsunternehmen

---

Zusammenfassung:

Die Kommunikationsplattform Slack stellt ihren Nutzern bereits seit einiger Zeit Apps und Integrationen zur Verfügung, die den Zugriff auf eine Anwendung direkt aus der Plattform heraus ermöglichen. Oft geht eine solche App mit der Funktionalität eines Chatbots einher, doch in vielen Fällen bietet eine Slack-App mehr als nur das und besitzt beispielsweise auch einen Home-Tab, der einen One-to-One-Raum zwischen User und Anwendung darstellt. Das Ziel dieser Arbeit ist es, eine Slack-App zu entwerfen und anschließend zu implementieren, deren Benutzer Mitarbeiter eines IT-Beratungsunternehmens sind. Diese sollen durch die Nutzung der App - trotz des Verteilten Arbeitens - zum gemeinsamen Austausch angeregt werden, sowie Möglichkeiten haben, ihre Gesundheit zu fördern und Büro- bzw. Homeoffice-Tage besser organisieren zu können. Dies soll zum Erhalt der Firmenkultur im Homeoffice-Zeitalter beitragen.

Zur Umsetzung der App, die den Namen *beecaring* trägt, wurde ein Serverless Cloud Deployment gewählt, welches verschiedene Cloud-Komponenten von AWS benutzt. Des weiteren wurde das Slack-eigene Framework Bolt für JavaScript verwendet, welches auf Node.js basiert und somit als Node-Modul eingebunden ist. Im Vordergrund stehen das Entwerfen und Implementieren des Home-Tabs der App, der Umgang mit Userdaten im Bezug auf Speicherung und Abfragen, das Senden sowohl interaktiver als auch zeitgesteuerter Nachrichten, sowie das Abfangen und Behandeln von Events, bzw. Interaktionen des Users mit der beecaring Slack-App.

Diese Arbeit ist nicht nur auf die Beschreibung des spezifischen Projekts ausgelegt, sondern soll auch anderen Entwicklern als Quelle und Leitfaden zum Planen und Implementieren einer Slack-App dienen, da die Entwicklung von Chatbots wie Slack-Apps ein hohes Potential für die Zukunft birgt, sich jedoch von der Entwicklung herkömmlicher Web-Apps unterscheidet. Diese Unterschiede, und auch eventuelle Herausforderungen, die dadurch entstehen, sollen am Ende deutlich werden.

Schlüsselwörter: Slack-App, Bolt Framework, Block Kit, AWS Serverless, AWS-Lambda-App, JavaScript, Node.js, Bot

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
AWS	Amazon Web Services
BSON	Binary JSON
CLI	Command Line Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DB	Database
ECR	Elastic Container Registry
ECS	Elastic Container Service
EKS	Elastic Kubernetes Service
FaaS	Function-as-a-Service
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
IAM	Identity and Access Management
iOS	Apple iPhone Operating System
JS	JavaScript
JSON	JavaScript Object Notation
MJS	Module JavaScript
MQL	MongoDB Query Language
NPM	Node Package Manager
NoSQL	Not only SQL
OAuth	Open Authorization

RDS	Relational Database Service
REST	Representational State Transfer
S3	Simple Storage Service
SHA-256	Secure Hash Algorithm
SSD	Solid State Drive
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
US	User Story
UX	User Experience
vCPU	Virtual Central Processing Unit
VM	Virtuelle Maschine

Kapitel 1

Einleitung

1.1 Problemstellung und Motivation

Spätestens seit dem Ausbruch der Covid-19-Pandemie bieten viele Unternehmen die Möglichkeit, vom Homeoffice aus zu arbeiten, wodurch sich weltweit das Verteilte Arbeiten etabliert hat. Laut einer Umfrage der Avantgarde Experts behielten nach dem Wegfall der gesetzlichen Homeoffice-Pflicht in Deutschland, die während des Corona-Lockdowns galt, 71% der befragten Arbeitgeber eine Homeoffice-Regelung irgendeiner Form bei, das heißt nur 29% gaben an, dass eine Anwesenheitspflicht im Büro bestehe [9]. Im Vergleich zu anderen Branchen ist die Homeoffice-Rate dabei vor allem in der IT-Brache sehr hoch [19]. Das Arbeiten von zuhause aus hat sich natürlich auch nicht ohne Grund etabliert: Es bietet den Arbeitnehmern mehr Flexibilität, Work-Life-Balance und die Möglichkeit, ihren Arbeitsplatz individuell anzupassen, was in der modernen Welt zunehmend als essentiell für Produktivität und Zufriedenheit angesehen wird [10], und oftmals mit der Bedeutung der Firmenkultur einhergeht. Allerdings sind mit dem Homeoffice nicht nur Vorteile verbunden. Zum Beispiel stellen fehlender Kontakt zu Kollegen, Bewegungsmangel sowie unklare Arbeits- und Erreichbarkeitszeiten eine Belastung für viele Personen dar, die von zuhause aus arbeiten [16]. Dies bestätigte sich auch bei einer Umfrage, die bei Mitarbeitern der Firma codecentric am Standort Nürnberg durchgeführt wurde.

1.1.1 codecentric AG

Die codecentric AG ist ein deutsches IT-Beratungsunternehmen mit Schwerpunkt auf individuelle Softwareentwicklung und IT-Dienstleistungen. Das Unternehmen mit Hauptsitz in Solingen wurde 2004 gegründet und hat seitdem mehr als 550 Mitarbeiter und Mitarbeiterinnen gewonnen und über 1.000 erfolgreiche Projekte, verteilt auf bislang 13 Standorte in Deutschland, durchgeführt. Die codecentric AG hat sich auf die Zusammenarbeit mit Kunden unterschiedlicher Branchen spezialisiert, um ihnen bei der Optimierung ihrer IT-Strukturen und der Entwicklung hochwertiger Softwarelösungen zu helfen. Zu den Kunden gehören dabei sowohl mittelständische Unternehmen, als auch namhafte Kunden wie Audi, die Deutsche Telekom, Xing oder

HolidayCheck. [13]

Mit einem eigenen Podcast, dem firmeneigenen Magazin „Der Softwerker“ und dem regelmäßigen Organisieren von Events wie Meetups oder Konferenzen setzt sich die Firma codecentric aktiv für die Weiterbildung interner und externer Arbeitnehmer ein. Die Weiterbildung zählt nämlich zur Firmenkultur, welche bei codecentric eine sehr bedeutende Rolle spielt. Deswegen achtet das Unternehmen auch verstärkt darauf, dass seine Mitarbeiter und Mitarbeiterinnen nicht unter den Folgen des Verteilten Arbeitens leiden.

1.1.2 Schwachstellen bisheriger Homeoffice-Lösungen

Natürlich hat man sich bei codecentric bereits Gedanken über die negativen Auswirkungen des Homeoffice gemacht und einige Ansätze entwickelt, um diesen entgegenzuwirken. Eine große Rolle spielt dabei die Kommunikationsplattform *Slack*, welche als interner Kommunikationskanal genutzt wird. Durch die standortübergreifende Nutzung von Slack und regelmäßig stattfindende Meetings wie beispielsweise dem sogenannten „Lean Coffee“ wird versucht, den Austausch zwischen Kollegen aufrecht zu erhalten. Doch das alleine reicht noch lange nicht aus. Außerdem gibt es neben dem mangelnden Kontakt zu Kollegen noch andere Aspekte, die im Homeoffice problematisch sind.

Ein Aspekt ist beispielsweise fehlende Organisation. Gemeint ist damit die Unklarheit, wer wann von wo aus arbeitet. Wenn die Arbeitszeiten sowie der Arbeitsort komplett flexibel sind, und keine Absprache darüber existiert, kann es oftmals zu Schwierigkeiten bei Planungen oder der Erreichbarkeit kommen.

Ein weiterer Aspekt ist der Bewegungsmangel: Während man zum Büro oft schon früh morgens einen kleinen Spaziergang macht oder gar mit dem Fahrrad zur Arbeit fährt, sind es im Homeoffice nur ein paar Schritte bis zum Arbeitsplatz. Da die Gesundheit der Mitarbeiter und Mitarbeiterinnen der codecentric AG sehr am Herzen liegt, bietet sie einmal wöchentlich Homeoffice-Sport an. Jedoch gibt es auch hier noch Verbesserungspotential, da der Termin nur einmal in der Woche für eine halbe Stunde und zu einer festen Uhrzeit stattfindet. Da um diese Uhrzeit nicht jeder am Homeoffice-Sport teilnehmen kann oder möchte, wäre es besser, den Zeitpunkt individuell für sich festlegen zu können, und das - wenn zeitlich möglich - auch mehrmals pro Woche oder gar mehrmals täglich. Natürlich kann dann aber nicht immer ein ausgebildeter Trainer dabei sein. Ein Problem ist also noch: Wie können die Qualität der Übungen trotzdem möglichst hoch bleiben und die Übungen individuell an die Mitarbeiter angepasst sein?

Wie bereits erwähnt, gibt es einige bestehende Ansätze, um dem Verlust der Kontakte zu Kollegen und Kolleginnen vorzubeugen, und Tools wie Slack gestalten es einfach, miteinander zu kommunizieren. Doch während man sich im Büro beispielsweise an der Kaffeemaschine trifft und daraufhin einen Austausch beginnt, fehlt im Homeoffice oft der Anstoß zum Gespräch mit Kollegen.

Es lässt sich bis heute im Homeoffice also doch noch nicht alles digital ersetzen, was

im Büro üblich bzw. möglich ist - Vielleicht gibt es aber Wege, sich dem zumindest zu nähern.

1.2 Zielsetzung

Das Ziel dieser Bachelorarbeit ist es deswegen, eine Slack-App zu entwerfen und zu implementieren, die folgende Features beinhaltet:

1. Eine Möglichkeit zum Eintragen der Büro-/Homeoffice-Tage mit Erinnerungen zum Eintragen und eine Übersicht der Einträge für das gesamte Team
2. Die Möglichkeit, flexibel Homeoffice-Sport zu machen
3. Anreize zum Austausch mit Kollegen

Die Frage ist dabei, wie die genannten Funktionalitäten am besten - das heißt am benutzerfreundlichsten - mithilfe der Slack-API umgesetzt werden können. Die folgenden Gedanken sind vorerst ein grober Anhaltspunkt und können sich im Laufe der weiteren Analyse und des Entwurfs noch ändern.

1.2.1 Eintragen und Anzeigen der Büro-/Homeoffice-Tage

Hierfür kann ein Bot erstellt werden, der die Nutzer wöchentlich darum bittet, einzutragen, an welchen Tagen sie planen, im Büro zu sein, und an welchen sie remote arbeiten möchten. Dies könnte zum Beispiel über eine einfache Slack-Message mit mehreren Auswahlmöglichkeiten (Montag bis Freitag) realisiert werden.

Natürlich sollte die Auswahl auch jederzeit abrufbar sein und abgeändert werden können. Dies kann beispielsweise über den Home-Screen der Slack-App geschehen, auf dem eine Art Dashboard implementiert werden könnte, sodass die Einträge aller Mitarbeiter eines Teams sichtbar sind und die eigenen Einträge bearbeitet werden können.

1.2.2 Individueller Homeoffice-Sport

Auch für die flexible Gestaltung des Homeoffice-Sports ist ein Bot eine Option. Jedoch sollte eingestellt werden können, wie häufig der Bot aktiv werden soll, also wahlweise auch gar nicht, falls kein Interesse an Sportübungen besteht. Besteht Interesse, so wäre eine Möglichkeit, dass über den Home-Tab der App verschiedene Auswahlmöglichkeiten, welche Bedürfnisse der Arbeitnehmer gerade hat, zur Verfügung stehen, und dort auch ausgewählt werden kann, wie oft der Wunsch nach kurzen Übungen besteht. Je nach Auswahl (beispielsweise „Rückenübungen“) würde der Bot dann kleinere Übungen in Form von Videos vorschlagen.

1.2.3 Anreize zum Austausch mit Kollegen

Wie bringt man Kolleginnen und Kollegen im Homeoffice über Slack miteinander ins Gespräch? Darüber hat sich bereits das Unternehmen Donut Technologies Inc. Gedanken gemacht und die Slack-Integration *Donut* entwickelt. Donut bietet seinen Usern unter anderem den sogenannten „Watercooler“ – einen virtuellen Wasserspender, bei dem sich das Team versammeln kann um soziale Barrieren zu überwinden, Kreativität zu wecken und die Unternehmenskultur durch authentische Gespräche zu stärken [17]. Und obwohl Donut bereits von einigen Mitarbeitern genutzt wird, gab es bei einer Umfrage zum Thema der sozialen Standortpflege – durchgeführt bei der codecentric AG in Nürnberg – noch Bedarf, Kollegen im Homeoffice miteinander ins Gespräch zu bekommen. Interessant wäre also vorerst eine Analyse, welche Features Donut bereits anbietet und inwiefern sie noch ausbau- und verbesserungsfähig sind. Ein weiteres Ergebnis der genannten Umfrage war außerdem, dass Events wie Spieleabende vermutlich häufiger stattfinden würden, wenn man besser planen könnte, wie viele Standort-Mitarbeiter an welchen Tagen im Büro sind. Hinsichtlich dieses Aspekts wäre das bereits genannte Feature zum Eintragen der Bürotage also ebenfalls sehr nützlich.

Kapitel 2

Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe erläutert, die für das Projekt von Bedeutung sind. Zunächst wird auf die Plattform Slack eingegangen und die generelle Rolle von Bots erklärt. Anschließend wird spezifisch auf Slack-Apps eingegangen - sowohl auf ihre verschiedenen Erscheinungsformen als auch auf die wesentlichen Entwicklungswerkzeuge wie das Framework Bolt und das UI-Framework Block Kit. Zudem wird erklärt, welche APIs zur Eventverarbeitung benötigt werden und welche Rolle Scopes und Permissions bei der Slack-App-Entwicklung spielen. Am Ende wird kurz auf die Slack-App *Donut* eingegangen, die in anderen Abschnitten gelegentlich als Vergleich herangezogen wird.

2.1 Slack

Stewart Butterfield, Eric Costello, Cal Henderson und Serguei Mourachov entwickelten Slack zunächst als internes Kollaborationstool, während sie an einem anderen Projekt arbeiteten, erkannten aber bald das Potenzial des Tools für den breiteren Unternehmensmarkt. Und tatsächlich wurde Slack nach dem Launch der Beta Version Mitte 2013 in kürzester Zeit eines der beliebtesten Kommunikationstools für Teams. [39, S. 11]

Ende 2023 zählt Slack über 750.000 Organisationen und 32,3 Millionen User, die die Plattform täglich nutzen. 79% der Organisationen berichteten dabei, dass die Nutzung von Slack die Team-Kultur positiv beeinflusst hat, und 91% der Nutzer berichteten, dass sich ihre Arbeitsweise im Remote-Umfeld nach der Umstellung auf Slack verbessert hat. [38]

Zu den Kunden von Slack zählen unter anderem Großunternehmen wie Pinterest, Airbnb, CNN und Target [39, S. 12]. Auch die codecentric AG ist eines der Unternehmen, zu dessen Arbeitsalltag Slack bereits fest dazugehört - Hier läuft die gesamte interne Kommunikation über Slack. Ein Grund dafür ist, dass Slack durch organisierte Bereiche bzw. Channels eine einfache Vernetzung zwischen Teams sowie über verschiedene Standorte hinweg ermöglicht. Die Kernidee von Slack ist außerdem, alle Tools an

einem zentralen Ort zu bündeln und damit den Workflow zu vereinfachen - Dies wird dadurch möglich, dass in Slack verschiedene Tools und Dienste integriert sowie für Slack entwickelte Apps verwendet werden können [48].

2.2 Bots

Einfach ausgedrückt ist ein Bot oder Chatbot lediglich ein User Interface, das es ermöglicht, in einer Anwendung (wie beispielsweise Slack oder Facebook) mit verschiedenen Diensten zu interagieren. In den meisten Fällen sind Bots digitale User innerhalb der Anwendung, das heißt sie sind nicht von Menschen betrieben, sondern von Software gesteuert. Über die Konversation mit dem User bringen Bots einen bestimmten Dienst in die Anwendung ein (vgl. Abbildung 2.1). Ein häufiger Irrtum ist, dass der Bot selbst der Dienst ist - In Wirklichkeit ist der Bot aber nur eine Schnittstelle zum Dienst, so wie eine Website eine Schnittstelle für die Buchung eines Fluges sein kann. [37, S. 2-3]

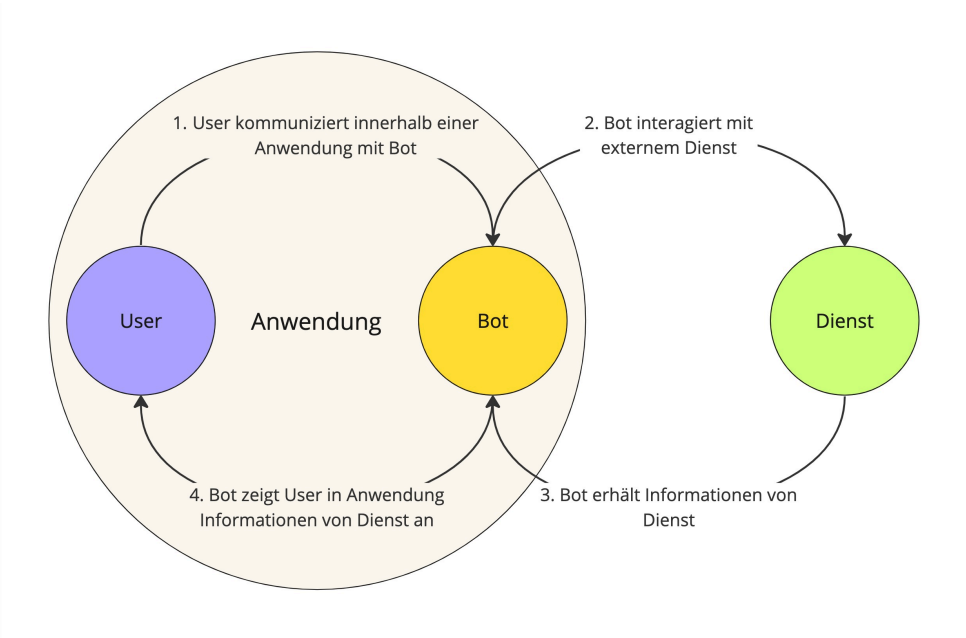

Abbildung 2.1: Interaktion mit einem Bot (Quelle: Eigene Darstellung)

2.3 Slack-Apps

Eine Slack-App kann innerhalb von Slack installiert werden und verschiedene Erscheinungsformen haben: Zu den meisten Apps gibt es einen Home-Tab, das heißt eine Seite, auf der direkt mit der App interagiert werden kann oder Informationen über die App entnommen werden können (Abbildung 2.2). Die Home-Seite der App ist ein privater One-to-One Raum zwischen User und App. Die App kann aber auch in Form einer Nachricht - entweder in einem Channel oder als Direktnachricht - (Abbildung 2.3) sowie als Modal, das heißt über ein Pop-Up Interface, in dem üblicherweise Daten des Users gesammelt werden (Abbildung 2.4), mit dem User interagieren. [44]

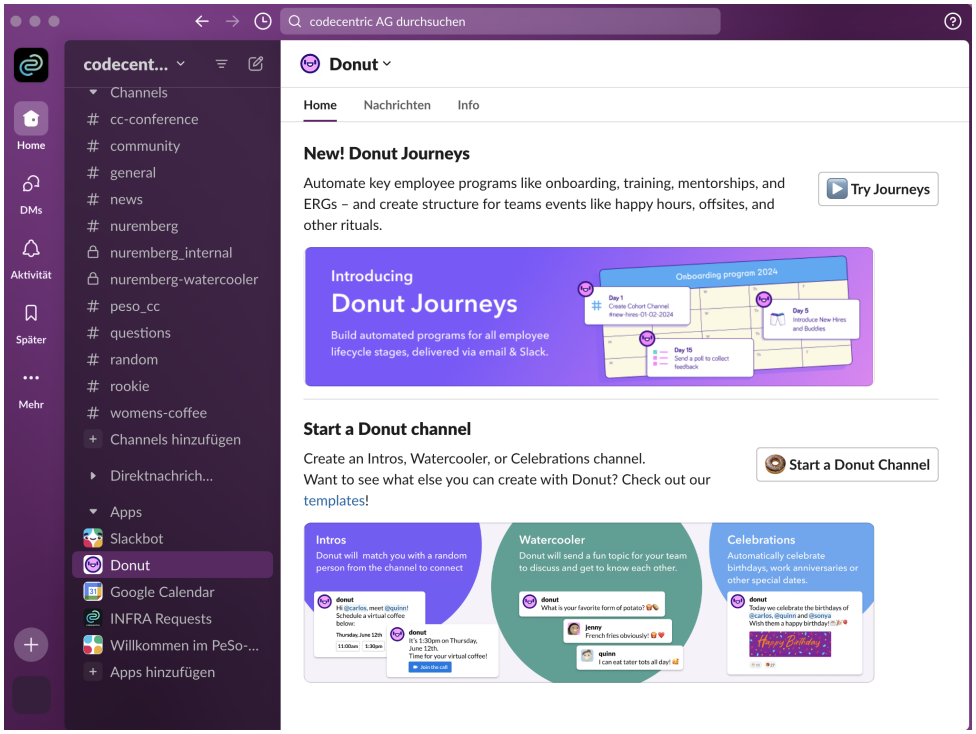

Abbildung 2.2: Home-Tab einer Slack-App am Beispiel der App *Donut* (Quelle: Eigener Screenshot)

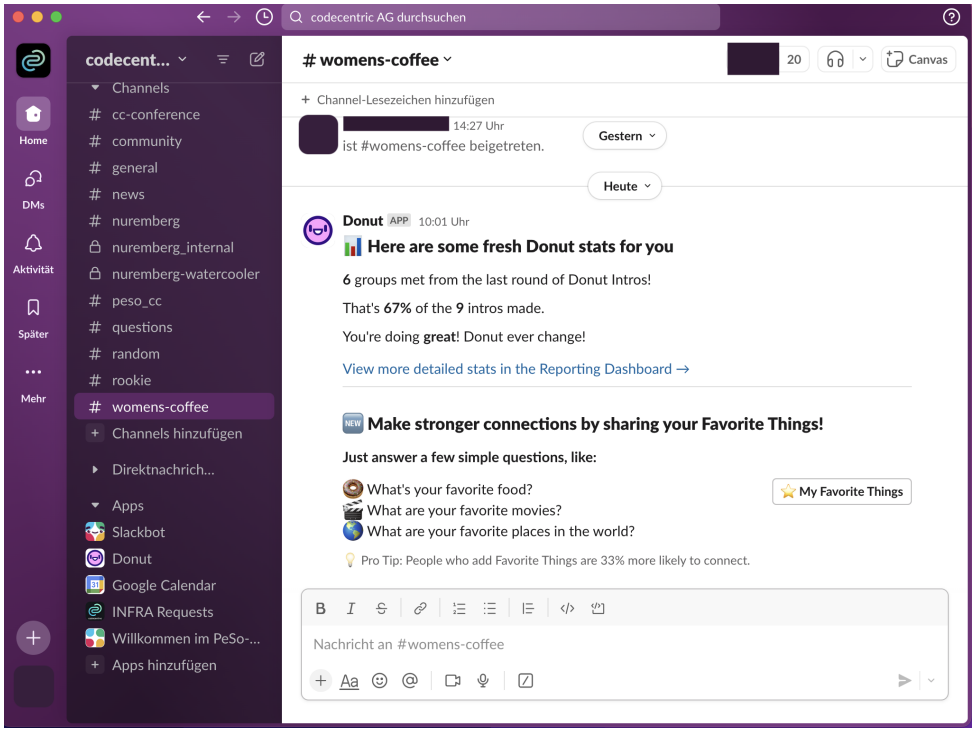

Abbildung 2.3: Nachricht einer Slack-App in einem Channel am Beispiel der App *Donut* (Quelle: Eigener Screenshot)

Es gibt dabei verschiedene Arten von Slack-Apps, wie in Tabelle 2.1 dargestellt.

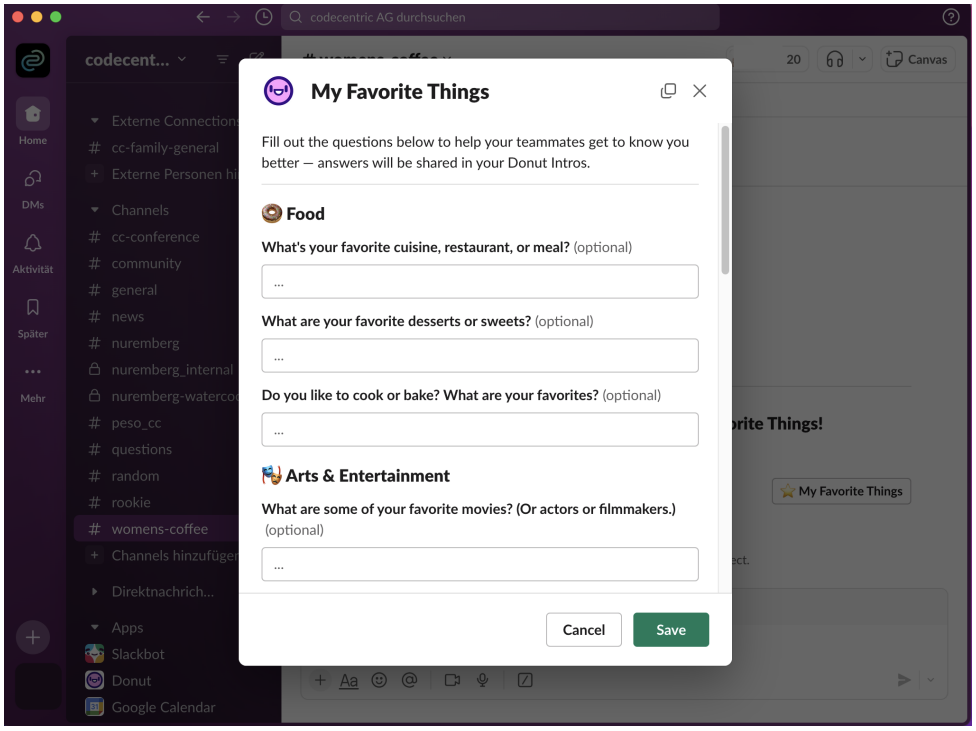

Abbildung 2.4: Modal einer Slack-App am Beispiel der App *Donut* (Quelle: Eigener Screenshot)

Workflow Apps	Erstellung automatisierter Tasks und Prozesse mithilfe des *Slack Workflow Builders*
Directory Apps	Öffentlich bereitgestellte Apps oder Integrationen anderer Dienste
Custom Apps	Individuelle Apps für einen einzelnen Arbeitsbereich oder eine Organisation

Tabelle 2.1: Slack-App-Typen [48]

2.4 Slack-API

Slack-Apps können also von Entwicklern auch selbst - benutzerdefiniert - erstellt werden. Dafür stehen Entwicklern eine Reihe an APIs zur Verfügung, über die auf verschiedene Funktionen von Slack zugegriffen werden kann, beispielsweise Nachrichten zu senden, Benachrichtigungen zu verwalten und vieles mehr. Dadurch können benutzerdefinierte Prozesse automatisiert und Slack an die spezifischen Anforderungen eines Teams oder einer Organisation angepasst werden. [44]

2.4.1 Bolt

Bolt ist ein Framework, das die schnelle Entwicklung von Slack-Apps ermöglicht, indem das Framework verschiedene Plattform-APIs und Features hinter einer einfachen Schnittstelle zusammenfasst. Beispielsweise umfasst Bolt Funktionen wie Token-

Validierung, Serverunterstützung, eine vereinfachte OAuth-Schnittstelle sowie eine automatische Wiederholungslogik. Das Framework ist verfügbar für die Programmiersprachen JavaScript, Python und Java. [25, S. 225]

2.4.2 Block Kit

Mit Bolt kann das sogenannte *Block Kit*, das UI-Framework für Slack, genutzt werden. Ein UI-Framework, auch bekannt als UI-Toolkit, ist eine Sammlung von vordefinierten Tools, Bibliotheken, Komponenten und Stilen, die zur Entwicklung von Benutzeroberflächen (UIs) für Softwareanwendungen verwendet werden. Diese Frameworks bieten Entwicklern eine Struktur und eine Reihe von vorgefertigten Elementen, die sie verwenden können, um effizient und konsistent Benutzeroberflächen zu erstellen. Bei Slack umfasst dies Dinge wie Schaltflächen und Symbole, die es Usern ermöglichen, mit einem Produkt zu interagieren. Mit sogenannten *Blocks* können Entwickler die Reihenfolge und das Erscheinungsbild dieser Informationen anpassen. Blocks sind wiederverwendbare Komponenten, die in einem Slack-Workspace beliebig gestapelt oder neu angeordnet werden können, um App-Layouts zu erstellen. Das Block Kit ist eine Sammlung von Layouts für Modals, Nachrichten oder Tabs und es ermöglicht eine strukturierte JSON-Darstellung, um die Funktionen einer Anwendung auszudrücken. Diese JSON-Darstellung, oft als *Payload* bezeichnet, ist von wesentlicher Bedeutung für die Speicherung und Übertragung von Daten. Innerhalb eines Blocks können wiederum sogenannte *Block Elements* genutzt werden. Diese machen das UI erst interaktiv - Block Elements können zum Beispiel Buttons, Checkbox Gruppen, Datepicker oder Bilder sein. [25, S. 189-198]

2.4.3 APIs und Events

Um die Block Elements interaktiv zu machen, muss diesen das passende Event zugeordnet werden. Es gibt eine ganze Reihe an möglichen Events: Von einem üblichen On-Click-Event über das Event, dass in einem bestimmten Channel eine bestimmte Nachricht gesendet oder empfangen wurde, oder dass jemand einem Channel beigetreten ist. Auf diese Events kann über die Events API von Slack zugegriffen werden. So können ausgewählte Events (zum Beispiel `team_join`, `link_shared`, `app_home_opened`) abonniert und anschließend festgelegt werden, wie auf diese Events reagiert werden soll. Hierfür wird eine weitere API von Slack benötigt: Die Web API, welche über 100 HTTP-Methoden bietet, die den Großteil der Slack-App-Funktionen unterstützen. Hierüber hat der Entwickler die Möglichkeit, über seine App umfangreiche, interaktive Informationen in Slack zu senden oder zu öffnen. Beispiele für solche Methoden sind: `chat.postMessage` (Senden einer Chat-Nachricht), `chat.unfurl` (Öffnen eines Links) oder `views.open` (Öffnen eines Modals). [47]

In Abbildung 2.5 ist beispielhaft dargestellt, wie auf ein Event in Slack reagiert wird - Hier kommt sowohl die Events API, als auch die Web API zum Einsatz.

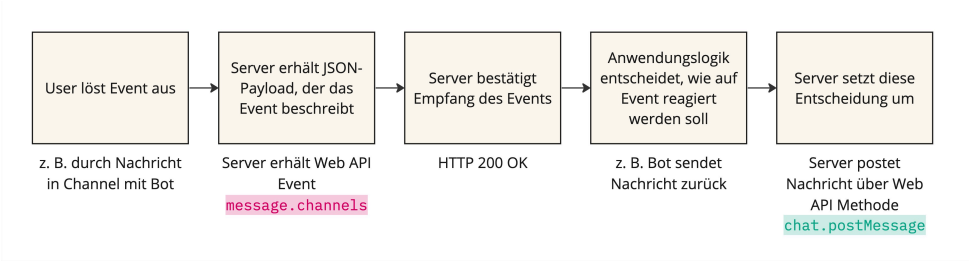

Abbildung 2.5: Reaktion auf ein Slack-Event (Beispiel) (Quelle: Eigene Darstellung)

2.4.4 Scopes und Permissions

Damit die APIs genutzt werden können, müssen aber erst sogenannte *Scopes* beim User erfragt und vom User angenommen werden. Diese Scopes sind dafür da, die Permissions der App zu managen. Beispielsweise gibt es den Scope `chat:write`, welcher benötigt wird, um Nachrichten in einem Channel zu verschicken. Die Scopes legt der Entwickler bereits bei der Konfiguration der App fest, beim User werden sie bei der Installation der App angefragt. Hat der User seine Erlaubnis erteilt, so wird ein Access Token ausgetauscht. Mit diesem Token können dann die Permissions dahinter für die verschiedenen APIs benutzt werden. Das alles ist Teil des OAuth-Flows einer Slack-App. [41]

2.5 Donut

Donut ist eine beliebte Slack-App, die entwickelt wurde, um das Gemeinschaftsgefühl in entfernten oder verteilten Teams zu stärken, indem zum Beispiel Teammitglieder nach dem Zufallsprinzip einander vorgestellt und für virtuelle Kaffee- oder Mittagstreffen zusammengebracht werden. Das Unternehmen *Donut Technologies Inc.* wurde 2016 von Christian Whitehouse, Dan Manian, Jeff Manian und Sarah Arnold gegründet [14].

Donut hat seitdem über 15 Millionen Verbindungen in mehr als 20.000 Unternehmen (beispielsweise auch Netflix) innerhalb und zwischen Abteilungen, Bürostandorten und Kontinenten hergestellt. Die App gibt es nicht nur für Slack, sondern auch für Microsoft Teams. [26]

Kapitel 3

Design-Prozess

Nun kann mit dem Entwurf der Slack-App begonnen werden. Ein essenzieller Teil des Entwurfs ist ein Design-Prozess, in dem die genauen Anforderungen an die Anwendung sowie entsprechende Lösungsansätze herausgearbeitet werden. Dabei stehen zunächst nicht die technischen Details im Vordergrund, sondern ist das Ziel dieses Prozesses vielmehr die Entwicklung eines Prototyps der Anwendung, um diese visuell greifbar zu machen. In diesem Kapitel werden die einzelnen Schritte des Design-Prozesses durchlaufen und erläutert. Der Autor und Designer Bernd Löbach beschreibt den Design-Prozess nicht nur als kreativen Prozess, sondern auch als einen „Problemlösungsprozess“ [24]:

- Ein Problem ist vorhanden und wird entdeckt
- Informationen über das Problem werden gesammelt, ausgewertet und kreativ (schöpferisch) verknüpft
- Es entstehen Problemlösungen, die nach aufgestellten Kriterien beurteilt werden
- Die geeignetste Lösung wird realisiert (zum Beispiel in ein Produkt überführt)

Laut Semler und Tschierschke ist „der ideale Design-Prozess einer App [...] iterativ. Die App sollte möglichst früh prototypisiert werden. Dabei sollten die Design-Entwürfe wiederholt anhand von prototypischen Realisierungen eines Interfaces evaluiert werden“ [36, S. 427]. Dieser Prozess kann je nach Kontext aus verschieden vielen Schritten bestehen bzw. unterschiedlich ablaufen. In diesem Fall wird sich an den von Semler und Tschierschke [36] sowie den von Jacobsen und Meyer [22] vorgeschlagenen Phasen des App-Designs orientiert.

3.1 Konkurrenz- und Mitbewerberanalyse

Da bereits eine grobe Idee vorliegt, welches Produkt entwickelt werden soll, können Kreativitätstechniken zur Ideenfindung, wie Brainstorming und Mindmapping, übersprungen und stattdessen direkt mit einer Analyse begonnen werden, welche Produkte es bereits am Markt gibt, die darauf abzielen, die in Kapitel 1 genannten Probleme

anzugehen. Konkurrenz ist erst einmal ein gutes Zeichen, da diese bedeutet, dass es einen potenziellen Markt für die Idee gibt [36, S. 89]. Die Slack-App für die breite Masse auf den Markt zu bringen, ist jedoch zunächst gar nicht das Ziel, sondern, eine auf das Team der codecentric AG am Standort Nürnberg zugeschneiderte App zu entwerfen. Daher ist die Frage nicht, wie sich die Anwendung am besten gegen die Konkurrenz behaupten kann, sondern, wieso bisher zur Verfügung stehende Produkte nicht eingesetzt oder genutzt werden und wie man die gefundenen Schwachstellen anderer Produkte in der zu entwickelnden Anwendung schließen könnte. Da die App im Großen und Ganzen drei verschiedene Funktionen haben soll, wird in dieser Analyse für jedes der Features nach Produkten, die dieses enthalten, gesucht. Bei der Recherche wird verstärkt nach Slack-Integrationen Ausschau gehalten, da eine grundlegende Voraussetzung dieser Arbeit ist, dass es sich um eine Slack-App handeln soll.

3.1.1 Produkte zur Organisation der Büro-/Homeoffice-Tage

Bisher hatten einige Mitarbeiter ihren Arbeitsort regelmäßig über *Google Kalender* eingetragen, doch möchte man nicht nur von einer bestimmten Person wissen, wann diese wieder im Büro anwesend ist, sondern eine Übersicht des ganzen Teams, hilft Google Kalender leider nicht weiter. Bei der Suche nach Produkten zur Organisation der hybriden Arbeitsweise stößt man schnell auf viele Desksharing-Tools, die darauf ausgelegt sind, Arbeitsplätze oder Besprechungsräume zu reservieren. Möchte man zusätzlich die Option zum Eintragen von Büro- und Homeoffice-Tagen, findet man Tools wie *Robin* [33] oder *OfficeRnD* [29]. Beide bieten eine Slack-Integration und nützliche Funktionen zur Organisation und Übersicht der verschiedenen Arbeitsorte. Zusätzlich gibt es bei beiden Produkten Analysen zur Büroanwesenheit etc. Office RnD lässt den User sein Dashboard sogar so konfigurieren, wie er es haben möchte. Diese Lösungen klingen zunächst perfekt, doch bucht man eines dieser Produkte, bekommt man viele weitere Features, die im Preis miteinberechnet sind - wie Möglichkeiten zur Arbeitsplatzreservierung - dazu. Da diese Funktionen am Standort Nürnberg der codecentric AG aktuell nicht gebraucht werden, und das zukünftige Produkt so schlank sein soll, dass Mitarbeiter nicht mit zu vielen Funktionen konfrontiert werden, die gar nicht benötigt werden, kommt eine derartige Lösung nicht infrage.

3.1.2 Produkte zur Aufrechterhaltung der körperlichen Gesundheit von Mitarbeitern

Eine App, die in München entwickelt wurde und eine Slack-Integration mit kleinen Aktivitätsübungen für die körperliche und mentale Gesundheit anbietet, ist *MinQi*. MinQi bietet sowohl die Teilnahme an Livesessions mit zertifizierten Trainern, als auch kurze (1-8 min. lange) Übungen, die entweder alleine oder zusammen im Team durchgeführt werden können. Es gibt tägliche Check-ins in Form eines Modals, in dem der User sein aktuelles Wohlbefinden tracken kann, und die Möglichkeit, Erinnerungen an Pausen zu aktivieren und Fokuszeiten zu planen. [53]

MinQi ist allerdings keine kostenfreie Lösung und die Livesessions, die MinQi hos-

tet und die vermutlich einen großen Teil des Preises ausmachen, sind ähnlich zum Homeoffice-Sport, den es bei codecentric bereits gibt. Soll der Fokus auf individuellen Übungsvorschlägen für zwischendurch liegen, ist die Slack-App *Wellness Coach* eine mögliche Alternative.

Der Nutzer kann hier nämlich mögliche Beschwerden auswählen, auf die er sich am jeweiligen Tag konzentrieren möchte, und er kann angeben, wann er an die Durchführung einer von Wellness Coach ausgewählten Übung erinnert werden möchte. Anschließend kann er die Übung, die er in Form eines Videos von der App erhalten hat, starten. Der User hat aber auch die Möglichkeit, in einem Channel eine gemeinsame Übungssession vorzuschlagen. Wellness Coach ist nicht nur auf die körperliche, sondern auch auf die mentale Gesundheit von Mitarbeitern fokussiert. [52]

3.1.3 Produkte zur Förderung des Austauschs mit Kollegen

In vorherigen Kapiteln wurde bereits über Donut [17] gesprochen. Da im Slack-Workspace der codecentric AG bereits Donut-Channels existieren, die auch aktiv - jedoch nicht von jedem - genutzt werden, wurde analysiert, wieso die Teilnehmerzahl nicht größer ist. Eine Umfrage unter den Standort-Mitarbeitern ergab, dass ein Grund oftmals sei, einem zusätzlichen Channel beitreten zu müssen, von dem man häufig Benachrichtigungen bekäme, allerdings nur alle zwei Wochen über ein „Donut Intro“ einem anderen Mitarbeiter vorgestellt würde, bzw. ein zufälliges Treffen ausgelost würde. Bei den Intros von Donut gibt es die Möglichkeit, eine Reihe an Fragen über sich zu beantworten, um einander besser kennenzulernen, doch ein häufiger Kritikpunkt war zudem, dass diese Fragen sich irgendwann nur noch wiederholen würden. Es müsse eine Möglichkeit geben, jedes Mal etwas Neues voneinander zu erfahren.

Ein weiteres Teambuilding-Tool für Slack ist *Ricotta*. Ricotta bietet viele verschiedene Möglichkeiten, den Kontakt zu Kollegen zu stärken. Anders als Donut bietet Ricotta neben „Icebreakers“ sogar Mini-Games und Trivia an [32]. Man sollte allerdings stets auf die Balance zwischen Mitarbeiterkontakt und produktiver Arbeit achten: Während es eben darum ging, den Kontakt zu Mitarbeitern vor allem im Homeoffice zu unterstützen, darf die Arbeit natürlich nicht zu kurz kommen. Es sollte überlegt und individuell entschieden werden, ob eine App wie Ricotta das Unternehmen mehr vorantreibt, oder bei der Arbeit eher im Wege steht.

3.1.4 Fazit

Zusammenfassend lässt sich sagen, dass für alle drei Features eine eigene Implementierung sinnvoll ist. Der ausschlaggebende Punkt ist vor allem auch: Alle drei Funktionen in Kombination sind so noch nicht am Markt vorhanden. Würde man für jedes gewünschte Feature eine eigene App installieren, bestünde schnell die Gefahr der „App Fatigue“, also der Überforderung, die aufgrund der Nutzung zu vieler verschiedener Apps entsteht [18]. Hiergegen eignet sich eine eigene App, die alle drei Aspekte bündelt und genau auf die Bedürfnisse der codecentric AG Nürnberg abgestimmt ist. Einige Inspirationen können dabei aber durchaus aus den bereits bestehenden und

soeben analysierten Produkten gezogen werden.

3.2 Zielgruppe

In dieser Phase geht es darum, die Zielgruppe, also die zukünftigen Nutzer, zu bestimmen und so gut wie möglich kennenzulernen. In diesem Projekt kann die Zielgruppe sehr klein gefasst werden: Sie besteht lediglich aus codecentric-Mitarbeitern des Standorts Nürnberg. Also müssen die Nutzeranforderungen dieser Gruppe näher betrachtet werden, damit genau das Produkt entwickelt werden kann, das sich von den Nutzern gewünscht wird. Aus den Bedürfnissen und Wünschen der Nutzer können dann Personas generiert und die ersten konzeptionellen Überlegungen abgeleitet werden. [22, S. 97]

3.2.1 User Research

Die Recherche über die Zielgruppe wird auch „User Research“ genannt. Sie ist ein sehr wertvoller erster Schritt, auf dessen Ergebnisse im späteren Projektverlauf immer wieder zurückgegriffen werden kann. Um die Nutzeranforderungen zu erheben, gibt es verschiedene Methoden zur Datensammlung. Eine Möglichkeit sind zum Beispiel Befragungen. Befragungen werden mithilfe von Fragebögen durchgeführt, was bedeutet, dass sehr große Stichproben gesammelt werden können, da die Befragung mit einer großen Anzahl an Teilnehmern durchgeführt werden kann. Ein Nachteil dabei ist allerdings, dass die Befragung auf die in den Fragebogen mit aufgenommenen Fragestellungen begrenzt ist. Ziel der Fragestellungen ist es, Erkenntnisse über die individuelle Meinung und Einstellung und die Gewohnheiten einer Einzelperson zu gewinnen. Eine solche Befragung kann entweder mündlich oder schriftlich erfolgen. Die häufigste Form sind schriftliche Befragungen, genauer gesagt Onlineumfragen. [22, S. 98]

Auch für diese Arbeit wurde eine Onlineumfrage durchgeführt, um Daten über das Verhalten der zukünftigen Nutzer zu erhalten. Zur Umfrage wurden alle Standort-Mitarbeiter der codecentric AG Nürnberg eingeladen. Außerdem wurden den Teilnehmern so oft wie möglich offene Fragen gestellt, oder - im Fall von geschlossenen Fragen - mit „Warum“ nachgehakt, um mehr zu erfahren. In Befragungen zur User Research sollten nämlich geschlossene Fragen, bei denen nur mit ja oder nein geantwortet werden kann, vermieden werden [36, S. 195]. Auf Grundlage der Auswertung der Onlineumfragen wurden im nächsten Schritt sogenannte Personas erstellt.

3.2.2 Personas

Personas sind prototypische Anwenderprofile. Eine Persona steht dabei stellvertretend für eine Gruppe an Nutzern, die die Anwendung später benutzen soll, und beschreibt die Eigenschaften, Verhaltensweisen und Motive der Nutzergruppe in Bezug auf eine Anwendung. Personas dienen dazu, die Bedürfnisse und Anforderungen der Nutzer während des Projekts im Auge zu behalten und diese entsprechend in die Entwicklung

der Anwendung einfließen zu lassen. Inhaltlich hat sich die folgende Darstellung bewährt: Abbildung eines realistischen Fotos der Person, Name und Alter der Person und stichpunktartige Zusammenfassung der wesentlichen Eckpunkte - teilweise auch über Grafiken, die den Grad der Technologiekompetenz oder Ähnliches aufzeigen. [22, S. 113-114]

In Abblildung 3.1 sind beispielhaft zwei Personas der Slack-App dargestellt. In diesem Fall werden tatsächlich gar nicht recht viel mehr Personas benötigt, da die Zielgruppe sehr klein ist und die meisten Standort-Mitarbeiter ähnlich viele Technologiekenntnisse usw. besitzen. Wäre die Zielgruppe allerdings größer, wären mindestens vier bis fünf Personas Pflicht, da es dann auch viel mehr verschiedene Charaktere und Interessen zu beschreiben gäbe [36, S. 88].

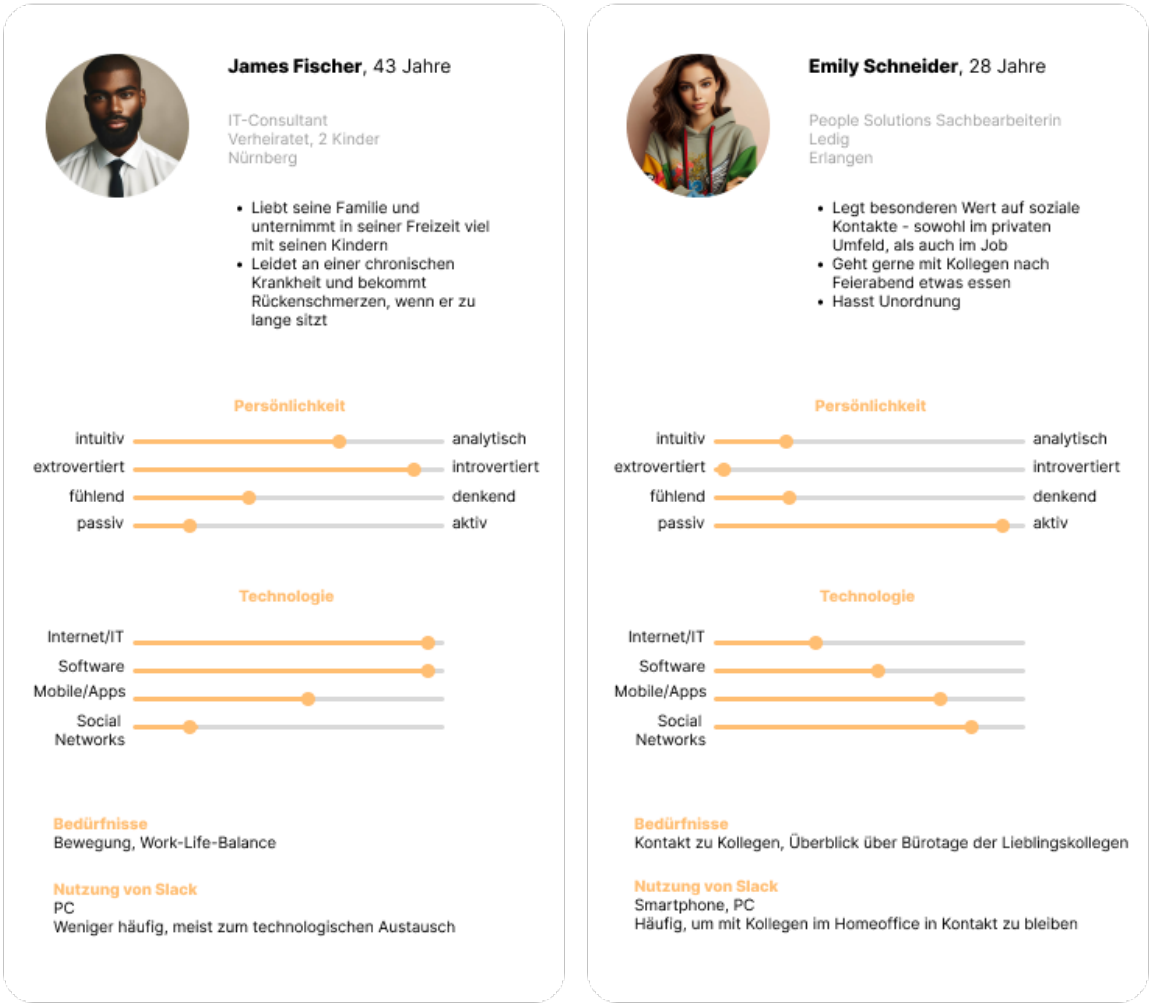

Abbildung 3.1: Personas (Quelle: Eigene Darstellung)

3.3 Zielplattform

An dieser Stelle sollte sich der App-Designer darüber Gedanken machen, für welche Plattform die App entwickelt wird, da diese Wahl entscheidend für die anschließende Umsetzung ist [36, S. 95]. In diesem Fall steht bereits fest, dass es sich um eine Slack-App handeln soll.

Vorsicht

Auch wenn die Rede von „Slack-App“ ist, handelt es sich in diesem Fall *nicht* - wie der Name vielleicht zuerst vermuten lässt - um eine Mobile-App, die im Google Play Store oder im App Store von Apple veröffentlicht wird und anschließend je nachdem auf einem Android- oder iOS-Gerät installiert werden kann. Die Slack-App, die in dieser Arbeit entwickelt wird, interagiert über das Web mit der Slack-API und wird online gehostet - daher ist sie eine Webanwendung. Sie unterscheidet sich jedoch in vielerlei Hinsicht von üblichen Webanwendungen.

Die Frage ist: Was ist das Besondere an einer Slack-App, und was muss bereits beim Design einer solchen Anwendung beachtet werden?

Besonderheiten von Slack
Wie bereits im Kapitel *Grundlagen* erläutert wurde, funktioniert eine Slack-App innerhalb der Plattform Slack und ist daher von dieser abhängig. Da Slack sowohl auf PCs als auch auf Smartphones genutzt wird, müssen die Inhalte der App sowohl für horizontale als auch vertikale Screens ausgelegt sein, und zudem auf kleineren genauso wie auf größeren Bildschirmen gut bedienbar sein. Doch diese Arbeit übernimmt größtenteils bereits die Nutzung von Slacks eigenem UI-Framework *Block Kit*, welches eine weitere Besonderheit beim Design von Slack-Apps ist: Jede Slack-App, die das Block Kit verwendet, sieht zumindest ähnlich aus, da die gleichen UI-Elemente genutzt werden. Viele Entscheidungen werden beim UI-Design also bereits abgenommen, doch führt dies oft auch zu Einschränkungen in der Freiheit des UI-Designers. Es sollte also vor der Umsetzung gut geplant werden, welche Benutzerelemente die App besitzen soll, und ob die Anordnung dieser Elemente von Slacks Block Kit auch genau so unterstützt wird, wie vorgestellt. Bevor man ein festes User Interface einplant, empfiehlt es sich, auf der Seite `https://app.slack.com/block-kit-builder/` auszuprobieren, ob das gewünschte Layout mit dem Block Kit umsetzbar ist. Der dritte Aspekt sind die verschiedenen Erscheinungsformen einer Slack-App. Es sollte gut überlegt werden, wie die App zum Einsatz kommt, bzw. wo sie aktiv wird: Handelt es sich um einen Bot in einem Channel? Sendet der Bot private Nachrichten? Stellt die App auf dem App-Home-Screen bestimmte Informationen oder Funktionen zur Verfügung?

3.4 Use Cases und User Stories

Die Ergebnisse der vorherigen Schritte des Design-Prozesses bilden eine wertvolle Grundlage für weitere Methoden in der Analyse- und Konzeptionsphase: Nutzungsszenarien bzw. Use Cases und User Stories. [22, S. 123]

3.4.1 Use Cases

Use Cases sind Anwendungsfälle oder Nutzungsszenarien und beschreiben, was der User konkret mit der Anwendung machen soll. Das heißt ein Anwendungsfall beschreibt, wie der Nutzer (auch als Akteur bezeichnet) mit der Anwendung interagiert, also welche Problemstellung er durch die Anwendung lösen kann, und wie diese wiederum darauf reagiert. Alle Use Cases zusammen definieren die funktionalen Anforderungen an die Anwendung. [22, S. 123]

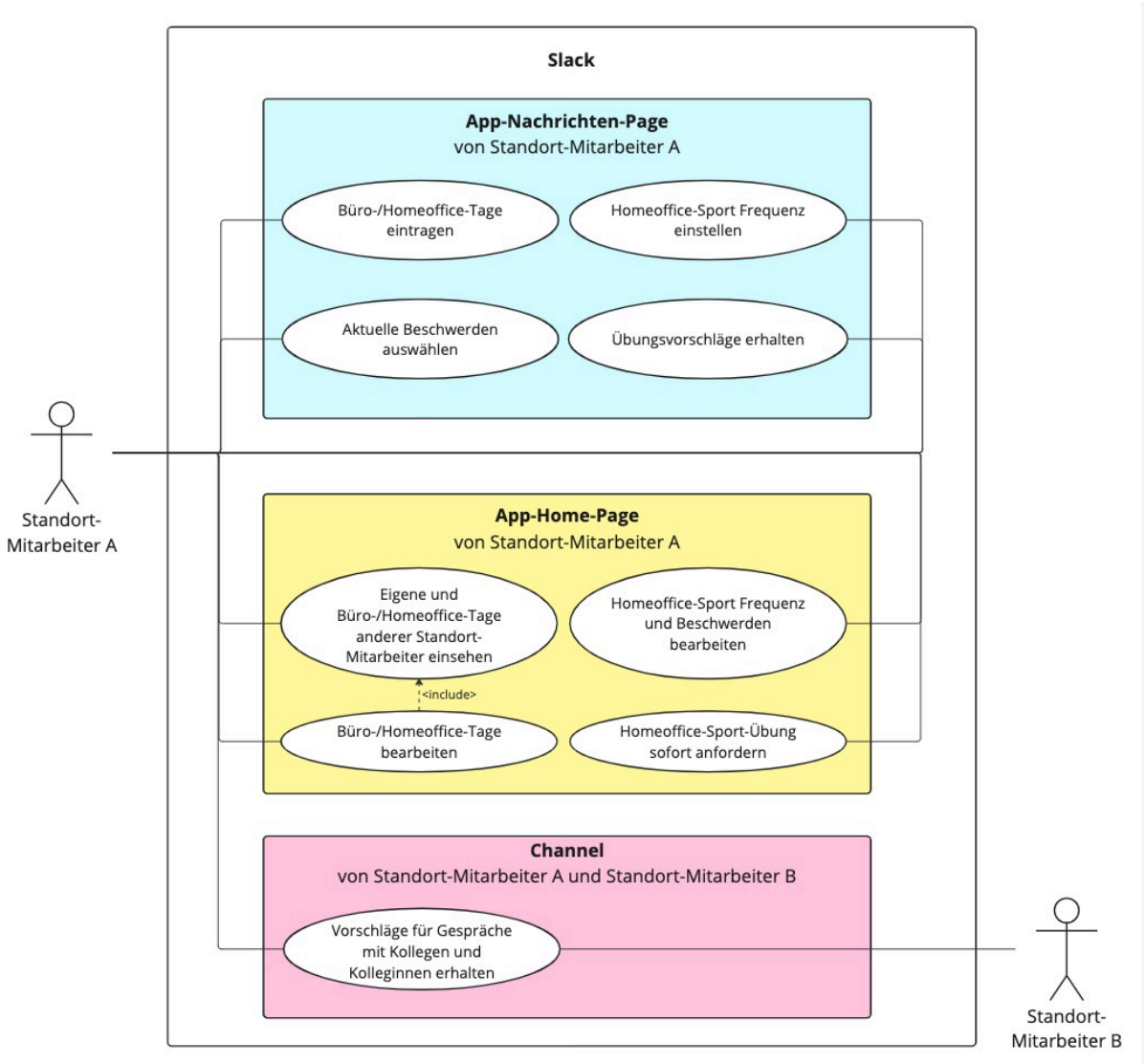

Abbildung 3.2: UML Use Case Diagramm der zu entwickelnden Slack-App (Quelle: Eigene Darstellung)

Die Use Cases können mithilfe eines UML Use Case Diagramms dargestellt werden. UML ist das Modellierungs-Toolkit zur Unterstützung bei der Modellierung von Diagrammen. Zur Erstellung von Anwendungsdiagrammen wird ein Satz spezieller Symbole und Konnektoren verwendet. In Tabelle 3.1 sind die wichtigsten UML-Modellierungen für Use Case Diagramme aufgelistet. [23]

Akteur	Strichmännchen
Anwendungsfall	Ovale Form mit entsprechender Beschriftung
Beziehung zum System	Linie zwischen Akteur und Anwendungsfall
Systemgrenze	Einrahmen eines oder mehrerer Anwendungsfälle mit einem Kästchen; Ein System kann wiederum eines oder mehrere Subsysteme haben

Tabelle 3.1: Die wichtigsten UML-Modellierungen für Use Case Diagramme

Die Abbildung 3.2 zeigt das Use Case Diagramm der zu entwickelnden Slack-App. Da eine Slack-App in verschiedenen Bereichen von Slack aktiv werden kann, wird nicht die App selbst als ein System dargestellt, sondern Slack als Plattform, und die verschiedenen Bereiche, in denen die App aktiv werden kann, als Subsysteme. Die Darstellung zeigt, dass Standort-Mitarbeiter A mit allen drei Subsystemen interagieren kann, während Standort-Mitarbeiter B nur mit einem der abgebildeten Subsysteme, nämlich dem Channel, den Standort-Mitarbeiter A und Standort-Mitarbeiter B gemeinsam haben, interagieren kann. Das liegt daran, dass sowohl App-Home als auch die App-Nachrichten-Page ein One-to-One Raum zwischen einem User und der App ist. Das heißt ein anderer User hat keinen Zugriff auf diesen Raum, aber er besitzt auf seiner Seite seine eigenen One-to-One Räume mit der Slack-App.

3.4.2 User Stories

User Stories sind kurze Geschichten aus der Sicht des Endbenutzers, die nicht nur die Interaktion zwischen dem User und der Anwendung darstellen, sondern auch den Kontext, also die Umgebung und Rahmenbedingungen, die ein Auslöser für eine Handlung des Nutzers sein können [22, S. 123]. Der Zweck einer User Story ist es, zu zeigen, welchen Wert ein Softwarefeature für einen Kunden oder Nutzer hat.

Eine User Story ist die kleinste Arbeitseinheit in agilen Frameworks wie Scrum und Kanban. In Scrum beispielsweise werden User Stories zu Sprints hinzugefügt und im Verlauf des Sprints „abgearbeitet“, indem die nötigen Softwarefeatures implementiert werden. User Stories werden oft als einfacher Satz formuliert und sind wie folgt aufgebaut: „Als [User] [möchte] ich, [damit].“ Man spricht dabei oft von *Kundentyp* + *Anforderung* + *Zweck*. Diese Struktur hilft dabei, zu definieren, wann eine Entwicklungsaufgabe oder Story erledigt ist. Im Allgemeinen ist eine Story „erledigt“, wenn der Benutzer die beschriebene Aufgabe ausführen kann. Um dies besser beurteilen zu können, werden pro User Story häufig auch sogenannte „Akzeptanzkriterien“ aufgelistet, die beschreiben, wann eine User Story als erledigt gilt. [31]

Im Folgenden werden die User Stories der zu entwickelnden Slack-App aufgelistet. Neben der genauen Beschreibung inklusive Akzeptanzkriterien besitzt jede User Story jeweils einen Bezeichner, einen Namen sowie eine Priorität. Letztere ist ein wichtiger Punkt bei der Projektplanung: Wichtigere User Stories werden vor weniger wichtigen

bearbeitet.

US-1 Büro-/Homeofficetage eintragen

Als Remote-Arbeiter möchte ich eintragen können, wann ich vorhabe, im Büro zu sein, damit sich meine Kollegen daran orientieren können.

Akzeptanzkriterien:

* Im System können Wochentage als Bürotage markiert werden
* Die im System ausgewählten Tage werden in einer Datenbank gespeichert

Priorität: Hoch

US-2 Eigene und Büro-/ Homeoffice-Tage anderer Standort-Mitarbeiter einsehen

Als Remote-Arbeiter möchte ich sehen können, welche meiner Kollegen vorhaben, wann im Büro zu sein, damit ich besser entscheiden kann, wann ich selbst ins Büro fahre, und Events besser planen kann.

Akzeptanzkriterien:

* Die in der Datenbank gespeicherten Bürotage aller Mitarbeiter werden vom System geladen und angezeigt

Priorität: Hoch

US-3 Büro-/ Homeoffice-Tage bearbeiten

Als Remote-Arbeiter möchte ich die Möglichkeit haben, meine Homeoffice-Planung zu ändern, und dies für meine Kollegen sichtbar zu machen, da ich oft aufgrund von Bahnstreiks oder Ähnlichem spontan doch nicht ins Büro fahren kann.

Akzeptanzkriterien:

* Selbst angelegte Bürotage können im System bearbeitet bzw. gecancelt werden
* Die Daten werden neu in der Datenbank gespeichert

Priorität: Hoch

US-4 Homeoffice-Sport sofort starten

Als IT-Consultant möchte ich die Möglichkeit haben, zu einem beliebigen Zeitpunkt eine Sportübung durchzuführen, damit ich akuten Beschwerden sofort entgegenwirken kann.

Akzeptanzkriterien:

* Das System stellt dem User die Übungen zu jedem beliebigen Zeitpunkt zur Verfügung

Priorität: Hoch

US-5 Individuelle Homeoffice-Sport-Übungen

Als IT-Consultant möchte ich meine Homeoffice-Sport-Übungen an meine aktuellen körperlichen Beschwerden anpassen können, um diese zu lindern und meinem Körper etwas Gutes zu tun.

Akzeptanzkriterien:

* Der User kann im System aus einer Auswahl an Beschwerden wählen
* Die ausgewählten Beschwerden werden (neu) in der Datenbank gespeichert
* Die gespeicherten Beschwerden werden bei der Auswahl der Übungsvideos berücksichtigt

Priorität: Hoch

US-6 Homeoffice-Sport-Frequenz einstellen

Als IT-Consultant möchte ich einstellen können, wie oft Interesse an Homeoffice-Sport besteht, damit ich nicht zu oft gestört, aber trotzdem an genügend Bewegung erinnert werde.

Akzeptanzkriterien:

* Das System lässt den User die Anzahl der täglichen Sporteinheiten auswählen
* Die ausgewählte Anzahl wird in der Datenbank gespeichert

Priorität: Niedrig

US-7 Homeoffice-Sport-Frequenz bearbeiten

Als IT-Consultant möchte ich die Möglichkeit haben, die Homeoffice-Sport-Frequenz zu bearbeiten, damit ich diese situationsabhängig anpassen kann.

Akzeptanzkriterien:

* Die ausgewählte Anzahl der täglichen Sporteinheiten wird vom System angezeigt und kann bearbeitet werden
* Die Daten werden neu in der Datenbank gespeichert

Priorität: Niedrig

US-8 Homeoffice-Sport-Übungen vorgeschlagen bekommen

Als IT-Consultant, der den ganzen Tag am Schreibtisch arbeitet, möchte ich an genügend Bewegung erinnert werden, und zu passenden Übungen angeleitet werden, damit meine körperlichen Beschwerden reduziert werden.

Akzeptanzkriterien:

* Das System gibt täglich anhand der in der Datenbank gespeicherten Anzahl und der bereits versendeten Sportübungen Übungsvorschläge in Form von zeitgesteuerten Nachrichten, welche Videos mit Übungen enthalten

Priorität: Niedrig

US-9 **Vorschläge für Gespräche mit Kollegen und Kolleginnen erhalten**

Als Remote-Arbeiter möchte ich Vorschläge bekommen, wie ich mit meinem Team auf einem einfachen Weg ein Gespräch beginnen könnte, um trotz der räumlichen Entfernung den Kontakt zu Kollegen und Kolleginnen zu pflegen.

Akzeptanzkriterien:

* Das System versendet in einem Channel eine Nachricht, die ein Gespräch zwischen Kollegen anregen soll

Priorität: Niedrig

3.5 Flussdiagramme und Wireframes

Bevor es an die Wireframes geht, sollte mit Stift und Papier eine erste Skizze der App (auch „Scribble“ genannt) angefertigt werden, um eine grobe Vorstellung davon zu bekommen, wie die App strukturiert ist, und um Ideen schneller greifbar zu machen. Außerdem kann mit einem Flussdiagramm (oder auch „Flowchart“) gearbeitet werden, um den Ablauf des Programms darzustellen, das heißt alle Wege, die der Nutzer in der Applikation gehen kann, zu visualisieren (vgl. Abbildung 3.3). Diese Wege sollten nach Möglichkeit abgekürzt werden, aber lediglich soweit, dass die Usability nicht darunter leidet. [36, S. 430-433]

Mit einem Wireframe nähert man sich dem fertigen Produkt noch weiter an. Wireframes werden im Gegensatz zu Scribbles mit Software erstellt und gehen deutlich mehr ins Detail. Dennoch sind sie immer noch nicht auf die finale Gestaltung konzentriert, sondern lediglich auf die Inhalte und Funktionen einer Anwendung. [22, S. 157-158]

Das bedeutet, sie enthalten im Gegensatz zu einem Mock-up noch keine Farben, Bilder, Grafiken oder unterschiedliche Typografie, sondern konzentrieren sich mehr auf die von Semler und Tschierschke [36, S. 435] aufgeführten Aspekte:

1. **Struktur**: In welcher Relation stehen die einzelnen Screens zueinander?
2. **Inhalt**: Was zeigt jeder einzelne Screen?
3. **Informationsstruktur**: Welche Navigationsstruktur liegt zugrunde? Wie werden die Informationen organisiert und dargestellt?
4. **Funktionalität**: Wie funktioniert jeder einzelne Screen?

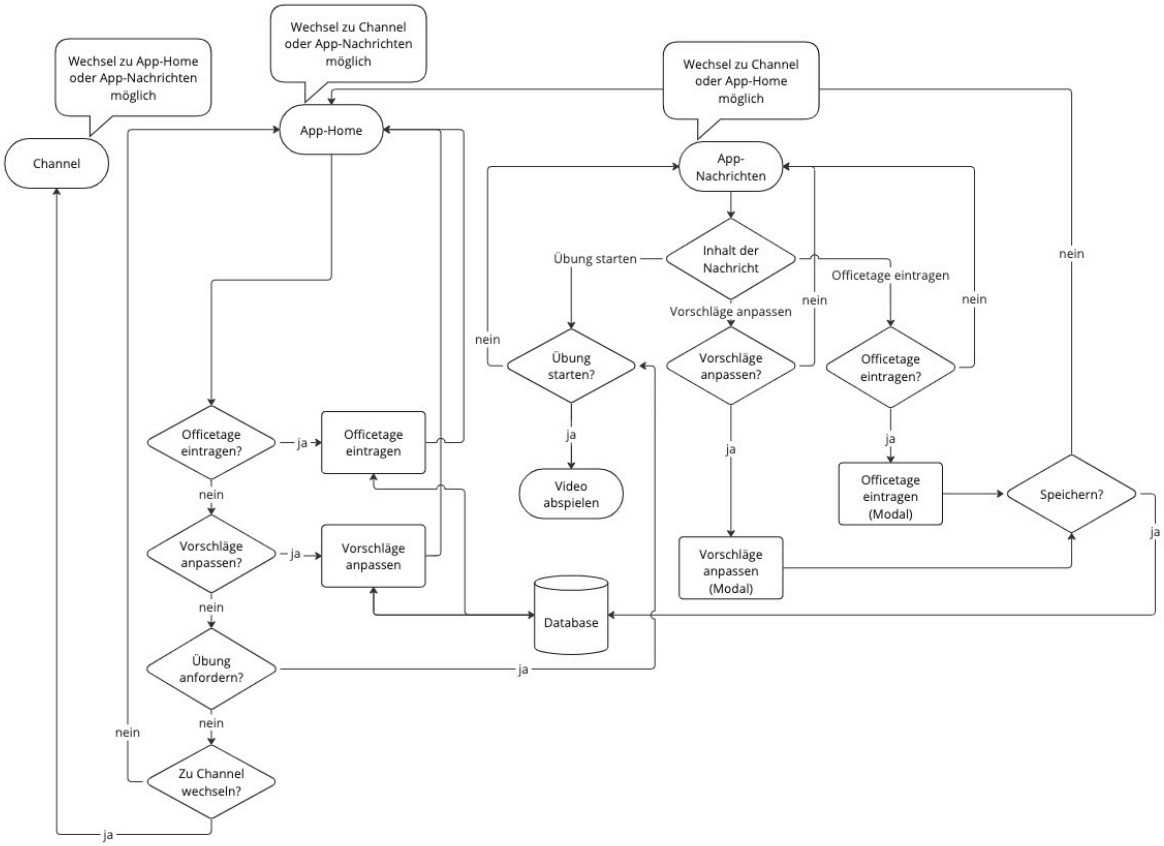

Abbildung 3.3: Flussdiagramm der zu entwickelnden Slack-App (Quelle: Eigene Darstellung)

5. **Verhalten**: Wie interagiert der Anwender mit dem Screen? Und wie verhält sich dieser Screen?
6. Sind **Empty States und Fehlermeldungen** in den Wireframes inkludiert?

Wireframes bieten einem Entwickler auch die Möglichkeit, den Entwurf auf Schwachstellen zu analysieren und rechtzeitig Lösungs- oder Verbesserungsvorschläge einzubringen. [36, S. 437]

Für diese Arbeit wurde das Tool *Figma* zum Erstellen der Wireframes eingesetzt. Es sind viele Design-Apps am Markt verfügbar, die für diesen Zweck verwendet werden können. Die Entscheidung für ein Tool kann je nach persönlichen Präferenzen und Anforderungen an die Funktionen des Tools getroffen werden. Aber um ein paar Vorteile von Figma zu nennen: Figma ermöglicht eine kollaborative Echtzeit-Bearbeitung von Designs im Team, bietet Plugins und Widgets, generiert bereits CSS und ist nicht nur eine Design-App, sondern auch eine Community-Plattform, auf der Ideen und Lösungen geteilt werden können. [50, S. 5]

3.6 Mock-ups und Prototypen

Auf Basis der angefertigten Wireframes werden anschließend Mock-ups erstellt. Ein Mock-up ist eine detailgetreue Nachbildung einer App und dient dazu, dem Entwickler vorzugeben, wie das User Interface der App zu programmieren ist. Das bedeutet: Farben, Formen, Typografie und sogar Ton werden in dieser Phase konkretisiert. In diesem Schritt bekam die Slack-App auch endlich einen Namen, nämlich *beecaring*. Der Name „beecaring“ entstand aufgrund des inoffiziellen Maskottchens des Nürnberger codecentric Teams, einer Biene (bee), sowie der App-Funktionalität, die sich um die Standortpflege und das Wohlergehen der Mitarbeiter kümmert (caring). Nachdem der Name feststand, wurde für beecaring ein passendes Corporate Design entworfen, welches in Anhang C abgebildet ist. Die Farben des Corporate Designs stellen eine Kombination der Unternehmensfarben von codecentric (türkis) und Slack (lila) dar. Die Mock-ups beinhalten bereits die Logos, die im Corporate Design enthalten sind. Ein Vergleich von Wireframe und Mock-up ist in Abbildung 3.4 dargestellt. Mithilfe der Mock-ups kann das Design- und UX-Konzept der App diskutiert werden, bevor mit der aufwendigen technischen Implementierung der Funktionen begonnen wird. [36, S. 438-439]

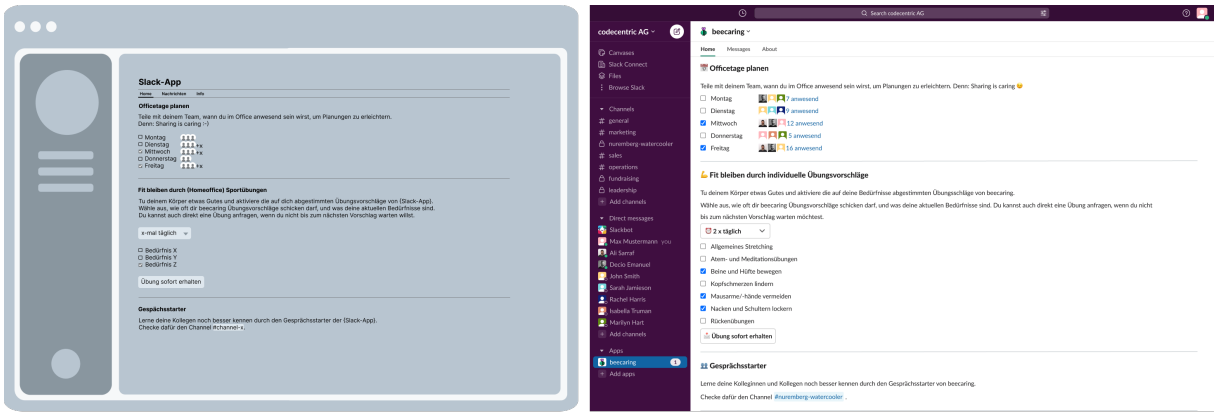

Abbildung 3.4: Wireframe vs. Mock-up (Quelle: Eigene Darstellung)

Bereits beim Erstellen der Wireframes, aber besonders beim Design der Mock-ups sollte stets auf die Einhaltung von Gestaltgesetzen geachtet werden, welche beispielsweise auch von Jacobsen und Meyer ausführlich [22, S. 43-67] beschrieben werden. In dieser Arbeit wird nicht näher auf die genauen Inhalte der Gesetze eingegangen. Für das zu bearbeitende Projekt wurde wie bereits bei den Wireframes das Tool Figma zur Erstellung der Mock-ups eingesetzt. Da die Slack-App ein ganz spezifisches UI haben soll, wurde mit einem Template der Figma Community gearbeitet, das bereits alle möglichen Elemente des Block Kits und des Design Systems von Slack beinhaltet. Die einzelnen Elemente mussten also nur noch so angebracht und modifiziert werden, dass die Inhalte denen der zuvor erstellten Wireframes entsprachen.

Ein klassisches Mock-up ist immer noch statisch, doch in den meisten Fällen möchte man ein interaktives Model testen, bevor man es richtig entwickelt. An dieser Stelle ist es sinnvoll, einen interaktiven Prototyp zu erstellen. Hierbei liegt der Fokus auf dem Simulieren des Verhaltens und der Funktionalität der App. Dadurch kann getestet

werden, ob die Lösungen, die in den vorherigen Schritten entwickelt wurden, auch wirklich wie gewünscht funktionieren. [36, S. 440-441]

Hierfür können die Mock-ups aus dem vorherigen Schritt und das Flussdiagramm zusammengeführt werden. Das bedeutet, die Mock-ups, besser gesagt die einzelnen Screens, müssen nur noch so verlinkt werden wie im Flussdiagramm angegeben. Auch dafür eignet sich Figma bestens: Wenn bereits Mock-ups vorliegen, können diese ganz einfach verlinkt werden, und die möglichen Szenarien können prototypisch durchgespielt werden. Spätestens beim Durchspielen des Prototyps am Ende einer Design-Prozess-Iteration sollte erkannt werden, ob es noch Schwachstellen im Designentwurf gibt. Wenn dies der Fall ist, können weitere Iterationen über den ganzen Prozess eingeleitet werden, solange, bis der Prototyp zufriedenstellend ist.

Kapitel 4

Architektur und Serverless Cloud-Deployment

Wenn alle Anforderungen vorliegen, und ein erster Prototyp existiert, kann mit der Planung der Architektur und der Infrastruktur begonnen werden. Wie bereits in Kapitel 3.3 erwähnt, sind Slack-Apps Webservices. Das bedeutet, Slack-Apps werden online gehostet. Dies kann entweder über eine private Infrastruktur - zum Beispiel einen firmeneigenen Server, ein privates Cloud-Setup oder Ähnliches - oder aber auch über einen Public Cloud Provider geschehen. In diesem Projekt fiel die Wahl auf ein Cloud-Deployment, genauer gesagt auf ein Serverless Cloud-Deployment. Was dies genau bedeutet, und welche Vorteile ein solches Deployment mit sich bringt, wird im Folgenden erläutert. Zudem wird auf die einzelnen Komponenten, die dafür benötigt werden, eingegangen.

Der Autor und Entwickler Daniel Stender vergleicht Public Cloud Computing mit Carsharing [51, S. 20]: Bei beiden handle es sich zunächst einmal um ein Mietangebot. Und bei dem Angebotsmodell *Infrastructure-as-a-Service* würden statt Kraftfahrzeugen reine Computing-Ressourcen wie CPU-Leistung, Arbeitsspeicher, Massenspeicher und Netzwerke für die allgemeine Benutzung zur Verfügung gestellt. Die angemieteten Ressourcen werden dabei im weltweiten Breitband-Netzwerk über Standardprotokolle zur Verfügung gestellt und es spielt grundsätzlich keine Rolle, wo genau sich die genutzten Rechner befinden.

Ein relativ neuer Ansatz im Cloud Computing ist das sogenannte *Serverless Computing*, bzw. *Function-as-a-Service* (FaaS). Während Entwickler beim herkömmlichen Cloud-Computing-Modell virtuelle Server bereitstellen, konfigurieren und verwalten müssen, um ihre Anwendungen auszuführen, übernimmt beim Serverless Computing der Cloud-Provider die Verantwortung für die Bereitstellung und Skalierung der Infrastruktur, die für die Ausführung der Anwendung erforderlich ist. Entwickler müssen sich dadurch nur auf den Code ihrer Anwendung konzentrieren, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen. FaaS-Dienste sind zum Beispiel *AWS Lambda, Microsoft Azure Functions* und *Google Cloud Functions*, die jeweils verschiedene Programmiersprachen unterstützen. FaaS-Funktionen können auf alle

Services des jeweiligen Anbieters zugreifen und in der Cloud durch verschiedene Ereignisse ausgelöst werden. Ein Vorteil von FaaS-Funktionen ist, dass diese nur berechnet werden, wenn sie auch laufen, und Entwickler dadurch nur für die tatsächliche Nutzung ihrer Anwendung bezahlen, anstatt für die kontinuierliche Bereitstellung von Servern, unabhängig davon, ob diese genutzt werden oder nicht. Amazon berechnet für Lambda zum Beispiel die Anzahl der Funktionsaufrufe, die tatsächliche Laufzeit und den benötigten Arbeitsspeicher. [51, S. 28]

In den folgenden Abschnitten werden die wichtigsten Komponenten, welche für das Serverless Cloud-Deployment dieses Projekts benötigt werden, genauer betrachtet und erläutert. Es ist zu beachten, dass es viele verschiedene Möglichkeiten gibt, eine Slack-App zu entwickeln, und dass nicht immer ein Cloud-Deployment gewählt werden muss. Selbst wenn die Wahl auf ein Cloud-Deployment fällt, gibt es noch etliche andere Wege, die zum gleichen Ziel führen können - beispielsweise die Nutzung von Microsoft Azure oder Google Cloud. In diesem Projekt wird jedoch hauptsächlich mit AWS gearbeitet. Daher wird im Folgenden vor allem auf verschiedene Services von AWS eingegangen, die im weiteren Verlauf zur Slack-App-Entwicklung verwendet werden.

4.1 AWS Lambda

In diesem Abschnitt wird AWS Lambda als zentraler Bestandteil des Serverless Cloud-Deployments für dieses Projekt näher betrachtet. Zudem wird ein Vergleich von AWS Lambda und AWS Fargate durchgeführt, anhand dessen aufgezeigt wird, wieso die Wahl für dieses Projekt auf Lambda gefallen ist.

4.1.1 Grundlagen

Wie bereits erwähnt, handelt es sich bei AWS Lambda um einen FaaS-Dienst, der es ermöglicht, Code auszuführen, ohne dabei Server verwalten zu müssen [42]. Lambda führt die Funktion nur bei Bedarf aus und skaliert automatisch. Daher zahlt der Benutzer nur für die Rechenzeit, die wirklich verbraucht wird und es fallen keine Kosten an, wenn der Code nicht ausgeführt wird. Lambda kann für unterschiedliche Zwecke verwendet werden, wobei meistens nie Lambda alleine, sondern mindestens ein weiterer Service in Verbindung mit Lambda verwendet wird. Häufig wird Lambda in folgenden Kombinationen genutzt [7]:

- Mit *Amazon S3*: Zur Echtzeitverarbeitung von Dateiuploads, zum Beispiel zur Validierung von Dateien
- Mit *Amazon Kinesis*: Zur Echtzeitverarbeitung von Streaming-Daten, zum Beispiel für Echtzeit-Analysen oder Überwachung
- Mit *Amazon DynamoDB*: Zur Ausführung von Datenbanktrigger-Funktionen, zum Beispiel zur Validierung oder Aktualisierung von Daten

- Mit *Amazon EventBridge*: Zur Reaktion auf bestimmte Ereignisse in AWS-Diensten oder benutzerdefinierte Ereignisse
- Mit *Amazon API Gateway*: Zur Erstellung von serverlosen APIs, zum Beispiel für Microservices-Architekturen oder Webanwendungen

Einige der genannten Services werden in diesem Kapitel nachfolgend noch genauer erläutert. Lambda ermöglicht die Integration anderer AWS-Dienste in der Regel durch die Einrichtung von Triggern, die bestimmte Ereignisse auslösen und dann eine Lambda-Funktion aufrufen. Die Lambda-Funktion kann dann weitere Aktionen ausführen und mit anderen AWS-Services interagieren. Ein Beispiel wäre Folgendes: Die Lambda-Funktion wird durch einen API Gateway ausgelöst, was bedeutet, dass ein HTTP-Request eingegangen ist. Die Lambda-Funktion verarbeitet daraufhin diese Anfrage und speichert Daten in einer DynamoDB-Tabelle. Damit hätte man eine serverlose Web API aufgebaut (Abbildung 4.1).

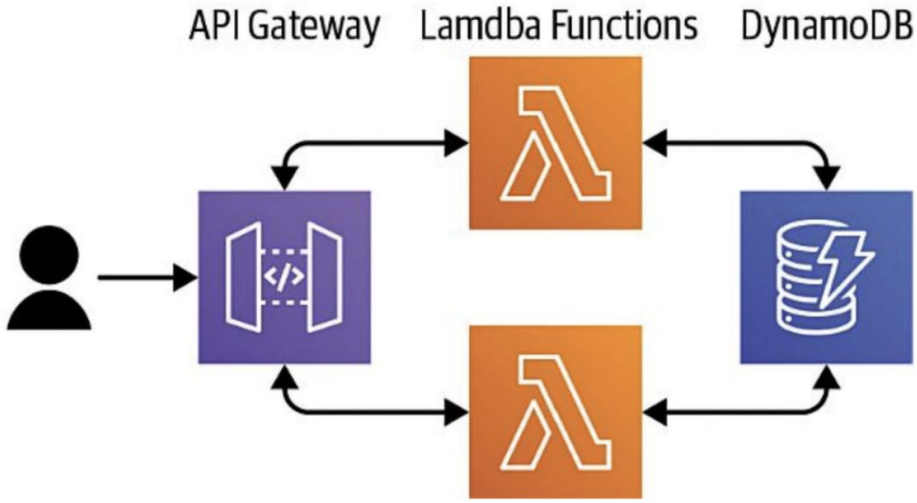

Abbildung 4.1: Aufbau einer Web API mithilfe von AWS Lambda (Quelle: [12, S. 28])

Insgesamt ermöglicht diese Architektur eine flexible und skalierbare Verbindung zwischen verschiedenen AWS-Services, wodurch komplexe Workflows einfach implementiert werden können.

Lambda selbst bringt einige nützliche Features mit sich. Beispielsweise unterstützt der Service die Verwendung von Umgebungsvariablen, die es ermöglichen, Konfigurationswerte oder andere Daten zwischen verschiedenen Ausführungen der Funktion zu übergeben. Ein weiterer Punkt sind Versionen: Über Lambda können verschiedene Versionen der Funktion verwaltet und verfolgt werden. Außerdem ermöglichen Container-Images, die Funktion in einem Container auszuführen, was zusätzliche Flexibilität und Kontrolle bietet. Darüber hinaus können AWS-Lambda-Funktionen als HTTP-Endpunkte verfügbar gemacht werden, was Entwicklern eine einfache Möglichkeit bietet, serverlose APIs zu erstellen. [7]

Natürlich bieten bereits auch einige andere Cloudanbieter FaaS-Dienste an. Laut Chapin und Roberts verfügt AWS Lambda jedoch über mehr Kapazität, mehr Reife und mehr Integrationspunkte als jede andere FaaS-Plattform [12, S. 27].

4.1.2 Lambda vs. Fargate

Wenn es zum Serverless Computing mit AWS kommt, wird statt Lambda oft auch ein anderer Service verwendet: *AWS Fargate*. Das Orchestrieren von Containern ist eine Sache, aber die Verwaltung der zugrunde liegenden Recheninfrastruktur ist eine ganz andere Herausforderung - AWS Fargate führt Container aus, ohne dass Entwickler jemals über die zugrundeliegende Infrastruktur nachdenken müssen. Das Ziel ist, die Rechenebene vollständig zu abstrahieren, damit der Entwickler sich lediglich auf die Verwaltung seiner Container-Workloads konzentrieren kann. Hier kommt der serverlose Aspekt von Fargate ins Spiel: Anstatt einzelne virtuelle Maschinen hochzufahren und zu verwalten, teilt der Entwickler Fargate mit, welche Rechen- und Speicherressourcen er benötigt, und Fargate kümmert sich um die Details. Fargate kann entweder in Kombination mit *AWS Elastic Container Service* oder *AWS Elastic Kubernetes Service* verwendet werden - je nachdem, ob Kubernetes zur Orchestrierung verwendet wird (EKS), oder ein eher einfaches Container Management Setup ausreichend ist (ECS). Ohne die Verwendung von Fargate müssen nochmals unterschiedliche Tools kombiniert werden, um die VMs für Container in Bezug auf Skalierung, Konfiguration und Überwachung zu verwalten. Fargate jedoch nimmt dem Entwickler diese Aufgaben ab und macht diese Verwaltungsebene quasi unsichtbar. [34]

AWS Fargate und AWS Lambda gehören zu verschiedenen Kategorien von Serverless-Diensten, doch sie teilen die Eigenschaft, dass sie Entwicklern die Komplexität der Workload-Verwaltung abnehmen. In Tabelle 4.1 sind beispielhaft einige Kategorien aufgelistet, anhand deren Mike Rosam [34] Lambda und Fargate miteinander vergleicht.

Was spricht nun gegen die Verwendung von AWS Fargate zum Serverless Deployment einer Slack-App? Tatsächlich kann auch Fargate für das Deployment der App gewählt werden - Die Wahl hängt jedoch immer ganz von den spezifischen Anforderungen an die Anwendung ab.

Einige Szenarien, in denen Fargate besonders nützlich ist, sind laut Rosam:

- Aufbau einer Microservices-Architektur, besonders wenn die Microservices lange laufen oder sich die Nachfrage schnell ändern kann
- Batch-Verarbeitungsaufgaben, bei denen Ressourcen für eine bestimmte Aufgabe bereitgestellt und dann wieder abgebaut werden
- Webanwendungen und APIs, insbesondere, wenn kontinuierlich laufende Komponenten erforderlich sind und die Nachfrage nach Ressourcen stark schwankt

Für Lambda eignen sich hingegen folgende Anwendungsfälle besonders gut:

- Serverlose API-Backends, bei denen AWS-Lambda-Funktionen in Reaktion auf REST- oder HTTP-Anfragen über Amazon API Gateway arbeiten
- Ereignisgesteuerte Datenverarbeitung, bei der AWS-Lambda-Funktionen auf Ereignisse reagieren, eine Verarbeitung durchführen und das Ergebnis zurückgeben
- Automatisierte Backend-Aufgaben, die zeitgesteuert ausgeführt werden müssen, zum Beispiel Datenbank-Backups, Protokollbereinigung oder das Versenden von

Benachrichtigungen zu bestimmten Zeiten

Kategorie	AWS Lambda	AWS Fargate
Rechenmodell	Serverless Computing Service, der Code als Reaktion auf Ereignisse ausführt und die Rechenressourcen automatisch verwaltet	Container Management Service, der Docker-Container ausführt, ohne Server oder Cluster zu verwalten
Preismodell	Der Preis wird für jede 100ms, die der Code läuft, und nach der Anzahl, wie oft die Funktion getriggert wird, berechnet	Die Gebühr richtet sich nach der Menge an vCPU- und Arbeitsspeicherressourcen, die die Containeranwendung anfordert
Ausführungszeit	Auf 15 Minuten begrenzt	Geeignet für Anwendungen mit langer Laufzeit
Skalierung	Soft-Limit von 1.000 gleichzeitig ausgeführten Funktionen pro Region, kann jedoch auf Anfrage erhöht werden	Über ECS können weitere Container hinzugefügt werden, wobei Fargate die zugrundeliegende Infrastruktur automatisch skaliert, um der Nachfrage gerecht zu werden
Deployment-Modell	Ereignisse lösen Funktionen aus	Die ECS- oder EKS-Orchestrierung gibt die von den Containern benötigten Ressourcen an und Fargate kümmert sich um die Details

Tabelle 4.1: Fargate vs. Lambda: direkter Vergleich

Im Wesentlichen eignet sich Fargate also gut für Anwendungen mit variablen Ressourcenanforderungen und Lambda für ereignisgesteuerte, serverlose Verarbeitungsaufgaben. Da es in dieser Bachelorarbeit um die Entwicklung einer Slack-App geht, die stark von ereignisgesteuerten Aktionen abhängig ist, und HTTP-Anfragen an das serverlose Backend senden wird, fällt in diesem Fall die Wahl auf AWS Lambda. Zudem wird das Deployment über AWS Lambda auch auf der Dokumentationsseite der Slack-API vorgeschlagen [42].

4.2 Slack-Requests über Amazon API Gateway

Wie bereits erwähnt worden ist, wird als Trigger von AWS-Lambda-Funktionen häufig Amazon API Gateway verwendet. Auch für dieses Projekt wird API Gateway benötigt, um eine nahtlose Integration zwischen Lambda und der Slack-Anwendung zu ermöglichen, indem API Gateway einen standardisierten HTTP-Endpunkt bereitstellt,

über den Anfragen an die Lambda-Funktionen geroutet werden können [11, S. 17]. Die URL des API-Endpunkts wird später als Request-URL für die Slack-App benötigt. Das bedeutet: Durch eine Benutzerinteraktion mit der Slack-App, zum Beispiel durch das Klicken eines interaktiven Buttons, sendet Slack eine HTTP-Anfrage an die in den App-Einstellungen konfigurierte Request-URL. Diese Anfrage enthält die vom Benutzer bereitgestellten Informationen sowie Details zur Art der Interaktion. Die von Slack gesendete Anfrage erreicht den API-Endpunkt, der in API Gateway konfiguriert ist. API Gateway empfängt die Anfrage und routet sie an die entsprechende AWS-Lambda-Funktion weiter. Die Lambda-Funktion wird automatisch gestartet und erhält die Slack-Anfrage als Eingabe. Die Funktion führt die erforderliche Logik aus, um auf die Slack-Anfrage zu reagieren. Dies kann die Ausführung von Logik jeglicher Art, das Abrufen oder Aktualisieren von Daten in externen Datenbanken oder die Generierung einer Antwork für den Benutzer beinhalten. Nach der Verarbeitung der Anfrage erstellt die AWS-Lambda-Funktion eine HTTP-Antwort. Diese Antwort kann verschiedene Inhalte enthalten, wie zum Beispiel Textnachrichten, interaktive Elemente oder andere Slack-spezifische Daten. Die Lambda-Funktion sendet die generierte HTTP-Antwort zurück an API Gateway und API Gateway leitet sie als HTTP-Antwort an Slack zurück. Schließlich empfängt der User die Antwort in der Slack-App.

4.3 Docker und Amazon Elastic Container Registry (ECR)

Kurz gesagt ist *Docker* eine Softwareplattform, die sicherstellt, dass der Code einer Anwendung in jeder Umgebung zuverlässig läuft. Damit das funktioniert, verpackt Docker Software in standardisierte Einheiten („Container“), die alles enthalten, was zum Ausführen der Software erforderlich ist, also nicht nur Code, sondern auch Bibliotheken, Systemtools und Laufzeit. [6]

Die gesamte Anwendungslogik der Slack-App als Lambda-Funktion direkt über AWS zu erstellen, ist aufgrund der Größe der Anwendung wohl keine praktikable Lösung. Deshalb bietet die Nutzung von Docker bzw. Container-Images eine geeignete Alternative. Die Erstellung der Container-Images erfolgt durch die Definition eines Dockerfiles im Bolt-Projekt. Ein Dockerfile spezifiziert die erforderlichen Abhängigkeiten und Konfigurationen der Anwendung. Dies ermöglicht eine reproduzierbare und konsistente Bereitstellung der Anwendungsumgebung. Mithilfe von Docker können die Container-Images außerdem anschließend in die *AWS Elastic Container Registry (ECR)* gepusht werden, wo sie zentral gespeichert und verwaltet werden können. Befindet sich ein Container-Image in ECR, kann es in AWS Lambda integriert werden. Durch die Auswahl des entsprechenden Images in Lambda wird die Anwendung deployt.

4.4 DynamoDB

Amazon DynamoDB ist ein vollständig verwalteter *NoSQL*-Datenbankdienst, durch den sich der Entwickler nicht mehr um Hardwarebereitstellung, Einrichtung und Konfiguration, Replikation, Software-Patches oder Clusterskalierung kümmern muss,

und somit der Verwaltungsaufwand für den Betrieb und die Skalierung einer verteilten Datenbank reduziert wird. Die Datenbanktabellen können in eine beliebige Datenmenge speichern und dies auch langfristig: Der Dienst gewährleistet eine hohe Verfügbarkeit und Langlebigkeit, indem er die Daten automatisch auf ausreichend viele Server verteilt, um Durchsatz- und Speicheranforderungen zu erfüllen. Daten werden auf SSDs gespeichert und automatisch über mehrere Verfügbarkeitszonen in einer AWS-Region repliziert, was eine integrierte Hochverfügbarkeit und Datenbeständigkeit bietet. Durch die Verwendung globaler Tabellen können DynamoDB-Tabellen in allen AWS-Regionen synchronisiert werden. Zudem bietet DynamoDB bedarfsgesteuerte Backup-Funktionen zur Sicherung, also zur langfristigen und gesetzlich vorgeschriebenen Aufbewahrung und Archivierung von Tabellen. Der Dienst ermöglicht auch eine Point-in-Time-Wiederherstellung für Tabellen, um sie vor versehentlichen Schreib- oder Löschvorgängen zu schützen. Abgelaufene Elemente können automatisch gelöscht werden, um die Speichernutzung und -kosten zu reduzieren. [5]

Bereits in den Akzeptanzkriterien der User-Stories war von einer Datenbank die Rede, doch wieso fällt die Wahl in diesem Projekt gerade auf DynamoDB, bzw. auf eine NoSQL-Datenbank?

4.4.1 NoSQL-Datenbanken vs. Relationale Datenbanken

Eine relationale Datenbank oder ein relationales Datenbankmanagementsystem speichert Informationen in Tabellen, wobei diese Tabellen oft in Relation zueinander stehen, nämlich wenn sie über gemeinsame Informationen verfügen. Die Spalten einer Tabelle definieren die zu speichernden Informationen, während die Zeilen für die tatsächlichen Daten verwendet werden. Jede Tabelle muss eine Spalte besitzen, die eindeutige Werte enthält - sogenannte „Primärschlüssel“. Anhand dieses Schlüssels lässt sich - im Fall einer Beziehung zwischen zwei Tabellen - in einer anderen Tabelle auf die in Relation stehende Tabelle referenzieren. Die Spalte, die den Primärschüssel einer anderen Tabelle beinhaltet, wird dann als „Fremdschlüssel“ bezeichnet. Die gebräuchlichste Art der Interaktion mit relationalen Datenbanken ist die Verwendung von *SQL (Structured Query Language)*. Entwickler können verschiedene SQL-Abfragen nutzen, um sogenannte *CRUD*-Vorgänge auf einer Datenbank auszuführen. CRUD steht dabei für Create (Erstellen), Read (Lesen), Update (Aktualisieren) und Delete (Löschen). [28]

NoSQL-Datenbanken sind dagegen nicht relational, was bedeutet, dass diese nicht das Konzept der strukturierten Daten mit Tabellen, Feldern und Spalten verwenden. Nicht relationale Datenbanken wurden mit Blick auf die Cloud entwickelt und eignen sich daher hervorragend für die horizontale Skalierung [28]. Dadurch sind sie also deutlich flexibler als relationale Datenbanken [20]. Es gibt verschiedene Typen von nicht relationalen Datenbanken, die die Daten auf unterschiedliche Weise speichern [28]:

- **Dokumentendatenbanken**: Speichern Daten in Dokumenten, bei denen es sich normalerweise um JSON-ähnliche Strukturen handelt (Unterstützen folgende Datentypen: Zeichenfolgen; Zahlen wie int, float und long; Date-Time-Objekte; Arrays; Objeke und sogar verschachtelte Dokumente)

- **Schlüsselwertdatenbanken**: Speichern eine Information in zwei Teilen: Schlüssel und Wert; der Schlüssel wird dann verwendet, um die Informationen aus der Datenbank abzurufen
- **Graphdatenbanken**: Verwenden eine Struktur aus Elementen: sogenannten Knoten, in denen die Daten gespeichert werden, und Kanten, welche Attribute über die Beziehung der beiden Knoten, die sie verbinden, besitzen
- **Spaltenorientierte Datenbanken**: Speichern Daten ähnlich wie relationale Datenbanken in Tabellen, Spalten und Zeilen, jedoch müssen die Namen und Formatierungen der Spalten nicht in jeder Zeile übereinstimmen; gelten als zweidimensionale Schlüsselwertspeicher, da sie eine mehrdimensionale Zuordnung verwenden, um Daten nach Zeile und Spalte zu referenzieren

Neben NoSQL-Datenbanken wie DynamoDB bietet AWS auch relationale Datenbanksysteme wie Amazon RDS an. Wirft man einen Blick auf Tabelle 4.2, wird jedoch schnell deutlich, welche Vorteile eine nicht relationale Datenbank im Vergleich zu einem relationalen Datenbankmanagementsystem besitzt: Sie bietet eine höhere Leistung, mehr Datenspeicher und eine bessere Skalierung, und durch flexible Schemata kann sie mehr Daten verschiedener Typen speichern, die ohne größere Schemaänderungen geändert werden können.

Kategorie	**Nicht relational**	**Relational**
Verfügbarkeit	Hoch	Hoch
Horizontale Skalierung	Hoch	Niedrig
Vertikale Skalierung	Hoch	Hoch
Datenspeicher	Optimiert für große Datenmengen	Mittlere bis große Datenmengen
Leistung	Hoch	Niedrig bis mittel
Zuverlässigkeit	Mittel	Hoch
Komplexität	Niedrig	Mittel (Verknüpfungen)
Flexibilität	Hoch	Niedrig (strenges Schema)

Tabelle 4.2: Nicht relationale vs. Relationale Datenbanken [28]

4.4.2 NoSQL: DynamoDB vs. MongoDB

Wenn es um NoSQL-Datenbankdienste geht, sind MongoDB und DynamoDB zwei der beliebtesten Lösungen [35]. Sie unterscheiden sich dabei vor allem in der Herangehensweise, Daten zu speichern und zu verwalten [20].

MongoDB gehört zu den dokumentenbasierten Datenbanken, das heißt, die Grundlage der Speicherung von Informationen in dieser Datenbank basiert auf Dokumenten, die im *BSON*-Format vorliegen [49, S. 444]. BSON steht für „Binary JSON“ und bedeutet,

dass Typ- und Längeninformationen als Binärstruktur kodiert werden, wodurch diese im Vergleich zum JSON-Format viel schneller durchlaufen werden kann [27].

MongoDB eignet sich aufgrund seiner guten Performance auch für größere Applikationen und bietet zudem einige weitere Features wie Clustering und Sharding, welche gerade bei sehr großen Datenmengen entscheidende Vorteile bringen. MongoDB ist für die verschiedensten Betriebssysteme wie *Linux*, *Windows*, *Solaris* und *Mac OS X* verfügbar und kann auf allen Systemen installiert werden, auf denen auch *Node.js* lauffähig ist. Für Abfragen nutzt MongoDB die *MongoDB Query Language (MQL)*. [49]

DynamoDB ist dagegen in erster Linie ein Schlüsselwert-Speicher, da sein Datenmodell aus Key-Value-Paaren in einer schemalosen, nicht relationalen Tabelle von Zeilen (Datensätzen) besteht [15]. Während DynamoDB nicht speziell für Dokumentenspeicherung entworfen ist, kann der Dienst jedoch dennoch mit JSON-Daten umgehen, indem er sie in einem speziellen Attributtyp, nämlich dem Typ `Map` speichert. Dies ermöglicht es, auch komplexe Datenstrukturen wie Dokumente in DynamoDB zu speichern [1].

Tabelle 4.3 fasst die wichtigsten Eigenschaften beider Datenbanksysteme, welche von Ionos [20] genannt werden, zusammen und stellt somit DynamoDB und MongoDB anhand ausgewählter Kriterien direkt gegenüber.

Kategorie	**DynamoDB**	**MongoDB**
Skalierbarkeit	Unendliche Skalierung innerhalb von AWS	Horizontale Skalierung durch Sharding (automatisches Aufteilen großer Datensätze und Verteilen der Last auf mehrere Server)
Funktionsweise	Schlüsselwertspeicher	Dokumentenorientiert, BSON-Format
Infrastruktur	Teil des AWS-Ökosystems, komplett verwaltet	Open-Source, selbst zu konfigurieren und zu warten
Programmiersprachen	Java, JavaScript, Perl, PHP, Python, Ruby, .net und einige andere	Fast alle
Sicherheit	Teil des IAM-Modells von AWS, gut geschützt	Selbstständige Implementierung von Sicherheitsfeatures

Tabelle 4.3: DynamoDB vs. MongoDB

Im Endeffekt hängt jedoch auch die Entscheidung, ob DynamoDB, MongoDB, oder vielleicht sogar eine ganz andere NoSQL-Datenbank, von den spezifischen Anforderungen des Projekts und den Präferenzen des Entwicklers ab. Eine falsche Entscheidung kann in den meisten Fällen nicht getroffen werden, höchstens eine gute oder eine weniger gute. Während MongoDB für manche Projekte die bessere Entscheidung sein

mag, ist DynamoDB stets eine gute Wahl, wenn im Projekt bereits Amazon-Dienste genutzt werden, was in dieser Arbeit der Fall ist, denn die Bereitstellung, Wartung und Integration in die bestehende Infrastruktur werden dadurch einfacher [35].

4.5 Amazon EventBridge Scheduler

Es war bereits die Rede davon, dass eine Slack-App stark Event-getrieben ist, da sie bestimmte Aktionen zu bestimmten Zeitpunkten erfordert. Aus diesem Grund ist Amazon EventBridge ein nützliches Tool bei der Entwicklung von Slack-Apps. Der Scheduler ist eine relativ neue Funktion von Amazon EventBridge, mit welcher Aufgaben und Events geplant werden können und die auf der Grundlage von Parametern ausgelöst wird, die der AWS-Nutzer definiert [2]. Dies funktioniert wie folgt: Zunächst definiert der Entwickler Regeln, die festlegen, wann bestimmte Aktionen ausgeführt werden sollen, also beispielsweise, wann die AWS-Lambda-Funktion getriggert werden soll. Diese Regeln können basierend auf Zeitplänen, Ereignissen oder benutzerdefinierten Auslösern erstellt werden. Danach wird ein Ziel definiert. Das Ziel gibt an, was genau bei Auslösung der Regel geschehen soll. Soll das Ziel eine AWS-Lambda-Funktion sein, so muss der Zieltyp `LAMBDA_Invoke` lauten. Ziel können aber auch andere AWS-Services oder sogar externe Ziele wie HTTP-Endpunkte sein. Sobald die Regel konfiguriert und das Ziel festgelegt ist, überwacht der EventBridge Scheduler kontinuierlich die Zeit und Ereignisse, um zu bestimmen, wann die Regel ausgelöst werden soll. Sind die festgelegten Bedingungen erfüllt, wird das definierte Ziel aktiviert, zum Beispiel wird die angegebene Lambda-Funktion gestartet und die Aktion durchgeführt, die in dieser definiert ist.

Im Zusammenhang mit einem Slack-Bot, der beispielsweise jeden Morgen um 8:00 Uhr eine Nachricht senden soll, könnte der EventBridge Scheduler so genutzt werden: Der Entwickler schreibt zunächst eine AWS-Lambda-Funktion, die eine Slack-Request, zum Beispiel zum Senden einer Nachricht in einen Channel, an die Webhook der Slack-App sendet. Hierfür kann die Webhook auf der Konfigurationsseite der Slack-App entnommen und kopiert werden. Nachfolgend kann der Entwickler einen neuen Zeitplan unter Amazon EventBridge erstellen, in dem er festlegt, dass jeden Morgen um 8:00 Uhr eine Aktion ausgelöst werden soll. Als Aktion, bzw. als Ziel kann er dann die zuvor erstellte Lambda-Funktion angeben. (Abbildung 4.2)

4.6 Amazon S3

Für das Homeoffice-Sport-Feature der App wird eine zuverlässige Lösung für das Hosting der Sportübungs-Videos benötigt. Hierfür bietet sich ein S3-Bucket von AWS optimal an. *Amazon Simple Storage Service (Amazon S3)* ist ein Speicherdienst, der es ermöglicht, große Mengen an Daten sicher und skalierbar zu speichern. Egal ob es sich um Daten für Analysezwecke, interne Anwendungen oder mobile Apps handelt, Amazon S3 bietet verschiedene Speicheroptionen zu günstigen Preisen. Außerdem ermöglicht S3 eine einfache Verwaltung von Daten und erlaubt es, Zugriffsrechte anzu-

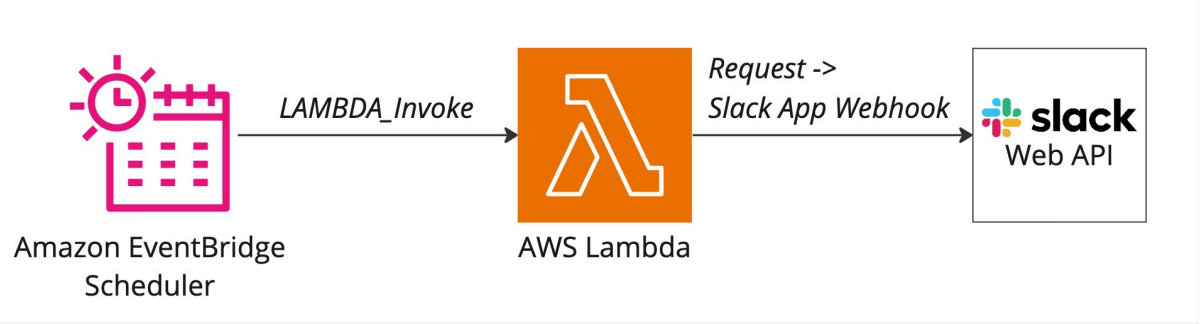

Abbildung 4.2: Zeitgesteuerte Slack Requests mit Amazon EventBridge Scheduler und AWS Lambda (Quelle: Eigene Darstellung)

passen, um den Anforderungen von Unternehmen, Organisationen und gesetzlichen Vorschriften gerecht zu werden. [3]

Nachdem ein Bucket erstellt wurde, und die entsprechenden Berechtigungen festgelegt wurden, können beliebige Dateien in den Bucket hochgeladen werden. Es kann auch eine Ordnerstruktur im Bucket angelegt werden, die im späteren Projektverlauf genutzt werden kann, um die Videos in verschiedene Gruppen zu unterteilen, wobei eine Gruppe für ein bestimmtes Bedürfnis (zum Beispiel „Nacken und Schultern lockern“) steht. Jede einzelne Datei eines S3-Buckets besitzt eine Objekt-URL, über die auf die Datei zugegriffen werden kann. Der externe Zugriff funktioniert dabei nur, wenn dies in den Richtlinien entsprechend festgelegt ist. Eine Bucket-Richtlinie wird in JSON geschrieben und beinhaltet:

- `Effect`: Definiert, ob die Richtlinie den Zugriff gewährt (`Allow`) oder verweigert (`Deny`)
- `Principal`: Die Identitäten, die auf den Bucket zugreifen dürfen
- `Action`: Die erlaubten Aktionen, die auf dem Bucket und dessen Objekten ausgeführt werden dürfen, wie `GetObject`, `PutObject`, `DeleteObject` usw.
- `Resource`: Die Ressourcen im Bucket, auf die die Aktionen angewendet werden können; dies können einzelne Objekte oder der gesamte Bucket sein
- `Condition` (optional): Zusätzliche Bedingungen, die erfüllt sein müssen, damit die Richtlinie greift

4.7 Zusätzlich benötigte Instanzen

In der Entwicklungsphase einer serverlosen Bolt-Anwendung kommen neben den bereits erwähnten AWS-Services zusätzliche Hilfsmittel zum Einsatz. Auch wenn diese nicht unmittelbar im Fokus stehen, sind sie dennoch essentiell und werden daher im Folgenden kurz erläutert.

4.7.1 IAM

Ohne IAM kommen Nutzer von AWS in der Regel nicht sehr weit. Deswegen sollten oder müssen hier relativ zu Beginn einige Einstellungen vorgenommen werden. IAM steht für *Identity and Access Management* und ist Amazons Service für das Management von Authentifizierung (Wer ist der Nutzer?) und Autorisierung (Welche Rechte besitzt der Nutzer?). Die vier Grundelemente von IAM sind Richtlinien („Policys“), Benutzer, Gruppen und Rollen. [51, S. 198-199]

Daniel Stender beschreibt diese wie folgt:

- **Richtlinien** beinhalten konkrete Zugriffsberechtigungen und legen somit zum Beispiel fest, auf welche Instanzen man zugreifen darf, und was man dort genau tun darf; sie können einem Benutzer oder einer Rolle zugewiesen werden
- **Benutzer** sind die Nutzer der unterschiedlichen AWS-Dienste; ihnen können Richtlinien zugeteilt werden; außerdem können sie einer oder mehreren Gruppen hinzugefügt werden
- **Gruppen** sind Benutzergruppen, die Benutzer zusammenfassen, für die dieselben Einstellungen gelten sollen
- **Rollen** enthalten eine oder mehrere Richtlinien; sie können einer AWS-Instanz zugeteilt werden, was bedeutet, dass diese Instanz die Zugriffsrechte bekommt, die in den der Rolle zugewiesenen Policys festgelegt worden sind

4.7.2 AWS CLI

Um ECR vom eigenen Arbeitsrechner aus anzusprechen bzw. um ein neues Docker-Image in die Container Registry von Amazon pushen zu können, wird das offizielle Kommandozeilen-Werkzeug *aws-cli* (*AWS Command Line Interface*) benötigt. Das Tool aws-cli ist ein Python-Programm und kann daher einfach mit `pip3` aus dem Python Package Index installiert werden. Der Befehl dafür lautet ganz einfach `pip3 install awscli`. Der Kommandozeitenbefehl von aws-cli ist schlicht `aws`. [51, S. 202-204]

Zur Inbetriebnahme von `aws` muss jedoch noch eine Initialisierung durchgeführt werden, denn der Client weiß zwar, wo sich die AWS-APIs befinden, doch er benötigt noch Zugangsdaten, um mit dem jeweiligen Benutzer auf dessen Konto zugreifen zu können. Dafür muss der Befehl `aws configure` ausgeführt werden, woraufhin die Credentials, nämlich der geheime Zugriffsschlüssel und die zugehörige ID des Users, verlangt werden. Außerdem können hier die Default-Region sowie das Format, in dem die Rückgaben von Befehlen ausgegeben werden sollen, angegeben werden. [51, S. 206-207]

4.7.3 Amazon CloudWatch

Amazon CloudWatch ist der Monitoring-Service von AWS und überwacht Ressourcen und Anwendungen in Echtzeit. Die Startseite zeigt automatisch Metriken für jeden genutzten AWS-Service an, um diese zu verfolgen. Mit den erfassten Metriken können

benutzerdefinierte Dashboards erstellt werden, um einen besseren Überblick über eigene Anwendungen zu haben. Außerdem gibt es die Möglichkeit, durch Alarme Metriken zu überwachen und Benachrichtigungen zu senden oder automatisch Änderungen an den überwachten Ressourcen vorzunehmen, wenn Schwellenwerte überschritten werden. [4]

In diesem Projekt wird CloudWatch jedoch hauptsächlich zum Debugging, genauer gesagt zur Protokollüberwachung, verwendet. Indem CloudWatch-Logs genutzt werden, können Protokolldateien von Anwendungen und Ressourcen überwacht werden. Durch das Analysieren der Protokolle können Fehler leichter erkannt und dadurch behoben werden.

4.8 Architekturüberblick und APIs

Um die Zusammenhänge etwas deutlicher zu machen, und auch während des Projekts als Entwickler selbst nicht aus den Augen zu verlieren, bietet es sich an, eine Architekturüberblick-Grafik zu erstellen. Der Überblick in Abbildung 4.3 soll zeigen, wie die für dieses Projekt verwendeten Komponenten zusammenhängen und miteinander interagieren. Interaktionen sind durch Pfeile dargestellt. Interessant sind hierbei besonders die Interaktionen mit der Events API und der Web API von Slack, daher wird im Folgenden kurz ihre Bedeutung wiederholt:

- Slack-User löst ein Event in der beecaring Slack-App aus, zum Beispiel durch einen Button-Klick
- Slack detektiert dieses Event und sendet über **Events API** Benachrichtigung an den Endpunkt der App (API Gateway)
- Bolt-basierte AWS-Lambda-Funktion verarbeitet das Event, entscheidet, welche Aktionen darauf folgen (beispielsweise DynamoDB Datenbank-Abfragen oder der Zugriff auf einen S3-Bucket) und wie die zugehörige Antwortnachricht aussehen soll
- Bolt-Anwendung sendet als AWS-Lambda-Funktion eine HTTP-Request an passenden Slack **Web API** Endpunkt, um gewünschte Aktion durchzuführen
- Slack verarbeitet Request und führt Aktion aus, zum Beispiel Nachricht in Channel senden

Doch nicht nur durch ein Slack-Event kann eine AWS-Lambda-Funktion ausgelöst werden, sondern auch durch ein in Amazon EventBridge festgelegtes zeitliches Event. Der Übersichtlichkeit halber wird in Abbildung 4.3 nur eine Lambda-Komponente dargestellt, und der Amazon EventBridge Scheduler verweist auf diese. Jedoch handelt es sich bei der AWS-Lambda-Funktion, die durch den EventBridge Scheduler getriggert wird, um eine eigene Funktion, die nur diejenigen Aufgaben enthält, die um die festgelegte Zeit ausgeführt werden sollen. Diesen Teil der Architektur sollte man sich also entkoppelt von der AWS-Lambda-Funktion, welche das in Bolt geschriebene Hauptprogramm enthält, und den damit verbundenen Komponenten vorstellen, so

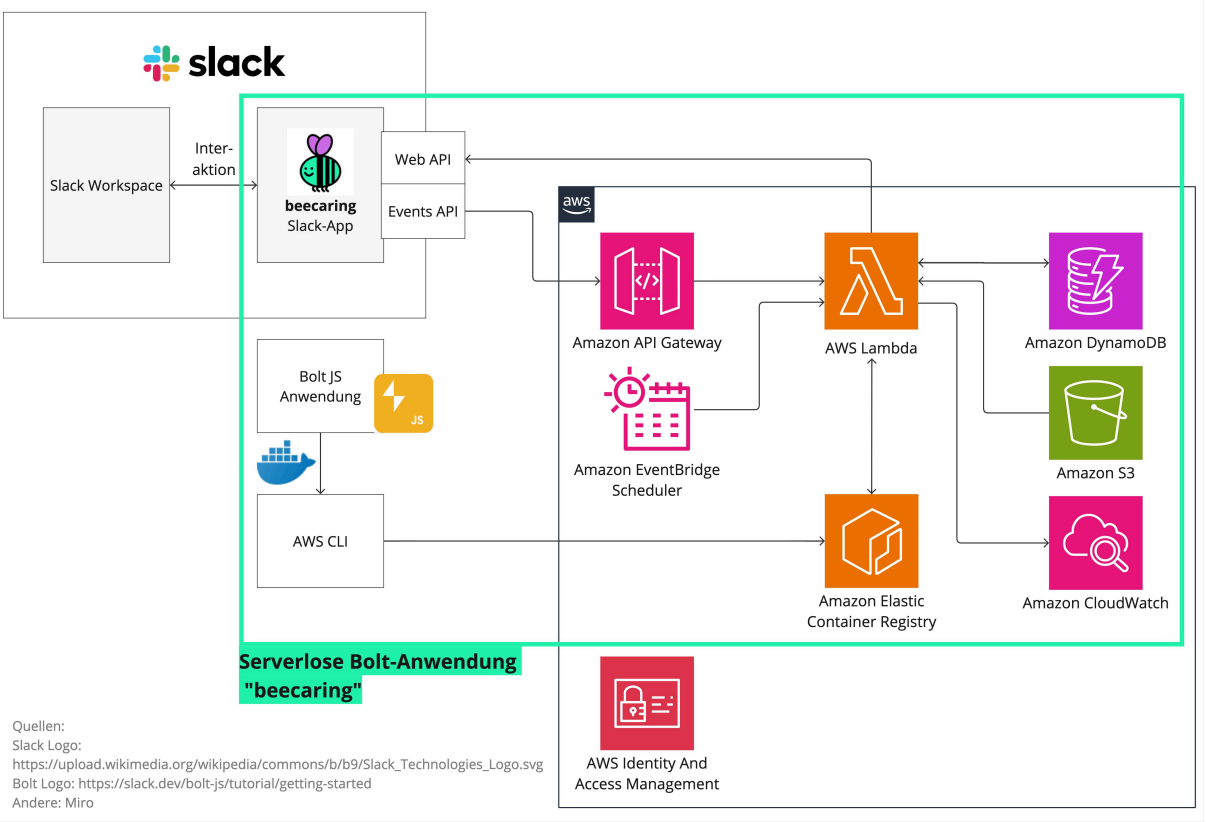

Abbildung 4.3: Architekturüberblick der serverlosen Bolt-Anwendung beecaring (Quelle: Eigene Darstellung)

wie bereits in Abbildung 4.2 gezeigt wurde. Die AWS-Lambda-Funktion, die durch EventBridge ausgelöst wird, muss nicht unbedingt auch in Bolt geschrieben werden, wenn die gewünschte Funktionalität es nicht erfordert. Mit der Web API von Slack kann die Lambda-Funktion über Webhooks auch kommunizieren, wenn es sich nicht um eine Bolt-Funktion handelt.

Wie die Bolt-Funktion, die das Hauptprogramm enthält, mithilfe von Docker und AWS CLI in die Amazon Elastic Container Registry gepusht und anschließend als AWS-Lambda-Funktion ausgewählt werden kann, wurde bereits in Kapitel 4.3 genauer beschrieben. Auch auf die einzelnen AWS-Komponenten wurde bereits ausführlich eingegangen.

Ein wichtiger Punkt ist an dieser Stelle aber noch: Der grüne Rahmen in Abbildung 4.3 umfasst die gesamte Logik der serverlosen Bolt-Anwendung beecaring. Die Logik für die beecaring Slack-App steckt als Bolt JS Code in der AWS-Lambda-Funktion. Für User, also nach außen hin sichtbar, ist die Anwendung jedoch lediglich als Slack-App, die den Namen *beecaring* trägt. Vergleicht man nun diese Abbildung mit Abbildung 2.1, so ist die in Abbildung 2.1 genannte *Anwendung* nun Slack, *User* steht für einen Slack-User, der *Bot* ist nun die beecaring Slack-App und der *Dienst* ist in diesem Fall die AWS-Lambda-Funktion. Man muss also verschiedene Bedeutungen von „App“ oder „Anwendung“ differenzieren.

Kapitel 5

Technische Umsetzung

Nach dem Abschluss des Design-Prozesses sowie des Architekturentwurfs kann mit der technischen Umsetzung begonnen werden. Das heißt, die serverlose Bolt-Anwendung beecaring kann nun implementiert werden. Dies umfasst nicht nur das Schreiben von JavaScript-Code mit dem Framework Bolt um beispielsweise festzulegen, wie auf bestimmte Events reagiert werden soll, sondern vor allem auch das Aufsetzen der AWS Cloud Infrastruktur und ganz zu Beginn die Initialisierung der App seitens Slack. In diesem Kapitel werden die wichtigsten Umsetzungsschritte dargelegt sowie ausgewählte Teile des Codes, der für die Realisierung der Anwendung nötig ist, erklärt.

5.1 App Initialisierung

Nach der Durchführung der in diesem Abschnitt beschriebenen Schritte sollte die Slack-App beecaring bereits in einem Workspace installiert sein. Die Initialisierung der App umfasst im Wesentlichen zwei Phasen: die Konfiguration der App in der Slack-Plattform und das Aufsetzen der Bolt JS Anwendung.

5.1.1 Konfiguration der Slack-App *beecaring* unter api.slack.com

Der erste Schritt bei der Entwicklung einer Slack-App ist die Konfiguration der App unter `api.slack.com`. Um eine neue App zu erstellen, müssen zu allererst ein App Name und der Workspace, in dem die App entwickelt werden soll, angegeben werden. Es ist dabei vorteilhaft, die App zu Beginn noch nicht in dem tatsächlichen Workspace (*codecentric AG*) zu erstellen, sondern einen neuen Workspace (beispielsweise *test-workspace*) für Testzwecke zu erstellen. Des Weiteren können unter `Basic Information` eine App-Beschreibung und ein App-Icon hinzugefügt werden. Dadurch erhält die App eine Identität und ein professionelleres Erscheinungsbild. Als Icon für beecaring wurde das Logo ausgewählt, das bereits in Kapitel 3.6 für das Corporate Design entworfen wurde (siehe auch Anhang C). Wurde die App initialisiert, ist sie allerdings noch nicht im Workspace installiert. Dafür muss der App zu Beginn mindestens eine Berechtigung erteilt werden. Unter `OAuth & Permissions` können diese Berechtigungen in Form von

Scopes erteilt werden. Um mit einem einfachen Beispiel zu starten, kann zum Beispiel der Scope `chat:write` gesetzt werden, welcher der App erlaubt, Nachrichten in einem Chat zu senden. Danach kann die App im Workspace installiert werden. Weitere Scopes können jederzeit während der Entwicklung hinzugefügt werden, die App muss dann lediglich durch einen Button-Klick neu im Workspace installiert werden.

Bei der Installation der Slack-App wird ein sogenannter *Bot User OAuth Token* generiert. Mit diesem kann sich die App später bei der Slack-API anmelden und Aktionen im Namen des Users durchführen, wie zum Beispiel Nachrichten senden oder lesen. Neben dem OAuth Token wird später noch eine weitere wichtige Zeichenfolge benötigt: Das sogenannte *Signing Secret* der App, zu finden unter `App Credentials`. Dieses dient später zur Verifikation, das heißt zur Überprüfung, ob eine bestimmte Anfrage tatsächlich von Slack kommt. Sowohl der OAuth Token als auch das Signing Secret sollten sicher aufbewahrt werden. [42]

5.1.2 Initialisierung der Bolt JS Anwendung

Bevor mit der eigentlichen Entwicklung begonnen werden kann, muss zuerst die nötige Umgebung aufgesetzt werden. Wie bereits erwähnt, kann das Framework Bolt in den Sprachen JavaScript, Python und Java verwendet werden. Da in diesem Projekt Bolt für JavaScript verwendet wird, welches ein Node-basiertes Framework mit TypeScript Bindings ist, muss dafür gesorgt werden, dass Node.js Anwendungen und Pakete auf dem Rechner laufen. Node.js ist eine serverseitige JavaScript-Laufzeitumgebung, die auf der V8 JavaScript Engine von Chrome aufbaut. Durch diese Architektur kann JavaScript außerhalb der üblichen Browserumgebung ausgeführt werden, was bedeutet, dass JavaScript nicht nur im Frontend, sondern auch im Backend eingesetzt werden kann [8, S. 6]. Die Plattform basiert auf einer Sammlung von etablierten Bibliotheken, die zusammengefasst eine sehr flexible Arbeitsumgebung schaffen. Der Kern der Plattform bietet lediglich einen Satz an Grundfunktionalität, für alle weiteren Anforderungen gibt es jedoch den *Node Package Manager (NPM)*, über den verschiedenste Pakete - wie auch Bolt - in die Anwendung eingebunden werden können [49, S. 45]. Node wird von einer einzigen Open-Source-Stiftung mit Hunderten von freiwilligen Entwicklern gepflegt.

Node.js kann unter der offiziellen Node Website, `https://nodejs.org/`, heruntergeladen und anschließend installiert werden. NPM wird durch die Installation von Node automatisch mitinstalliert und kann daher ab diesem Zeitpunkt genutzt werden. Ein neues Node Projekt kann dann mit dem Befehl `npm init` in einem beliebigen Ordner erstellt werden. Die `package.json` Datei, die dabei automatisch angelegt wurde, enthält die Liste aller Abhängigkeiten und deren Versionen. Dies erlaubt eine einfache Installation eines Projekts via NPM [8, S. 8]. Mithilfe von NPM kann nun auch Bolt im Projekt installiert und als Abhängigkeit in die `package.json` Datei hinzugefügt werden: `npm install @slack/bolt`.

Jetzt werden das Signing Secret und der Slack Bot Token benötigt, welche unter `api.slack.com` der jeweiligen App entnommen werden können. Diese sollten am besten als Umgebungsvariablen in einer `.env` Datei gespeichert werden. Darauf folgend

wird üblicherweise eine `app.js` oder `app.mjs` Datei im Projektordner angelegt, und der folgende Code hinzugefügt [42]:

```
const { App } = require('@slack/bolt');

// Initialisieren der Bolt-App mit Bot Token und Signing Secret
const app = new App({
  token: process.env.SLACK_BOT_TOKEN,
  signingSecret: process.env.SLACK_SIGNING_SECRET
});

(async () => {
  // Starten der Bolt-App
  await app.start(process.env.PORT || 3000);

  console.log('beecaring Bolt app is running!');
})();
```

Listing 5.1: Initialisieren einer Bolt JS Anwendung

Nach dem Speichern der `app.js` Datei und dem Laufenlassen des Befehls `node app.js` sollte die App eine Meldung auf der Konsole ausgeben, dass sie läuft. Allerdings läuft die App damit vorerst lokal auf dem eigenen Rechner. Um sie global zu hosten, muss zuerst die Cloud Infrastruktur aufgesetzt werden. Bevor damit begonnen wird, muss jedoch noch im Projektordner des Bolt-Programms ein Dockerfile gemäß Listing 5.2 erstellt werden. In diesem Fall wird dabei als Base Image das offizielle AWS Lambda Node.js 18 Base Image von Amazon ECR benutzt. Durch `COPY`-Befehle werden die benötigten Dateien und Ordner in das `/var/task/` Verzeichnis kopiert, welches das Arbeitsverzeichnis für AWS-Lambda-Funktionen innerhalb des Containers darstellt.

```
# Verwenden des offiziellen AWS Lambda Node.js 18
# Base Images von Amazon ECR
FROM public.ecr.aws/lambda/nodejs:18

# Kopieren der erforderlichen Dateien und Abhaengigkeiten
# in das Verzeichnis der AWS-Lambda-Funktion
COPY *.mjs *.js package*.json /var/task/
COPY events/*.js /var/task/events/
COPY views/*.json /var/task/views/
COPY functions/*.js /var/task/functions/

# Installieren der Produktionsabhaengigkeiten mit 'npm ci'
RUN npm ci --production

# Festlegen des Standardbefehls, um
# den Lambda-Funktionshandler auszufuehren
CMD [ "app.handler" ]
```

Listing 5.2: Dockerfile der Bolt JS Anwendung

5.2 AWS Cloud Infrastruktur aufsetzen

Dieser Abschnitt behandelt die Umsetzung des in Kapitel 4 entworfenen Cloud-Deployments. Dies umfasst das Hosting der Anwendung mittels AWS Lambda, die Verknüpfung von Lambda mit der Slack-App über einen API Gateway, das Erstellen einer DynamoDB-Tabelle und das Einrichten eines S3 Buckets für den Dateiupload von Übungsvideos. Darüber hinaus wird erläutert, wie Slack-Anfragen verifiziert werden und wie die Implementierung dieser Verifikation vereinfacht werden kann.

5.2.1 Hosting

Um die Anwendung nicht nur lokal, sondern auch in der Cloud bereitzustellen, sind mehrere Schritte erforderlich. Dazu gehört die Einrichtung einer geeigneten Hosting-Umgebung. Die nachfolgenden Schritte orientieren sich an dem Verfahren von Anton Putra [30].

Nachdem ein neuer IAM User erstellt wurde, `aws-sdk` installiert und das AWS Profil über das AWS CLI mithilfe des Befehls `aws configure` konfiguriert wurde, wird zunächst eine neue IAM Rolle erstellt. Dieser wird eine Richtlinie zugeordnet, welche der Rolle die folgenden Aktionen erlaubt:

- Das Erstellen von Log-Gruppen in CloudWatch Logs der eigenen Region
- Das Erstellen von Log-Streams und das Hinzufügen von Log-Ereignissen in einer spezifischen Log-Gruppe
- Alle Aktionen auf einer DynamoDB-Tabelle in der eigenen Region

Danach muss ein neues ECR-Repository (`beecaring`) auf AWS angelegt werden. Die darauffolgenden Schritte lassen sich entnehmen, indem unter dem in ECR angelegten Repository der Button `Push-Befehle anzeigen` geklickt wird. Daraufhin werden nämlich bereits Kommandos angezeigt, die einfach kopiert und in das Terminal des Codeeditors eingefügt werden können. Dies führt letztendlich dazu, dass ein neues Image gebaut und in das ECR-Repository gepusht wird.

Nun kann eine neue AWS-Lambda-Funktion erstellt werden. Dabei wird ausgewählt, dass ein bereits existierendes Container-Image für die Funktion bereitgestellt werden soll, anstatt eine neue Funktion ohne Vorgabe zu erstellen. Der Funktion wird in diesem Fall der Name `beecaring` gegeben, und die IAM-Rolle, die im vorherigen Schritt bereits erstellt wurde, zugeteilt. Im Anschluss wird das aktuelle Container-Image aus dem ECR-Repository `beecaring` als Image ausgewählt und die AWS-Lambda-Funktion deployt.

5.2.2 Verknüpfung der Slack-App mit AWS Lambda

Was jetzt noch fehlt, um die Funktion überhaupt triggern zu können, ist ein API Gateway. Hierfür eignet sich eine neue HTTP API, bei welcher lediglich die eben erstellte AWS-Lambda-Funktion `beecaring` als Integration gewählt werden muss. Da

die Slack-App nur POST Requests an die Lambda-App schicken wird, um Events zu senden, kann bei der HTTP-Konfiguration `POST` als Methode ausgewählt werden. Nach Abschluss der Konfiguration stellt API Gateway eine URL als API-Endpunkt bereit. Diese wird als Request URL für Slack benötigt. Slack sendet dann, wenn Events auftreten, HTTP POST Requests an diese URL. Die URL muss daher ebenfalls auf der Konfigurationsseite der Slack-App (`api.slack.com`) unter dem Reiter `Event Subscriptions` angegeben werden, wodurch Slack-Events aktiviert werden.

Allerdings wird nicht jede URL sofort von Slack akzeptiert. Denn, sobald eine URL eingegeben wird, sendet Slack eine Anfrage mit einem Challenge-Parameter, und der API-Endpunkt muss mit dem Challenge-Wert antworten [47]. Um zu verstehen, wieso dies wichtig ist, und wie das Ganze funktioniert, wird im Folgenden kurz auf den Signatur- und Verifikationsvorgang von Slack-Requests eingegangen.

5.2.3 Verifizieren von Slack Requests durch Signing Secret

Geht eine HTTP-Anfrage bei der AWS-Lambda-Funktion ein, so muss zuerst geprüft werden, ob die Anfrage tatsächlich von der Slack-App stammt. Slack signiert seine Anfragen mit einem für die App einzigartigen Geheimnis. Mithilfe signierter Geheimnisse kann die App sicher überprüfen, ob Anfragen von Slack authentisch sind. Dies funktioniert wie folgt: Bei jeder HTTP-Anfrage, die Slack sendet, fügt Slack einen *X-Slack-Signature* HTTP-Header hinzu. Die Signatur wird erstellt, indem der Body der Anfrage (als JSON-Payload) mit der SHA-256-Funktion gehasht und mit einem HMAC-Signaturgeheimnis kombiniert wird. Die resultierende Signatur ist für jede Anfrage eindeutig und enthält keine geheimen Informationen, sodass die Sicherheit der App gewährleistet ist. Für den Validierungsvorgang wird das Signing Secret benötigt, welches unter `api.slack.com > App Credentials` ausgelesen werden kann. Mit diesem kann die Bolt-basierte Lambda-App basierend auf der Anfrage eine eigene Signatur berechnen und anschließend überprüfen, ob die selbst berechnete Signatur mit der Signatur auf der Anfrage übereinstimmt. [41]

Um die Signaturprüfung automatisch durchführen zu lassen, und sich nicht um die Implementierung der Verifikation kümmern zu müssen, kann der `AwsLambdaReceiver` von `@slack/bolt` verwendet werden. Das heißt, der in 5.1.2 gezeigte Code muss entsprechend Listing 5.3 modifiziert werden.

```
1  const { App, AwsLambdaReceiver } = require('@slack/bolt');
2
3  const signingSecret = process.env.SLACK_SIGNING_SECRET;
4  const botToken = process.env.SLACK_BOT_TOKEN;
5
6  /* Initialisieren des benutzerdefinierten Receivers mit Signing
    Secret */
7  const awsLambdaReceiver = new AwsLambdaReceiver({
8    signingSecret: signingSecret
9  });
10
```

```
11  /* Initialisieren der Bolt-App mit Bot Token und AWS Lambda
     Receiver */
12  const app = new App({
13    token: botToken,
14    receiver: awsLambdaReceiver,
15    logLevel: LogLevel.DEBUG
16  });
17
18  /* Asynchroner Start des AWS Lambda Receivers, Verarbeiten des
     eingehenden Ereignisses damit und Werfen eines Fehlers, wenn
     Validierung der Slack-Nachricht nicht erfolgreich */
19  export async function handler(event, context, callback) {
20    const handler = await awsLambdaReceiver.start();
21    console.log('beecaring Lambda app is running!');
22    const response = await handler(event, context, callback);
23
24    if (response.statusCode != 200) {
25      throw new Error('Failed to validate slack message');
26    }
27
28    return response;
29  }
```

Listing 5.3: Erstellen eines benutzerdefinierten Receivers mit `AwsLambdaReceiver`

5.2.4 DynamoDB Datenbank

Eine wichtige Komponente der beecaring Cloud-Infrastruktur fehlt bisher noch: DynamoDB zum persistenten Speichern der Nutzerdaten. Ein Datensatz der Anwendung beecaring soll gemäß Abbildung 5.1 aufgebaut sein. Dabei ist `userid` die User-ID von Slack (String), `officedays` sind die vom User ausgewählten Bürowochentage (Array aus Strings wie „monday“, „friday“), `needs` die ausgewählten Bedürfnisse (Array aus Strings wie „lower_body“, „neck_and_shoulders“) und `num_exercises` ist die gewählte Anzahl der Übungen pro Tag (String, der die Werte „never“, „once“, „twice“, „three_times“ oder „four_times“ annehmen kann). Ein Array wird in DynamoDB durch den Typ „L“ für „List“ dargestellt, während die Abkürzung „S“ einen String symbolisiert.

Das Erstellen einer neuen DynamoDB Tabelle erfolgt unter `AWS Konsole > DynamoDB > Tabellen > Tabelle erstellen`. Neben einigen Einstellungen, die den Default-Wert behalten können, müssen ein Tabellenname (in diesem Fall `beecaring`) und ein Partitionsschlüssel angegeben werden. Der Partitionsschlüssel ist eine Komponente des Primärschlüssels der Tabelle. Es handelt sich dabei um einen Hash-Wert, der verwendet wird, um Daten aus der Tabelle abzurufen und diese den verschiedenen Hosts zuzuweisen. Dies dient dazu, die Skalierbarkeit und Verfügbarkeit der Datenbank zu gewährleisten [1]. Als Partitionsschlüssel wurde in diesem Fall `userid` gewählt, da ein Datensatz zu einem User gehören soll und die User-IDs von Slack, die nachher in diesen Feldern gespeichert werden, ebenfalls einzigartig sind.

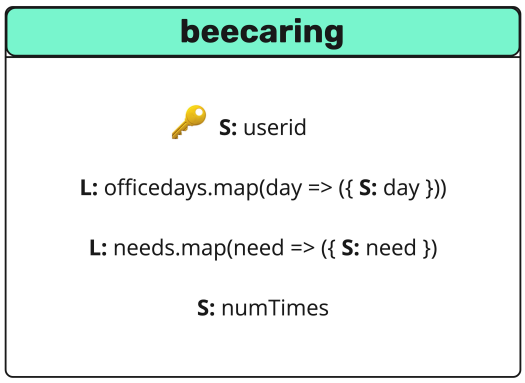

Abbildung 5.1: Modellierung der DynamoDB Tabelle (Quelle: Eigene Darstellung)

Nun kann in den Code-Editor gewechselt werden. Hier wird eine Datei mit dem Namen `dynamodb.js` angelegt, in der festgelegt wird, um welche DynamoDB-Instanz es sich handelt. Dafür muss `aws-sdk` importiert, und anschließend die Region, in welcher die Tabelle erstellt wurde, angegeben werden. Dann wird eine neue DynamoDB-Instanz für die aktuellste API-Version angelegt. In einer weiteren Datei, `db.js`, müssen dann noch die wichtigsten Datenbank-Operationen implementiert werden. Dafür wird zuerst die eben angelegte DynamoDB-Instanz importiert. Die DynamoDB-Instanz muss nicht zwingend in einer eigenen Datei ausgelagert sein, doch das Auslagern hat den Vorteil, dass sich die DynamoDB-Instanz beim Testen des Codes von `db.js` mit dem Testframework Jest leichter mocken lässt, wenn sie als separate Einheit importiert wird. Nun können die folgenden Methoden implementiert werden, welche für verschiedene Operationen auf der Datenbank zuständig sind:

- `getItem`
  - Zuständig für das Abrufen eines bestimmten Datensatzes anhand des Primärschlüssels `userId`
  - Führt die AWS DynamoDB Methode `getItem()` auf der DynamoDB-Tabelle aus
  - Erhält `userId` als Eingabe und gibt die für diesen User gespeicherten Attributwerte zurück
- `saveItem`
  - Zuständig für die Speicherung neuer Datensätze oder das Updaten bereits existierender Datensätze
  - Führt nacheinander die AWS DynamoDB Methoden `getItem()`, `updateItem()` und `putItem()` auf der DynamoDB-Tabelle aus
  - Erhält alle Attributwerte als Eingabe, erstellt einen dynamischen Update-Ausdruck und sendet den geänderten Datensatz zurück an DynamoDB
- `scanItems`

  - Zuständig für das ungefilterte Abrufen aller Datensätze der Tabelle
  - Führt die AWS DynamoDB Methode `scan()` auf der DynamoDB-Tabelle aus
  - Erhält den Tabellennamen `beecaring` als Eingabe und gibt alle in dieser Tabelle gespeicherten Items zurück

- `removeAttribute`
  - Zuständig für das Löschen eines bestimmten Attributwertes eines bestimmten User-Datensatzes
  - Führt die AWS DynamoDB Methode `updateItem()` auf der DynamoDB-Tabelle aus
  - Erhält `userId` und das zu löschende Attribut als Eingabe und führt ein Update mit dem Ausdruck `REMOVE` auf dem User-Datensatz aus

5.2.5 Dateiupload der Übungsvideos in Amazon S3 Bucket

Wie im Kapitel „Serverless Cloud-Deployment“ bereits erwähnt, eignet sich ein Amazon S3 Bucket hervorragend, um die Videos, die für die Homeoffice-Sportübungen benötigt werden, zu speichern und Slack-Nutzern über die Objekt-URLs der einzelnen Videos zur Verfügung zu stellen. Dafür muss ein neuer Bucket erstellt werden: `AWS Konsole > S3 > Bucket erstellen`. Beim Anlegen eines neuen Buckets werden der Bucket-Typ (in diesem Fall: „Allzweck“), der Bucket-Name („beecaring-videos“) und das Vornehmen einiger Sicherheitseinstellungen verlangt. In diesem Fall ist es wichtig, dass der gesamte öffentliche Zugriff *nicht* blockiert wird, denn sonst wäre der Zugriff auf die Videos aus Slack heraus nicht für alle Nutzer möglich. Daher muss zudem eine Bucket-Richtlinie hinzugefügt werden, die im JSON-Format den gewünschten Zugriff erlaubt (siehe Listing 5.4).

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::beecaring-videos/*"
  }
  ]
}
```

Listing 5.4: Bucket-Richtlinie

Wurde der Bucket erstellt, können innerhalb des Buckets anschließend einzelne Ordner zur Kategorisierung der Übungsvideos für die unterschiedlichen Bedürfnisse angelegt werden. Also zum Beispiel ein Ordner `back`, der Videos zu Rückenübungen beinhaltet, ein Ordner `lower_body`, der Videos zum Bewegen von Beinen und Hüfte

beinhaltet usw. Anschließend können die Videos in die passenden Ordner hochgeladen werden. Da das Erstellen der Übungsvideos nicht in den Rahmen dieser Bachelorarbeit fällt, werden vorerst gelabelte Dummy-Videos verwendet, die jedoch zu einem späteren Zeitpunkt problemlos ausgetauscht werden können.

5.3 Events und Actions

Mit den vorherigen Abschnitten wurde die Grundlage geschaffen, um nun mit der Erweiterung der Bolt JS Anwendung zu beginnen. Ab jetzt kann auf Events und Actions reagiert werden, die von den Nutzern der Slack-App ausgelöst und an die AWS-Lambda-App gesendet werden. Dies umfasst auch die Erstellung eines Home-Tabs für die Slack-App sowie die Verarbeitung von Nutzerdaten und das Senden von Nachrichten.

5.3.1 Home-Tab der Slack-App

Der Home-Tab einer Slack-App existiert nicht automatisch von Beginn an, da viele Apps gar keinen Home-Tab benötigen. Daher muss dieser zuerst unter `api.slack.com > App Home` aktiviert werden. Danach wird eine Home-View benötigt, welche die Anordnung der UI-Elemente, besser gesagt der „Blocks“, in JSON beschreibt. Im Anhang A finden sich beispielhafte JSON-Darstellungen von Views, die auf Blocks basieren. Diese Darstellungen dienen dazu, den Aufbau einer solchen JSON-View zu veranschaulichen. Die Home-View besteht immer aus einer einzelnen View, welche bis zu 100 Blocks enthalten kann [40]. Es bietet sich an, einen Ordner `views` zu erstellen, und in diesem alle JSON-Dateien für die verschiedenen Views (neben der App-Home-View zum Beispiel auch Views für Nachrichten) abzulegen. Die Views können mithilfe des Block Kit Builders unter `https://app.slack.com/block-kit-builder` prototypisch erstellt und entsprechend angepasst werden. Doch wie kommt die Home-View nun auf den Home-Tab? Dafür muss unter `api.slack.com > Event Subscriptions > Subscribe to bot events` das Event `app_home_opened` abonniert und anschließend im Code festgelegt werden, wie auf das Event reagiert werden soll. Das geschieht über den Aufruf `app.event()` in `app.mjs`. Eingabeparameter sind der Name des Events und ein asynchroner Callback, der die Objekte `body` und `client` enthält (siehe Listing 5.3), wobei `body` der gesamte Event-Payload und `client` die Web-Client-Instanz ist, die zur Interaktion mit der Slack-API verwendet wird.

```
1 app.event('app_home_opened', async ({ body, client }) => {
2   await processAppHomeOpened(body, client);
3 });
```

Listing 5.5: `app.event()` Aufruf (Beispiel)

Das Event wird in der Funktion `processAppHomeOpened` verarbeitet, die sich im Ordner `events` befindet. Dieser Ordner enthält alle Dateien zur Eventbehandlung. In der Funktion `processAppHomeOpened` werden zunächst die Informationen des Users, der das Event getriggert hat, aus der Datenbank geladen. Mit diesen Informationen werden

die Blocks dann entsprechend aktualisiert, indem die aus der Datenbank geladenen Werte als `initial_options` für Radio Buttons etc. gesetzt werden. Zum Schluss muss die View natürlich noch veröffentlicht werden. Dies geschieht wie in Listing 5.6 zu sehen.

```
1  const result = await client.views.publish({
2    user_id: body.event.user,
3    view: homeView
4  });
```

Listing 5.6: Veröffentlichen einer View für einen bestimmten User

5.3.2 Interaktivität

Nachdem ein neues Container-Image erstellt, in das ECR-Repository gepusht und anschließend als neue AWS-Lambda-Funktion deployed wurde, sollte der Home-Tab die gewünschten Blocks anzeigen. Allerdings haben interaktive Elemente wie Buttons noch keine Funktion. Das Hinzufügen der gewünschten Reaktion auf Button-Klicks etc. funktioniert ähnlich wie das Abonnieren des App-Home-Events, nur dass die Aufrufe für diese Elemente nicht `app.event()`, sondern `app.action()` lauten müssen. Ein Beispiel ist in Listing 5.7 zu sehen. Dieser Aufruf erhält neben einer Callback-Funktion den Wert des `action_id`-Attributs des Block-Elements. Jedes Block-Element vom Typ `action` kann so eine Action-ID besitzen. Wichtig ist außerdem, die Event Subscriptions unter `api.slack.com` stetig aktuell zu halten, das heißt eine neue Subscription hinzuzufügen, wenn auf ein neu auftretendes Event reagiert werden soll.

```
1  app.action('send_exercise', async ({ ack, body, client }) => {
2    await processSendExercise(ack, body, client);
3  });
```

Listing 5.7: `app.action()` Aufruf (Beispiel)

Für alle Auswahl-Elemente, die sich auf dem Home-Tab befinden (Checkboxen, Static und Multi Static Selects etc.) wird in der jeweiligen `processX`-Funktion ähnlich vorgegangen:

- Der ausgewählte Wert oder die ausgewählten Werte (welche beim Auslösen der Action im `body`-Objekt mitgegeben werden) werden wie in Listing 5.8 ausgelesen
- Die View wird entsprechend angepasst und neu veröffentlicht (zum Beispiel muss, wenn ein User auswählt, dass er montags ins Office kommt, sein Profilbild und sein Benutzername in der Übersicht dem Tag „Montag“ hinzugefügt werden)
- Der ausgewählte Wert oder die ausgewählten Werte werden durch den Aufruf `db.saveItem()` in der Datenbank gespeichert; dabei wird nur das geänderte Attribut angegeben und für die anderen Attribute werden leere Werte eingesetzt
- Evtl. werden weitere Aktionen wie das Senden einer Nachricht gemäß Listing 5.9 durchgeführt

```
const selectedOptions = body.actions[0].selected_options;
```

Listing 5.8: Auslesen der gewählten Optionen

```
const result = await client.chat.postMessage({
  token: process.env.SLACK_BOT_TOKEN,
  channel: id, // Channel- oder User-ID
  text: "Dies ist eine Nachricht von beecaring."
  /* Hier koennen auch Blocks angegeben werden, wenn die
   Nachricht interaktiv sein soll */
});
```

Listing 5.9: Senden einer Slack-Nachricht

5.4 Ausgewählte Bolt JS Funktionen

Besondere Funktionalitäten der Slack-App, die ein wenig mehr Logik und Code benötigen, werden in einzelne JavaScript-Funktionen ausgelagert, welche sich im Unterordner `functions` des Bolt-Projekts befinden. Im Folgenden wird auf die wichtigsten Punkte dieser Funktionen eingegangen.

5.4.1 `getExercises`

Diese Funktion ist für die Rückgabe der relevanten Übungsvideos verantwortlich. Dafür werden die für einen Nutzer in der Datenbank gespeicherten Bedürfnisse als Eingabeparameter benötigt.

Die Liste der infrage kommenden Videos erhält man wie folgt: Es werden verschiedene Arrays für die verschiedenen Bedürfnisse angelegt. Diese enthalten die URLs der jeweils dazugehörigen Videos als String. Ein Array repräsentiert also auch einen Ordner des S3-Buckets mit den enthaltenen Videos. Zudem muss die in der Slack-App angebotene Auswahl an Bedürfnissen mit der Ordnerstruktur, die im S3-Bucket angelegt wird, manuell synchronisiert sein. Nun wird ein neues Array namens `videolist` angelegt. `videolist` soll die URLs aller Video-Kategorien enthalten, die der User als Bedürfnisse angegeben hat. Das bedeutet: In einer `for`-Schleife wird für jedes Bedürfnis der Wert des Bedürfnisses geprüft und der Inhalt des entsprechenden Arrays, welches das Bedürfnis repräsentiert, dem Array `videolist` durch `videolist.concat()` hinzugefügt.

Die Funktion, die `getExercises` aufgerufen hat, wählt dann lediglich noch ein zufälliges Element, das heißt eine URL, aus dem Array, das sie zurückbekommen hat. Die URL wird dann als Link auf das entsprechende Video über eine Slack-Nachricht an den User, für den das Video ausgewählt wurde, gesendet. Das Senden der Nachricht ist in Listing 5.10 dargestellt.

```
const result = await client.chat.postMessage({
  token: process.env.SLACK_BOT_TOKEN,
  channel: channelId,
```

```
4     /* Der angegebene Text erscheint nur als Benachrichtigung am
      Geraet des Users, wenn die Nachricht beim User eingeht */
5     text: `Hallo ${userName}! Starte jetzt die Uebung, die ich fuer
      dich ausgesucht habe, um dich fit und gesund zu halten. Viel
      Spass! :man_in_lotus_position: :blossom:`,
6     /* Die angegebenen Blocks sind die eigentliche Nachricht, die
      im Chat des Users mit der Slack-App angezeigt wird */
7     blocks: [
8       {
9         "type": "section",
10        "text": {
11          "type": "mrkdwn",
12          "text": `Hallo ${userName}! Starte jetzt die Uebung, die
      ich fuer dich ausgesucht habe, um dich fit und gesund zu halten.
      Viel Spass! :man_in_lotus_position: :blossom:`
13        },
14        /* Button, der den User auf die URL des Uebungsvideos
      fuehrt */
15        "accessory": {
16          "type": "button",
17          "text": {
18            "type": "plain_text",
19            "text": "Uebung starten",
20            "emoji": true
21          },
22          "value": "link_to_video",
23          "url": randomVideo,
24          "action_id": "start_video"
25        }
26      }
27    ]
28  });
```

Listing 5.10: Senden eines Übungsvorschlags

5.4.2 `getPresentUserPictures`

Diese Funktion ist für die Rückgabe der Benutzernamen und Profilbilder der Personen zuständig, die an einem bestimmten Tag angegeben haben, im Büro zu sein. Der Eingabeparameter ist also der Tag, für den man die Nutzerinformationen bestimmen möchte.

Zuerst werden mit `db.scanItems()` alle Userdatensätze aus der Datenbank geladen. Als nächstes werden nur diejenigen Datensätze herausgefiltert, deren `officedays`-Liste den jeweiligen Tag beinhaltet. Die gefilterten Ergebnisse müssen dann noch auf den jeweiligen User gemappt werden, indem zuerst von jedem Datensatz die User-ID extrahiert wird. Mit dieser kann über Slacks Web API gemäß Listing 5.11 der Nutzername und das Profilbild des Slack-Users abgefragt und anschließend gebündelt

zurückgegeben werden. Das Profilbild eines Nutzers ist dabei als URL repräsentiert, die den Pfad zum Profilbild auf den Slack-Servern darstellt und von der Slack Web API zur Verfügung gestellt wird.

```
const { WebClient } = require('@slack/web-api');
const web = new WebClient(process.env.SLACK_BOT_TOKEN);

// ...

const userInfo = await web.users.info({
  user: userId
});

// Extrahieren der Profilbild-URL und des Benutzernamens von der User-Info
const profilePictureUrl = userInfo.user.profile.image_72;
const username = userInfo.user.real_name;
```

Listing 5.11: Abrufen der User-Informationen über Slacks Web API

Die Funktion `getPresentUserPictures` wird, nachdem ein User seine Officetage bearbeitet hat, also innerhalb von `processSelectOfficedays`, und beim Öffnen von App Home, also innerhalb von `processAppHomeOpened`, aufgerufen. Dies geschieht in einer `for`-Schleife, in der die Funktion für jeden Wochentag aufgerufen und das Ergebnis, also die Liste der Profilbilder aller an diesem Tag eingetragenen Nutzer, mit dem Index des Wochentages in einer Map gespeichert wird. Wurde jeder Tag durchlaufen, existiert also eine Map, die jedem Tag die eingetragenen Nutzer, beziehungsweise deren Profilbild-URLs, zuweist. Die jeweilige Liste an Profilbildern soll nun dynamisch in der View neben dem entsprechenden Wochentag angezeigt werden. Dies funktioniert in Slack-Apps nur innerhalb eines `context`-Blocks. Einem solchen Block können mehrere Elemente zugeordnet werden, welche dann in der gleichen Zeile angeordnet werden. Es wird also für jeden Tag ein `context`-Block erstellt (siehe Listing 5.12), wobei jeder Block die folgenden Elemente erhält: Den Wochentag als Blocktyp `plain_text` (Z. 14-22), die Profilbilder als Typ `image` (Z. 25-32) und eine Information über die Anzahl der an diesem Tag eingetragenen Nutzer als `mrkdwn` oder `plain_text` (Z. 35-48). Die letzten beiden Elemente sollen natürlich nur angezeigt werden, wenn sich mindestens ein Nutzer für den jeweiligen Tag eingetragen hat. Hovert der Benutzer über ein Profilbild, soll zusätzlich der zum Bild gehörige Nutzername angezeigt werden (Z. 30). Das Ersetzen der Blocks erfolgt wie folgt: In der View werden die Blocks vor und nach dem zu ersetzenden Block mit Indizes markiert. Diese werden im Code herausgefiltert (Z. 2-3) und anschließend genutzt, um den Block, der dazwischen liegt, zu entfernen (Z. 6). Der Block wird dann entsprechend angepasst und schließlich wieder an der richtigen Stelle eingefügt (Z. 56).

```
// Indizes der Blocks vor und nach dem Block, der abgeaendert werden soll
const startIndex = homeView.blocks.findIndex(block => block.block_id === "before_overview");
```

```
const endIndex = homeView.blocks.findIndex(block => block.block_id === "after_overview");

// Entfernen des context-Blocks, der ersetzt werden soll
homeView.blocks.splice(startIndex + 1, endIndex - startIndex - 1);

// Iterieren ueber profilePicturesByDay Map und Erstellen von context-Elementen
const contextElements = [];
for (const weekday in profilePicturesByDay) {
  if (profilePicturesByDay.hasOwnProperty(weekday)) {
    const profileInfo = profilePicturesByDay[weekday];
    const day = officedayMap[weekday];
    const contextElement = {
      type: "context",
      elements: [
      {
        type: "plain_text",
        text: day
      }
      ]
    };

    // Hinzufuegen der Profilbilder mit Nutzername als Alternativtext
    profileInfo.forEach(profile => {
      const { username, profilePictureUrl } = profile;
      contextElement.elements.push({
        type: "image",
        image_url: profilePictureUrl,
        alt_text: username
      });
    });

    // Hinzufuegen der Personenanzahl, wenn diese groesser Null
    if (profileInfo.length > 0) {
      if (profileInfo.length === 1) {
        contextElement.elements.push({
          type: "mrkdwn",
          text: `1 Person`
        });
      }
      else {
        contextElement.elements.push({
          type: "mrkdwn",
          text: `${profileInfo.length} Personen`
        });
```

```
47          }
48        }
49
50        // Hinzufuegen des context-Elements zu den bereits
      existierenden Elementen
51        contextElements.push(contextElement);
52      }
53    }
54
55    // Einfuegen des abgeaenderten Blocks
56    homeView.blocks.splice(startIndex + 1, 0, ...contextElements);
```

Listing 5.12: Hinzufügen der Profilbilder und Personenanzahl zu jeweiligem `context`-Block

5.5 Zeitgesteuerte Nachrichten

Eine wichtige Funktion von Chatbots wie Slackbots ist das sogenannte „Scheduling“. Darunter versteht man die zeitliche Planung von Nachrichten. Auch beecaring soll eine Chatbot-Funktion besitzen, um täglich eine Gesprächsstarter-Nachricht in einen Channel senden zu können und um Übungsvorschläge zu bestimmten Zeiten (je nach gewünschter Anzahl des Nutzers) verschicken zu können. Scheduling kann mit Slack jedoch auf verschiedene Arten passieren.

Eine Option in Slack ist beispielsweise die Nutzung des Workflow-Builders, bei dem man einen einfachen Workflow nach einem Zeitplan erstellen kann. Dieser wird jedoch von der App unabhängig erstellt und ist für Aufgaben, die von App-Variablen abhängig sind, oder für komplexere Logik ungeeignet.

Aus diesem Grund bietet es sich an, komplexere Nachrichten direkt im Bolt-Programm zu planen. Slack bietet hierfür die Methode `client.chat.scheduleMessage()`, welche neben `channel` und `text`, die auch für `client.chat.postMessage()` benötigt werden, ein `post_at`-Attribut besitzt, das die Zeit, zu der die Nachricht gepostet werden soll, im Unix-Zeitstempel-Format angibt [46]. Soll die Nachricht aber in einem bestimmten Intervall (beispielsweise täglich oder wöchentlich) wiederholt werden, stößt man auch hier an eine Grenze. Dann ist ein wenig mehr Logik notwendig, um die Wiederholung der Nachricht sicherzustellen.

Ein zuverlässiger Ansatz, der genutzt werden kann, um Slack-Nachrichten durch eine Wiederholungslogik zeitgesteuert zu senden, besteht darin, Amazon EventBridge zu nutzen. Ein einfacher Fall ist dabei eine Nachricht, welche lediglich in einen Channel gepostet wird und dabei keine weiteren Aufgaben (wie die Verarbeitung von Nutzerdaten) übernimmt, also nichts anderes als ein einfacher Chatbot. Dafür kann eine neue, eigene AWS-Lambda-Funktion direkt über AWS erstellt werden. In Listing 5.13 ist der Code für das Verschicken einer einfachen Nachricht abgebildet. Es handelt sich dabei um einen Python-Code. Interessant ist dabei besonders Zeile 12, welche eine HTTP-Anfrage an die URL erstellt, die in der Umgebungsvariable `MY_WEBHOOK` gespeichert ist. Der Wert dieser Variable, sprich die Webhook-URL, lässt sich über `api.slack.com` >

`Incoming Webhooks` entnehmen, sobald eingehende Webhooks aktiviert wurden. Die Umgebungsvariable `MY_WEBHOOK` wird nachfolgend in den Konfigurationen der AWS-Lambda-Funktion, zu finden unter `Konfiguration > Umgebungsvariablen`, festgelegt und bekommt den Wert der Webhook-URL.

```
import json
import urllib.request
import os

def lambda_handler(event, context):
message = {
  "text": "..."
}

# Sendet die Nachricht ueber eine Webhook an Slack und bestaetigt
 , dass die Operation erfolgreich war
header = {'Content-Type': 'application/json; charset-utf-8'}
req = urllib.request.Request(os.environ["MY_WEBHOOK"], json.dumps
 (message).encode('utf-8'), header)
response = urllib.request.urlopen(req)
response.read()

return {
  'statusCode': 200
}
```

Listing 5.13: Chatbot-Funktion über AWS Lambda mit Python

Wie bereits in Kapitel 4.5 erläutert, wird als Trigger für die AWS-Lambda-Funktion eine Amazon EventBridge Regel erstellt, welche die Funktion zu bestimmten Zeitpunkten auslöst. Wenn die Regel beispielsweise aus dem Ausdruck `cron(0 12 ? * MON-FRI *)` besteht, bedeutet das, dass jeden Mittag von Montag bis Freitag um 12:00 ein Event eintritt. Der Entwickler wird über Amazon EventBridge Schritt für Schritt durch den Prozess zum Erstellen eines neuen Zeitplans geleitet und baut somit eine vollständige Pipeline für den Chatbot auf.

Gesprächsstarter-Bot
Im Fall des Gesprächsstarter-Bots wird bei jedem Auslösen der AWS-Lambda-Funktion (sprich: jeden Werktag um 12:00 Uhr) ein neuer Text dynamisch erstellt, der zufällig einen Emoji aus verschiedenen Unicode-Ranges wählt, sodass im besten Fall täglich eine andere Nachricht in den Team-Channel gesendet wird, die Mitarbeiter täglich zu neuen Gesprächen anregen soll (siehe Abbildung B.6).

Übungsvorschlags-Planung
Aufgrund der zeitlichen Beschränkung dieser Arbeit konnte das Scheduling der Übungsvorschlag-Nachrichten nicht mehr umgesetzt werden. Jedoch wurde bereits ein Entwurf für die Implementierung dieser Funktionalität ausarbeitet. Dieser ist in Abbildung 5.2 dargestellt und wird im Folgenden kurz erklärt.

Die Idee hinter dem Entwurf ist, dass in Amazon EventBridge alle möglichen Events

für diejenigen Zeiten definiert werden, zu denen Übungsvorschläge gesendet werden können:

- Hat der User „1 x täglich“ als Übungsanzahl gewählt, soll er um 14:00 Uhr eine Nachricht mit einem Übungsvideo bekommen
- Hat der User „2 x täglich“ als Übungsanzahl gewählt, soll er um 10:30 Uhr und um 16:00 Uhr eine Nachricht mit einem Übungsvideo bekommen
- Hat der User „3 x täglich“ als Übungsanzahl gewählt, soll er um 10:30 Uhr, um 14:00 Uhr und um 16:00 Uhr eine Nachricht mit einem Übungsvideo bekommen
- Hat der User „4 x täglich“ als Übungsanzahl gewählt, soll er um 10:30 Uhr, um 12:00 Uhr, um 14:00 Uhr und um 16:00 Uhr eine Nachricht mit einem Übungsvideo bekommen

Daraus resultiert, dass nur um 10:30 Uhr, 12:00 Uhr, 14:00 Uhr und 16:00 Uhr Nachrichten geschickt werden, und zwar nur an Werktagen. Diese Events können in Amazon EventBridge als Zeitpläne in Form von Cron-Ausdrücken beschrieben werden, und als Trigger wird jeweils eine neue AWS-Lambda-Funktion gewählt, die gleich näher beschrieben wird. Wichtig ist, dass ein Event die Lambda-Funktion mit einem `time`-Parameter Aufruft, der den Wert der Trigger-Zeit besitzt. Der Parameter kann als Payload „key: value“ übergeben werden.

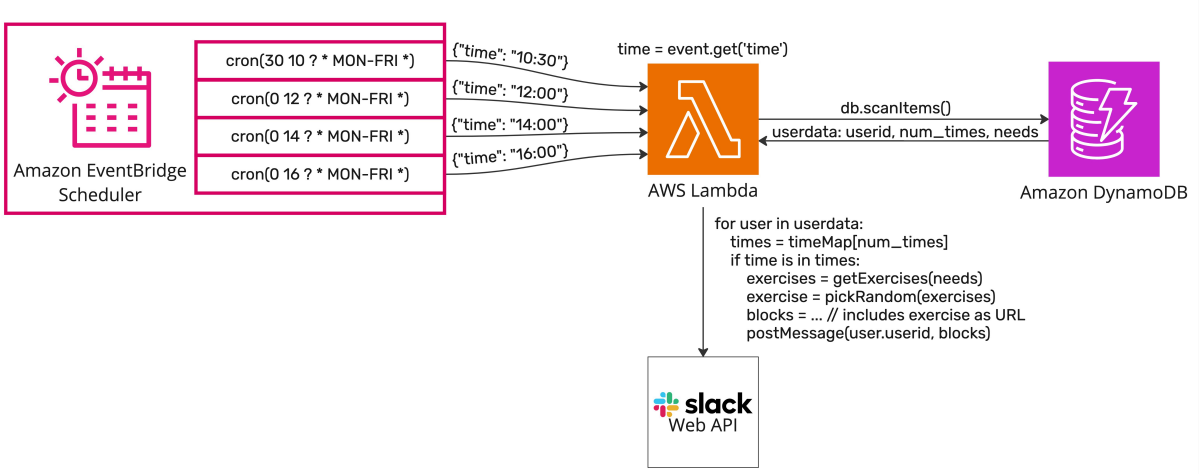

Abbildung 5.2: Übungsvorschlag-Scheduling: Modellierung mit Pseudocode (Quelle: Eigene Darstellung)

Die AWS-Lambda-Funktion, die speziell für diese Funktionalität erstellt wird, führt dann einen Datenbank-Scan durch und prüft für jeden User, ob der User um die in `time` enthaltene Uhrzeit eine Nachricht mit einem Übungsvorschlag bekommen muss. Wenn ja, wird für diesen User ein passendes Video ausgewählt und mit diesem wird dann eine Nachricht aus Blocks zusammengestellt, die an den User gesendet wird.

Wie bereits erwähnt ist dies jedoch vorerst ein Entwurf und beinhaltet weitaus mehr Logik als in Abbildung 5.2 dargestellt. Anders als beim Gesprächsstarter-Bot empfiehlt es sich hier nämlich nicht, die Funktion direkt in AWS Lambda zu schreiben, sondern ein

neues Docker-Image zu erstellen und dieses als AWS-Lambda-Funktion auszuwählen. Dadurch kommt wieder Amazon ECR ins Spiel, und da auch die passenden Übungsvideos abgerufen werden müssen, ist zudem eine Verbindung zu DynamoDB sowie der Zugriff auf den S3-Bucket erforderlich. Außerdem muss noch ein Weg gefunden werden, wie die Verifikation (siehe Kapitel 5.2.3) gelingt, da die Bolt-basierte Lambda-App in diesem Fall keine Anfrage von Slack, sondern von AWS bekommt. Das heißt, sie wird nicht von einem Slack-Event getriggert, sondern von einem EventBridge-Zeitplan, und dies könnte ein Problem für den Handler darstellen.

Kapitel 6

Evaluation

Nachfolgend wird anhand der in Kapitel 3.4.2 erarbeiteten User Stories analysiert, inwiefern das in dieser Arbeit prototypisch entwickelte Produkt den Anforderungen an das System entspricht.

US-1 Alle Akzeptanzkriterien wurden erfüllt. Auf dem Home-Tab der App können Nutzer über eine Multiple Selection eintragen, an welchen Wochentagen sie im Büro anwesend sind. Die Auswahl wird in einer DynamoDB-Tabelle gespeichert.

US-2 Alle Akzeptanzkriterien wurden erfüllt. Auf dem Home-Tab der App können Nutzer sowohl ihre eigenen als auch die Bürotage anderer Standort-Mitarbeiter einsehen. Dies geschieht über eine bildliche Darstellung der Nutzerprofilbilder und Nutzernamen pro Wochentag. Hierfür werden die DynamoDB-Datensätze aller App-User auf die gespeicherten Bürotage überprüft, und die Nutzer den jeweiligen Tagen zugeordnet. Außerdem wird über die Web API für jeden User eine Information ausgelesen, welche die zugehörige Profilbild-URL und den Benutzernamen zurückgibt.

US-3 Alle Akzeptanzkriterien wurden erfüllt. Die eigene Auswahl der Bürotage kann unter dem Home-Tab jederzeit abgeändert werden. Ändert der Nutzer seine Auswahl, wird die in DynamoDB gespeicherte Liste, welche die Bürotage enthält, mit den Änderungen überschrieben.

US-4 Alle Akzeptanzkriterien wurden erfüllt. Über einen Button auf dem beecaring Home-Tab kann der User einen Übungsvorschlag anfordern, der ihm per Nachricht direkt zugesendet wird. Die Nachricht enthält die Übung in Form eines Videos, welches über einen S3-Bucket gehostet wird.

US-5 Alle Akzeptanzkriterien wurden erfüllt. Auf dem Home-Tab kann der User aus einer Reihe von Bedürfnissen in Form von Checkboxen wählen, welche Übungsvorschläge er aktuell wünscht. Die Auswahl kann jederzeit geändert werden und wird nach jedem Bearbeiten neu in der Datenbank abgespeichert. Sendet die App zukünftig Nachrichten mit Übungsvorschlägen, werden die aktuellen Bedürfnisse des Users berücksichtigt und nur solche Videos vorgeschlagen, die in die passende Kategorie fallen.

US-6 Alle Akzeptanzkriterien wurden erfüllt. Der User hat über ein Dropdown auf dem Home-Tab die Möglichkeit, die Anzahl der Übungsvorschläge pro Tag einzustellen. Er kann dabei zwischen 0-mal und 4-mal täglich wählen. Somit kann er die Homeoffice-Übungen auch deaktivieren, wenn kein Interesse besteht. Die gewählte Anzahl wird in der DynamoDB-Tabelle gespeichert.

US-7 Alle Akzeptanzkriterien wurden erfüllt. Die aktuelle Auswahl der Anzahl an Übungsvorschlägen bleibt für den jeweiligen Nutzer auf dem Home-Tab der App sichtbar und kann jederzeit abgeändert werden. Bei Änderung der Auswahl wird die in der Datenbank gespeicherte Anzahl mit der neuen überschrieben.

US-8 Die Akzeptanzkriterien wurden aufgrund der zeitlichen Beschränkung dieser Arbeit nicht erfüllt. Es ist zum aktuellen Zeitpunkt also noch nicht möglich, dass ein User Übungsvorschläge über den Tag verteilt (je nach der ausgewählten Anzahl an Übungen) erhält. Daher macht die in US-6 und US-7 behandelte Auswahl der Übungsanzahl im Moment nur wenig Sinn. Bevor die App in dem für sie bestimmten *codecentric*-Workspace installiert wird, muss also die Auswahl für die Übungsanzahl ausgeblendet werden. Sie kann wieder hinzugefügt werden, wenn eine nächste Version der App entwickelt wurde, welche dann zusätzlich die zeitlich geplanten Übungsvorschläge beinhaltet. Ein Entwurf zur Umsetzung dieser Funktionalität existiert bereits (siehe Kapitel 5.5). Bis zur Implementierung der zweiten Version ist es Usern jedoch bereits möglich, eine Übung zum aktuellen Zeitpunkt anzufordern.

US-9 Alle Akzeptanzkriterien wurden erfüllt. Eine dynamische Nachricht wird täglich zur Mittagspause in einen Team-Channel gesendet. Diese enthält stets ein zufälliges Element, das mit jeder Nachricht variiert, und somit täglich für neue Gespräche sorgen soll.

Ein Blick zurück auf Abbildung 3.2 und Abbildung 3.3 zeigt, dass ursprünglich geplant war, die Auswahl der Officetage sowie die Auswahl der Anzahl an Übungsvorschlägen und der Bedürfnisse auch über Nachrichten von beecaring zu ermöglichen. Dies sollte den Nutzer dazu anregen, seine Angaben regelmäßig zu aktualisieren. Derzeit ist diese Funktionalität jedoch nicht vorrangig, da alle Angaben auch im Home-Tab gemacht werden können. Künftig wäre es denkbar, dass die User jeden Freitagnachmittag daran erinnert werden, ihre Bürotage für die kommende Woche neu einzutragen, oder dass beecaring jeden Morgen nach dem Befinden des Nutzers fragt, sodass dieser seine Bedürfnisse direkt im Chat mit der App angeben kann. Um jedoch eine Überlastung des Nutzers mit zu vielen Benachrichtigungen zu vermeiden, ist es sinnvoll, die Slack-App vorerst ohne Erinnerungsnachrichten für den Standort zu veröffentlichen. Nach einer Testphase kann Feedback eingeholt werden, um festzustellen, ob solche Erinnerungen gewünscht sind.

Zusammenfassend kann man sagen, dass im Rahmen dieser Arbeit alle User Stories der Priorität *hoch* und sogar einige Stories mit niedriger Priorität erfolgreich umgesetzt werden konnten. Anhang B enthält Screenshots der entwickelten App, die zusätzlich einen Überblick darüber geben, welches Produkt am Ende dieser Arbeit entstanden ist. Somit kann die App nach einem Austausch der Übungsvideos bereits im tatsächlichen

Unternehmens-Workspace installiert und in Betrieb genommen werden, und noch fehlende Features in der nächsten Version realisiert werden. Erst nach der Migration in den Ziel-Workspace und einer längeren Testphase der Mitarbeiter lässt sich eine Aussage darüber treffen, ob die App tatsächlich den Herausforderungen im Homeoffice entgegenwirkt und die soziale Standortpflege verbessert. Mit der Entwicklung der App wurde jedoch bereits eine gute Grundlage geschaffen, an einer Verbesserung der Mitarbeiterzufriedenheit im Homeoffice-Zeitalter zu arbeiten. Die Code-Qualität der Slack-App wurde sorgfältig durch verschiedene Methoden sichergestellt. Zum einen wurde das Testframework Jest verwendet, um Unit-Tests für die einzelnen Komponenten der App durchzuführen und sicherzustellen, dass jede Funktionseinheit wie erwartet arbeitet und alle möglichen Eingabewerte korrekt verarbeitet werden. Die Testabdeckung liegt dabei bei ca. 80 Prozent. Zusätzlich wurden durch End-to-End Tests mit verschiedenen Slack-Test-Usern echte Benutzerinteraktionen mit der App simuliert, um die Funktionalität des Systems in der tatsächlichen Umgebung zu prüfen.

Kapitel 7

Diskussion

Das Ziel dieser Arbeit war es, eine Slack-App zu entwerfen und zu entwickeln, die die soziale Standortpflege eines IT-Beratungsunternehmens verbessern soll, insbesondere im Hinblick auf die sozialen Herausforderungen, die durch das Homeoffice entstehen oder verstärkt werden. Die zentrale Frage war dabei: Wie kann dieses Ziel am besten durch eine Slack-App erreicht werden? Am Ende dieser Arbeit wurde ein möglicher Lösungsansatz entwickelt. Jedoch stellt sich an diesem Punkt eine weitere entscheidende Frage: Ist eine Slack-App überhaupt der geeignete Lösungsweg für das genannte Problem? Im Folgenden wird die gesammelte Erfahrung bei der Entwicklung einer Slack-App analysiert und sowohl auf die positiven Aspekte als auch auf mögliche Herausforderungen eingegangen.

Durch benutzerdefinierte Apps ist es möglich, Slack kontinuierlich mit neuen Funktionalitäten zu erweitern. Dies ist besonders in der heutigen Zeit nützlich, da viele Unternehmen bereits an „App Fatigue“ leiden. Das bedeutet: Sie müssen tagtäglich eine Vielzahl verschiedener Programme nutzen, was oft zu einer Überlastung und einer verminderten Produktivität führt. Laut einer Studie eines deutschen Software-Unternehmens nutzen 45 Prozent der Mitarbeiter mindestens sechs bis zehn verschiedene Anwendungen pro Woche, und vier von fünf Mitarbeitern wechseln mindestens drei- bis viermal pro Stunde zwischen Anwendungen hin und her, wodurch sich viele Arbeitnehmer gar überwältigt fühlen [21]. Indem neue Funktionen direkt in Slack integriert werden, können Unternehmen die Anzahl der benötigten Anwendungen reduzieren und so die Effizienz der Mitarbeiter steigern. Slack-Apps haben also definitiv einen großen Nutzen, wenn es darum geht, einzelne Workflows eines Unternehmens zu verbessern.

Bei der Nutzung des Block Kit Builders zur Erstellung des User Interfaces zeigte sich jedoch ein großes Defizit von Slack-Apps: Nämlich, dass die Abhängigkeit von Slacks Block Kit gewisse Einschränkungen mit sich bringt. Zwar gestaltete sich die UI-Erstellung durch eine manuelle Anordnung der Blocks und automatische Formatierung in JSON sehr benutzerfreundlich, jedoch mussten einige UI-Planungen an diesem Punkt angepasst werden, da die Auswahl der Block Elements sowie die Anordnungsmöglichkeiten dieser Elemente durch das Block Kit stark begrenzt sind. Bereits

in Abbildung 7.1 ist zu sehen, dass nur wenige Block-Typen eine größere Anzahl an Block Elements unterstützen. Zudem ist jede Home-View einer Slack-App auf eine Anzahl von 100 Blocks beschränkt. Entscheidet man sich dafür, eine Slack-App bzw. eine Bolt-Anwendung anstelle einer Webanwendung mit React, Vue, Angular oder vergleichbaren Frameworks zu entwickeln, schränkt das die Flexibilität des Designs also erheblich ein. Ist man dazu gezwungen, das Block Kit zu verwenden, um eine komplexere Slack-App zu entwickeln, kann infolgedessen die User Experience beeinträchtigt werden. Dies wirft die Frage auf, ob nicht eine externe Webanwendung die bessere Lösung wäre, wenn dadurch ein benutzerfreundlicheres Interface geschaffen werden kann. Würde man beispielsweise die Übersicht der Bürotage als Dashboard in eine externe Webpage auslagern, könnte nicht nur für eine Woche, sondern auch für Wochen im Voraus die Anwesenheit der Mitarbeiter in einer Art Kalenderansicht geplant werden. Benutzereinträge könnten übersichtlicher dargestellt werden, und es wäre möglich, Filteroptionen zu integrieren oder Visualisierungen zur Anwesenheitsquote anzuzeigen. Allerdings ist eine Anbindung an Slack erforderlich, wenn der Kontext einer Slack-App auf der externen Website beibehalten werden soll. Dies stellt durch die Nutzung der Slack API als Schnittstelle jedoch kein großes Problem dar.

Block type	Available in surfaces	Compatible block elements
Actions block	Modals Messages Home tabs	Many. See below.
Context block	Modals Messages Home tabs	Image
Divider block	Modals Messages Home tabs	None.
File block	Messages	None.
Header block	Modals Messages Home tabs	None.
Image block	Modals Messages Home tabs	None.
Input block	Modals Messages Home tabs	Many. See below.
Rich text block	Modals Messages Home tabs	None.
Section block	Modals Messages Home tabs	Many. See below.
Video block	Modals Messages Home tabs	None.

Abbildung 7.1: Einschränkungen des Slack Block Kits (Quelle: [45])

Die Entwicklung der Slack-App mit dem Bolt-Framework verlief insgesamt sehr positiv. Ein wesentlicher Faktor dafür ist die benutzerfreundliche API von Bolt, die durch umfassende Dokumentation unterstützt wird und die Erstellung von Slack-Apps erheblich vereinfacht. Sie abstrahiert komplexe Aspekte der Slack-API und stellt

Entwicklern verschiedene Funktionen und Methoden zur Verfügung, um Ereignisse zu verarbeiten, Nachrichten zu senden, auf Benutzeraktionen zu reagieren und vieles mehr. Darüber hinaus erleichtert Bolt die Authentifizierung und Autorisierung von Benutzern, was sowohl die Sicherheit als auch die Benutzerfreundlichkeit der App verbessert. Bolt ermöglicht es Entwicklern sogar, ihre bevorzugte Programmiersprache zu wählen, da verschiedene Sprachen wie JavaScript, Python und Java unterstützt werden.

Das Framework bietet mit `client.chat.scheduleMessage()` sogar ein nützliches Feature zum Planen einzelner Nachrichten im Voraus an. Die API-Methode hat jedoch eine Einschränkung: Man kann zwar eine Nachricht für ein bestimmtes Datum (beispielsweise Freitag) und eine bestimmte Uhrzeit (zum Beispiel 16:00 Uhr) planen, aber es sind keine Wiederholungen (täglich, wöchentlich, monatlich usw.) im Zeitplan möglich. Für einen Zeitraum von fünf Wochen könnte der Entwickler die fünf Timestrings noch manuell festlegen, aber für einen unbestimmten Zeitraum wird das schwierig. Hier wäre eine Wiederholungslogik wie bei Cron nützlich, sprich eine Möglichkeit, wiederkehrende Aufgaben mit einer einfachen Syntax festzulegen, die beliebig viele Wiederholungen unterstützt und flexibel anpassbar ist. Natürlich kann auch eine Lösung ohne die Methode `client.chat.scheduleMessage()` entwickelt werden, indem man beispielsweise Amazon EventBridge nutzt, aber es wäre eine wünschenswerte Weiterentwicklung von Slack, diese Funktionalität zukünftig einfacher zugänglich zu machen.

Eine weitere Einschränkung von Slack, die man während der Programmierung einer Slack-App beachten sollte, ist die limitierte Antwortzeit. Laut der Slack-Dokumentation muss die Anwendung innerhalb von 3000 ms mit `HTTP 200 OK` antworten, um ein Timeout seitens Slack zu vermeiden [43]. Besonders bei komplexen Aufgaben oder der Einbindung externer APIs reicht diese Zeit möglicherweise nicht aus. Ein Timeout-Fehler während eines End-to-End-Tests zeigte auf, dass ein Teil des Codes noch nicht optimal ausgeführt wurde. Dies führte zur Umstrukturierung einzelner Funktionen, um mehrere Aufgaben parallel ausführen zu können, und um sicherzustellen, dass die Antwortzeiten innerhalb der vorgegebenen Grenzen bleiben.

Zusammenfassend lässt sich sagen, dass die Entwicklung benutzerdefinierter Slack-Apps für Unternehmen, die Slack bereits als Hauptkommunikationskanal nutzen, eine erhebliche Bereicherung darstellt. Allerdings muss nicht immer jedes Feature innerhalb der Plattform abgebildet werden, da in manchen Fällen externe Anbindungen sinnvoller sein können. Diese ermöglichen den Aufbau komplexerer Systeme, die eine bessere User Experience bieten können. Auch in dieser Arbeit stellte sich heraus, dass zur Lösung der Problemstellung eine Slack-App alleine keine optimale Lösung darstellt. Durch eine externe Webanwendung in Kombination mit einer Slack-App könnten Benutzer allerdings weiterhin die gewohnte Slack-Umgebung nutzen, während sie von den erweiterten Funktionen und der verbesserten Benutzerfreundlichkeit der externen Webanwendung profitieren. Dies würde eine hybride Lösung darstellen, die die Vorteile beider Plattformen kombiniert und eine umfassende, benutzerfreundliche Lösung bietet. So könnte zukünftig die Übersicht der Officetage als Dashboard über eine externe Website gehostet werden, und die anderen Funktionalitäten könnten

weiterhin in der Slack-App laufen. Es sollte jedoch für jeden Anwendungsfall auf Grundlage der Anforderungen und der Komplexität individuell entschieden werden, welche Lösung die sinnvollste ist.

Best Practices

Die wesentlichen Erkenntnisse, die aus den geschilderten Herausforderungen bei der Slack-App-Entwicklung gewonnen wurden, bilden die Basis für folgende Empfehlungen:

Genaue UI-Planung: Nutzen des Block Kit Builders bereits während des Design-Prozesses, um sicherzustellen, dass die UI-Vision mit den Möglichkeiten des Block Kits vereinbar ist, bevor die Umsetzung festgelegt wird

Effizientes Scheduling mit externen Diensten: Planung wiederkehrender Nachrichten oder Aktionen mit Cron-Ausdrücken, beispielsweise per Amazon EventBridge, da Bolt derzeit keine eingebaute Wiederholungsfunktionalität bietet

Handling von Timeouts: Vermeidung von Timeout-Fehlern durch Nutzung mehrerer asynchroner AWS-Lambda-Funktionen für komplexe Aufgaben oder die Integration externer APIs

Kapitel 8

Zusammenfassung und Ausblick

In der heutigen Arbeitswelt wird es zunehmend wichtiger, Anwendungen zu bündeln und nahtlose Integrationen innerhalb einer Anwendung zu bieten, um die Anzahl der benötigten Apps oder Programme zu reduzieren und somit die Produktivität zu steigern. Auch die Kommunikationsplattform Slack ermöglicht solche Integrationen und bietet zudem die Möglichkeit, benutzerdefinierte Slack-Apps zu entwickeln. Dies eröffnet neue Chancen für Entwickler und schafft einen innovativen Markt für maßgeschneiderte Apps. In dieser Bachelorarbeit wurde eine solche Slack-App für ein Team des IT-Beratungsunternehmens *codecentric AG* entwickelt. Die Motivation hinter der Aufgabenstellung war ein Ausgleich der Nachteile, die Arbeitnehmer im Homeoffice erfahren. Besonders seit der Corona-Pandemie machten sich Auswirkungen des Homeoffices auf die Firmen- und Standortkultur bemerkbar. Das sollte durch die Slack-App *beecaring* geändert werden. beecaring kümmert sich um die Mitarbeiterzufriedenheit und das Standortmanagement, indem die App drei grundlegende Funktionen bietet:

- Die Verwaltung von Bürotagen, welche für eine bessere Übersicht sorgen und eine höhere Anwesenheitsquote schaffen soll
- Das Anbieten individuell auf die Nutzer abgestimmter Übungen im Homeoffice, um die Gesundheit der Mitarbeiter zu stärken
- Das Setzen von Impulsen zum Austausch mit Kollegen, um den Kontakt zwischen Mitarbeitern zu fördern

Nach dem Herausarbeiten der genauen Anforderungen durch User Research, Verfassen von Use Cases und User Stories sowie Erstellen von Wireframes, Mock-ups und einem Prototypen wurde die Architektur der Anwendung geplant. Für das Backend wurde dabei ein Serverless Cloud-Deployment gewählt, welches eine Infrastrukur mit verschiedenen AWS-Diensten bildet. Die Implementierung der App erfolgte mithilfe von Slacks eigenem Framework Bolt für JavaScript, das als Node.js-Modul eingebunden ist. Der Code wurde mittels Docker als Container-Image in die Amazon ECR Registry hochgeladen und anschließend als AWS-Lambda-Funktion ausgewählt. Die Schnittstelle zu Slack wird durch eine HTTP-API bereitgestellt, die mit Amazon API Gateway erstellt wurde. HTTP dient als Protokoll, wenn der App-Nutzer ein Event auslöst,

etwa durch das Klicken eines Buttons, oder umgekehrt, wenn die Bolt-Anwendung eine Nachricht oder eine View an den User sendet. Abhängig von der Richtung der Anfrage kommt dabei entweder die Events API oder die Web API von Slack zum Einsatz. Der dabei versendete Payload enthält Informationen zum jeweiligen Event und gegebenenfalls den Inhalt einer Nachricht oder eines Interfaces in Form von JSON. Das User Interface setzt sich aus sogenannten Blocks zusammen, die ebenfalls im JSON-Format vorliegen. Diese Blocks sind UI-Elemente des Block Kits, dem UI-Framework von Slack, das für ein konsistentes Design von Slack-Apps sorgt. Über die API von Slack konnten unter anderem Methoden zur Eventbehandlung verwendet werden. Da eine Slack-App stark eventgesteuert ist, spielt Amazon EventBridge eine wesentliche Rolle. Damit können Nachrichten zu bestimmten Zeiten geplant werden, was der App eine Chatbot-Funktionalität verleiht. Die Speicherung der Benutzerdaten erfolgt über Amazon DynamoDB, eine NoSQL-Datenbank mit Schlüssel-Wert-Speicher. Die Arbeit beschäftigte sich auch mit dem Vergleich verschiedener Datenbanken sowie dem Vergleich von AWS Lambda und AWS Fargate im Zusammenhang mit Slack-Apps, um eine fundierte Entscheidung für die Architektur treffen zu können.

Am Ende dieser Arbeit wurde deutlich, inwiefern sich der Entwurf und die Implementierung einer Bolt-App von der Entwicklung einer Website mit React, Angular oder vergleichbaren Frameworks unterscheidet. Abschließend wurde daher diskutiert, wie man mit diesen Unterschieden und möglichen Herausforderungen umgehen kann, und wie eine Slack-Anwendung entwickelt werden kann, die von Nutzern gerne angenommen wird.

Ausblick
In der Zukunft könnte die App beecaring dahingehend erweitert werden, dass sie auch für verschiedene Standorte genutzt werden kann. Dies würde erfordern, dass im Home-Tab der App eine Standortauswahl integriert wird. Auf dieser Grundlage könnte die Übersicht der anwesenden Mitarbeiter pro Wochentag entsprechend gefiltert werden, sodass nur die Mitarbeitenden des ausgewählten Standorts angezeigt werden. Somit hätten Mitarbeiter die Möglichkeit, sich auch an einem anderen Standort einzuchecken, wenn sie diesen besuchen. Für die Implementierung dieser Funktionalität müsste jedem Nutzer zusätzlich zu jedem Wochentag ein Standort zugeordnet werden. Zudem wären die aktuellen Methoden für Datenbank-Abfragen bei einer größeren Anzahl von Nutzern nicht optimal. Um die Skalierbarkeit und Effizienz sicherzustellen, muss es zukünftig möglich sein, gefilterte Abfragen direkt auf der DynamoDB-Datenbank auszuführen, anstatt alle Daten aus der Tabelle abzurufen und erst anschließend zu filtern. Zudem ist in Planung, die Übersicht der Bürotage auf eine externe Website auszulagern, um ein umfangreiches, benutzerfreundliches Dashboard zu schaffen. In diesem könnten Bürotage bereits mehrere Wochen im Voraus geplant werden, und es könnten sogar Statistiken zur Büroanwesenheit angezeigt werden. Dies würde Mitarbeitern eine umfassende und transparente Planung ihrer Arbeitswochen ermöglichen und beecaring noch effizienter und innovativer machen.

Literaturverzeichnis

[1] Amazon Web Services Inc.: *Amazon DynamoDB Documentation*. Amazon Web Services, April 2024. https://docs.aws.amazon.com/dynamodb/. – Zugriff am 26.04.2024

[2] Amazon Web Services Inc.: *Amazon EventBridge Scheduler*. Amazon Web Services, April 2024. https://aws.amazon.com/de/eventbridge/scheduler/. – Zugriff am 29.04.2024

[3] Amazon Web Services Inc.: *Amazon S3*. Amazon Web Services, Mai 2024. https://aws.amazon.com/de/s3/. – Zugriff am 16.05.2024

[4] Amazon Web Services Inc.: *Was ist Amazon CloudWatch?* Amazon Web Services, April 2024. https://docs.aws.amazon.com/de_de/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html. – Zugriff am 29.04.2024

[5] Amazon Web Services Inc.: *Was ist Amazon DynamoDB?* Amazon Web Services, April 2024. https://docs.aws.amazon.com/de_de/amazondynamodb/latest/developerguide/Introduction.html. – Zugriff am 25.04.2024

[6] Amazon Web Services Inc.: *Was ist Docker?* Amazon Web Services, April 2024. https://aws.amazon.com/de/docker/. – Zugriff am 24.04.2024

[7] Amazon Web Services Inc.: *What is AWS Lambda?* Amazon Web Services, April 2024. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. – Zugriff am 23.04.2024

[8] Asjes, Paul: *Building Slack Bots*. 1. Packt Publishing, 2016. – ISBN 978–1–78646–080–6

[9] Avantgarde Experts: *Die gesetzliche Homeoffice-Pflicht ist weggefallen: Welche Regelung gilt aktuell in Ihrem Unternehmen?* YouGov, August 2022. https://de.statista.com/statistik/daten/studie/1326759/umfrage/homeoffice-regelungen-in-unternehmen/. – Zugriff am 04.03.2024

[10] Bilendi: *Welche Kriterien in Bezug auf die Jobzufriedenheit sind für Sie heute wichtig?* Spiegel, April 2023. https://de.statista.com/statistik/daten/studie/1403985/umfrage/kriterien-fuer-die-jobzufriedenheit-von-boomern/. – Zugriff am 05.03.2024

[11] Casperson, Matthew: *API Gateway, Cognito and Python Lambdas*. 2020. – Nur als E-Book verfügbar und unter keinem Verlag veröffentlicht

[12] Chapin, John ; Roberts, Mike: *Programming AWS Lambda: Build and Deploy Serverless Applications with Java*. 1. O'Reilly Media, 2019. – ISBN 978–1–4920–4100–9

[13] codecentric AG: *Creating the digital future together*. codecentric AG, März 2024. https://www.codecentric.de/. – Zugriff am 05.03.2024

[14] Crunchbase Inc.: *Donut*. Crunchbase, März 2024. https://www.crunchbase.com/organization/donut. – Zugriff am 08.03.2024

[15] DcyllaDB Inc.: *Introduction to DynamoDB*. ScyllaDB, April 2024. https://www.scylladb.com/learn/dynamodb/introduction-to-dynamodb/. – Zugriff am 26.04.2024

[16] Deutsche Gesetzliche Unfallversicherung e.V. (DGUV): *Welche dieser Aspekte empfinden Sie im Homeoffice als besonders belastend?* DGUV, November 2021. https://de.statista.com/statistik/daten/studie/1279392/umfrage/belastende-faktoren-im-homeoffice/. – Zugriff am 05.03.2024

[17] Donut Technologies Inc.: *donut*. Donut Technologies Inc., März 2024. https://www.donut.com/. – Zugriff am 06.03.2024

[18] Frezza, Bill: *App Fatigue Sets In As The Digital Revolution Ages*. Forbes. https://www.forbes.com/sites/billfrezza/2013/12/10/app-fatigue-sets-in-as-the-digital-revolution-ages-3/?sh=20271c622178. – Zugriff am 02.04.2024

[19] ifo Institut: *Anteil der Beschäftigten im Dienstleistungssektor, die zumindest teilweise im Homeoffice arbeiten von Dezember 2021 bis November 2022 (nach Branche)*. ifo.de, Dezember 2022. https://de.statista.com/statistik/daten/studie/1283705/umfrage/nutzung-von-homeoffice-im-dienstleistungssektor/. – Zugriff am 04.03.2024

[20] IONOS SE: *MongoDB vs. DynamoDB: Vergleich der beiden NoSQL-Systeme*. IONOS, Oktober 2023. https://www.ionos.de/digitalguide/server/knowhow/mongodb-vs-dynamodb/. – Zugriff am 25.04.2024

[21] IT Verlag für Informationstechnik GmbH: *Chaos um digitale Tools*. it-daily.net, April 2021. https://www.it-daily.net/it-management/business-software/chaos-um-digitale-tools. – Zugriff am 14.06.2024

[22] Jacobsen, Jens ; Meyer, Lorena: *Praxisbuch Usability & UX*. 2. Rheinwerk Verlag, 2019. – ISBN 978–3–8362–6953–7

[23] Lucid Software Inc.: *UML - Use Case Diagramm*. Lucidchart. https://www.lucidchart.com/pages/de/uml-anwendungsfalldiagramm. – Zugriff am 21.03.2024

[24] Löbach, Bernd: *Der Designprozess - ein Problemlösungsprozess*. Stiftung Deutsches Design Museum, 2021. https://designwissen.net/der-designprozess-ein-problemlosungsprozess/. – Zugriff am 20.03.2024

[25] Markovich, Moshe: *Supercharge your Slack Productivity*. 1. Packt Publishing, 2021. – ISBN 978–1–80056–962–1

[26] Microsoft: *Donut*. Microsoft, März 2024. https://appsource.microsoft.com/en-us/product/office/wa200006072?tab=overview. – Zugriff am 08.03.2024

[27] MongoDB Inc.: *JSON and BSON*. MongoDB, April 2024. https://www.mongodb.com/json-and-bson. – Zugriff am 26.04.2024

[28] MongoDB Inc.: *Relational vs. Non-Relational Databases*. MongoDB, April 2024. https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases. – Zugriff am 25.04.2024

[29] Officernd Limited: *Powering Flexible Working*. OfficeRnD. https://www.officernd.com/. – Zugriff am 28.03.2024

[30] Putra, Anton: *How to Build Slack Bot?* GitHub, August 2021. https://github.com/antonputra/tutorials/tree/main/lessons/076. – Zugriff am 31.05.2024

[31] Rehkopf, Max: *User Stories mit Beispielen und Vorlage*. Atlassian. https://www.atlassian.com/de/agile/project-management/user-stories. – Zugriff am 21.03.2024

[32] ricotta.team: *Trivia, Icebreakers and fun Games for Team Building*. Ricotta. https://www.ricotta.team/. – Zugriff am 02.04.2024

[33] Robin Powered: *It's time the office worked for you*. Robin. https://robinpowered.com/. – Zugriff am 28.03.2024

[34] Rosam, Mike: *Fargate vs Lambda: a comparison of serverless technologies*. Quix, November 2023. https://quix.io/blog/fargate-vs-lambda-comparison. – Zugriff am 24.04.2024

[35] Sarkar, Daivi: *DynamoDB vs. MongoDB - Battle of The Best NoSQL Databases*. ProjectPro, März 2024. https://www.projectpro.io/article/dynamodb-vs-mongodb/826. – Zugriff am 26.04.2024

[36] Semler, Jan ; Tschierschke, Kira: *App-Design*. 2. Rheinwerk Verlag, 2019. – ISBN 978–3–8362–7050–2

[37] Shevat, Amir: *Designing Bots*. 1. O'Reilly Media, 2017. – ISBN 978–1–4919–7482–7

[38] Shewale, Rohit: *14 Slack Statistics In 2024 (Users, Organizations & Revenue)*. Demandsage, November 2023. https://www.demandsage.com/slack-statistics/. – Zugriff am 07.03.2024

[39] Simon, Phil: *Slack For Dummies*. 1. John Wiley & Sons Inc, 2020. – ISBN 978–1–119–66950–0

[40] Slack Technologies LLC: *App Home*. Slack, Juni 2024. https://api.slack.com/surfaces/app-home. – Zugriff am 06.06.2024

[41] Slack Technologies LLC: *Authentication*. Slack, März 2024. https://api.slack.com/authentication. – Zugriff am 11.03.2024

[42] Slack Technologies LLC: *Bolt for JavaScript*. Slack, April 2024. https://slack.dev/bolt-js/. – Zugriff am 23.04.2024

[43] Slack Technologies LLC: *Handling user interaction in your Slack apps*. Slack, Juni 2024. https://api.slack.com/interactivity/handling. – Zugriff am 18.06.2024

[44] Slack Technologies LLC: *An introduction to the Slack platform*. Slack, März 2024. https://api.slack.com/start. – Zugriff am 11.03.2024

[45] Slack Technologies LLC: *Reference: block elements & interactive components*. Slack, Juni 2024. https://api.slack.com/reference/block-kit/block-elements. – Zugriff am 17.06.2024

[46] Slack Technologies LLC: *Send or schedule a message*. Slack, Juni 2024. https://api.slack.com/messaging/sending. – Zugriff am 11.06.2024

[47] Slack Technologies LLC: *Using Slack APIs*. Slack, März 2024. https://api.slack.com/apis. – Zugriff am 11.03.2024

[48] Slack Technologies LLC: *Eine zentrale Plattform für dein Projekt-Team und deine Arbeit*. Slack, März 2024. https://slack.com/intl/de-de/features. – Zugriff am 07.03.2024

[49] Springer, Sebastian: *Node.js*. 2. Rheinwerk Verlag, 2016. – ISBN 978–3–8362–4003–1

[50] Staiano, Fabio: *Designing and Prototyping Interfaces with Figma*. 2. Packt Publishing, 2023. – ISBN 978–1–83546–460–1

[51] Stender, Daniel: *Cloud-Infrastrukturen*. 1. Rheinwerk Verlag, 2020. – ISBN 978–3–8362–6948–3

[52] Wellness Coach: *Wellness Coach app for Slack*. Wellness Coach. https://www.wellnesscoach.live/slack. – Zugriff am 02.04.2024

[53] Winyasa GmbH: *Integrate recharging moments into your SlackWorkspace with a click!* MinQi. https://www.minqi.io/en/slack. – Zugriff am 02.04.2024

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

JSON-Darstellungen von Block Kit Views

Abbildung A.1: Minimierte JSON-Übersicht der beecaring Home-View (Quelle: Eigene Darstellung)

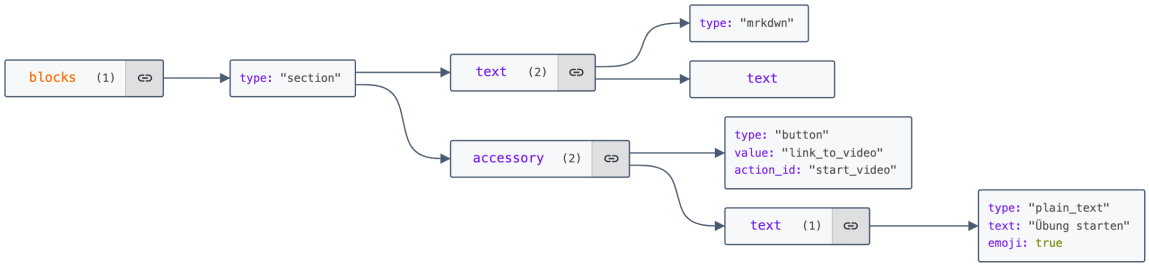

Abbildung A.2: JSON-Darstellung einer in Blocks definierten Slack-Nachricht (Quelle: Eigene Darstellung)

Anhang B

Screenshots der Slack-App beecaring

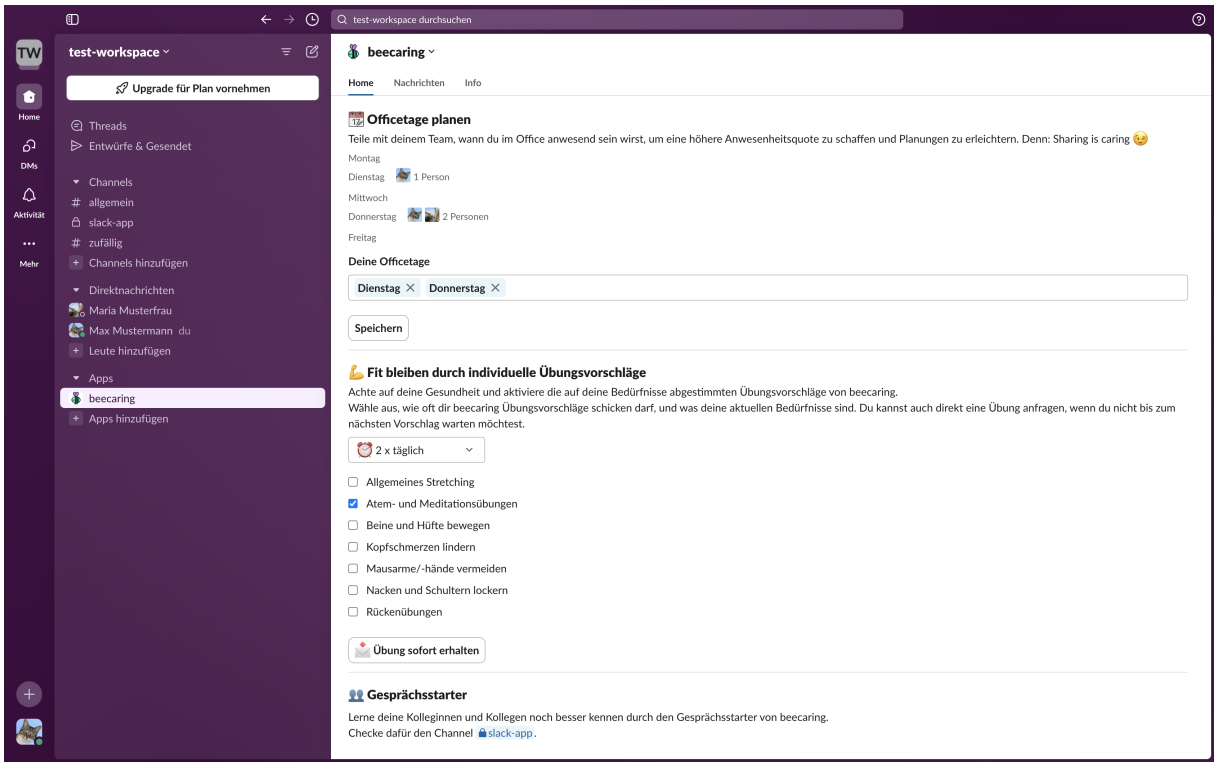

Abbildung B.1: beecaring Home-Tab (Quelle: Eigene Darstellung)

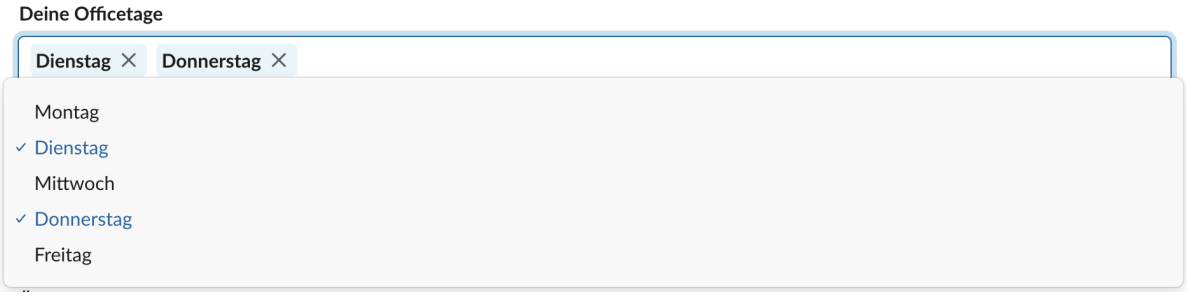

Abbildung B.2: Officetag-Auswahl (Quelle: Eigene Darstellung)

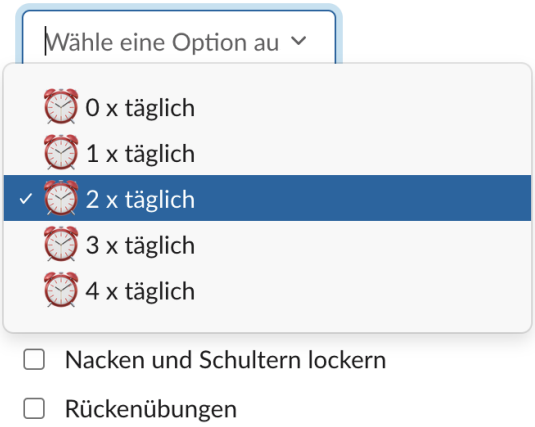

Abbildung B.3: Auswahl der Übungsvorschlag-Anzahl (Quelle: Eigene Darstellung)

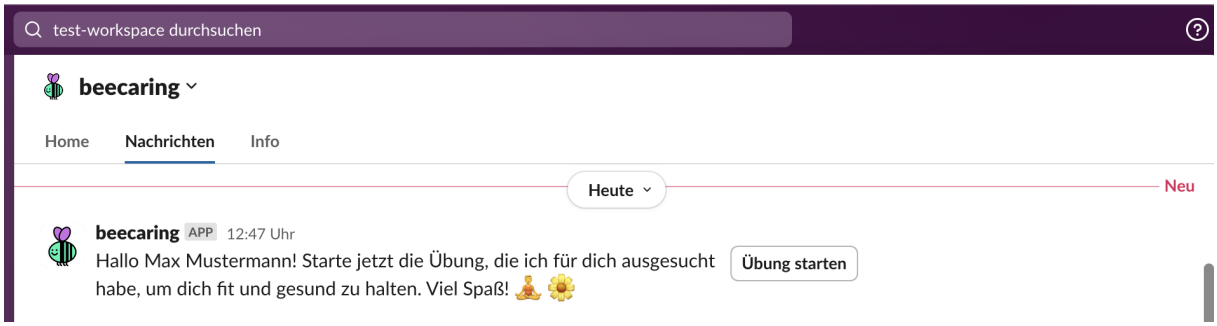

Abbildung B.4: Privat-Nachricht mit Übungsvorschlag (Quelle: Eigene Darstellung)

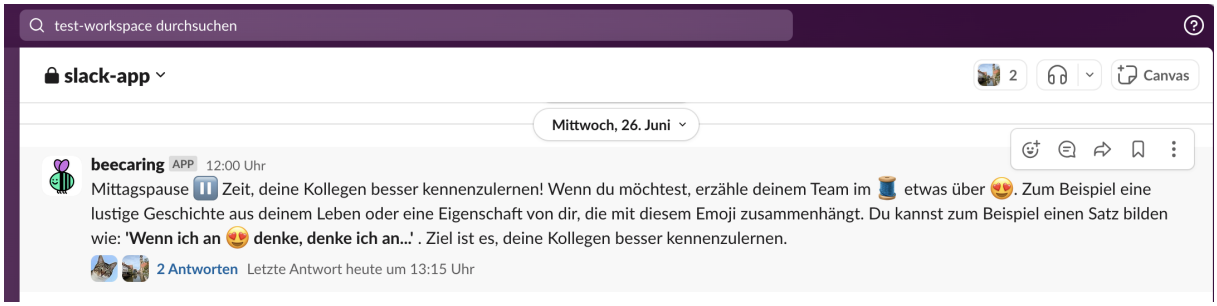

Abbildung B.5: Zeitlich geplante Gesprächsstarter-Nachricht (Quelle: Eigene Darstellung)

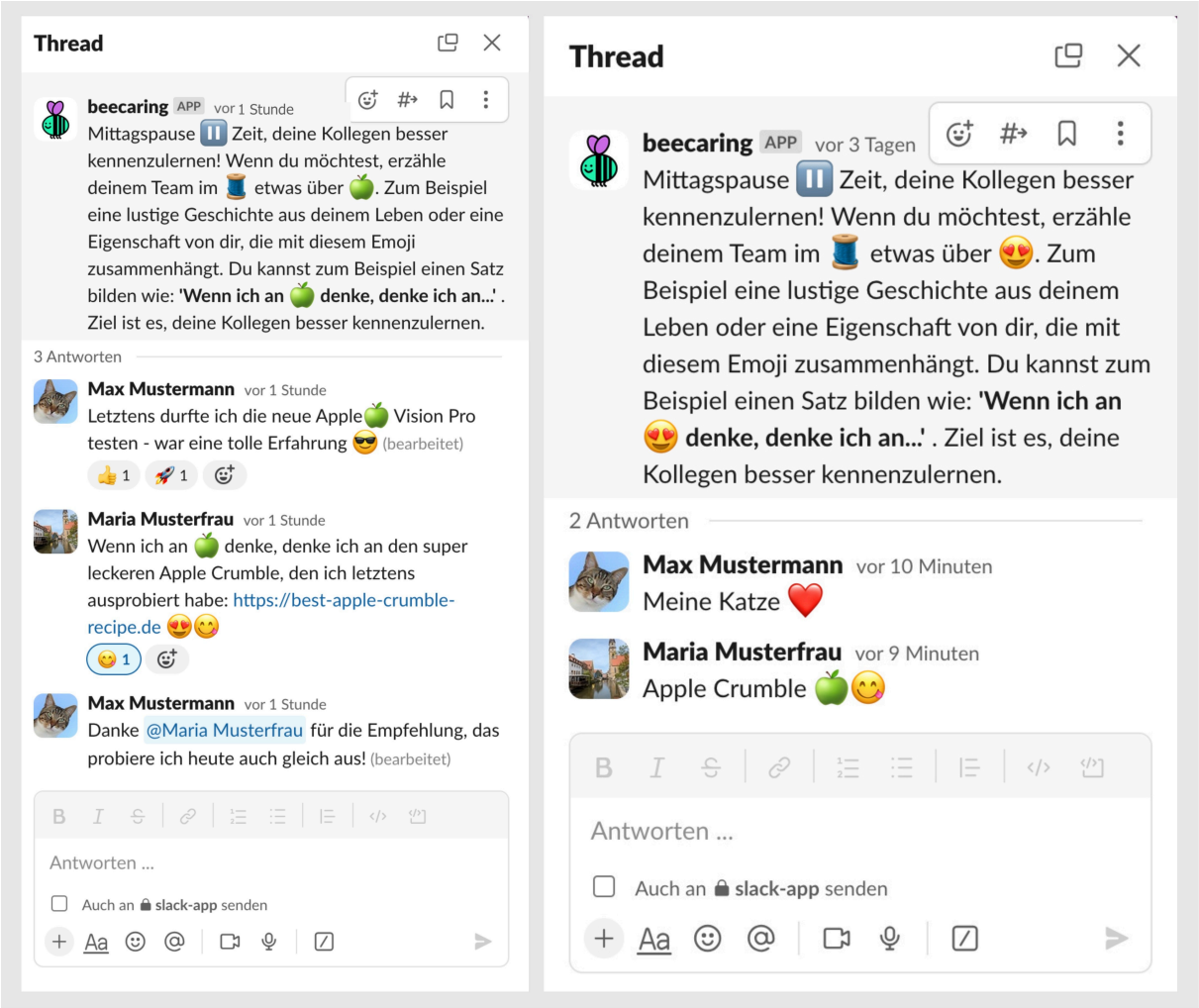

Abbildung B.6: Beispiel-Interaktionen in einem Thread durch Gesprächsstarter-Bot (Quelle: Eigene Darstellung)

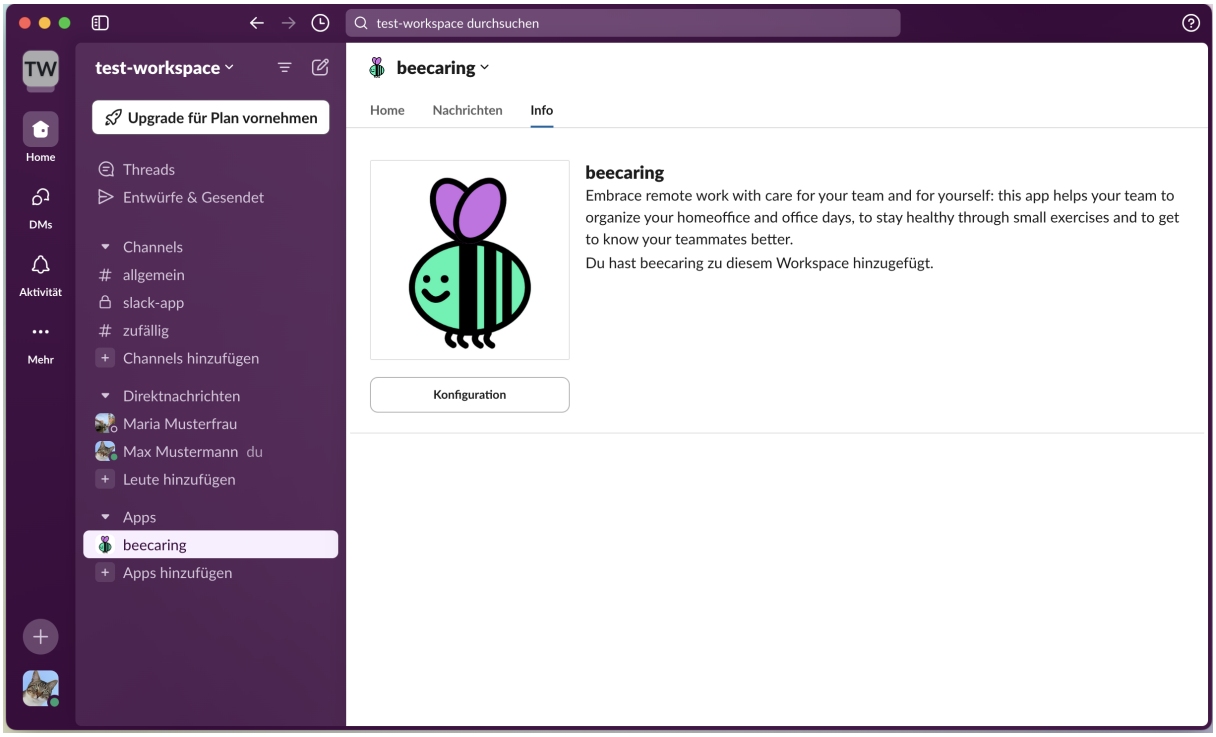

Abbildung B.7: beecaring Info-Tab (Quelle: Eigene Darstellung)

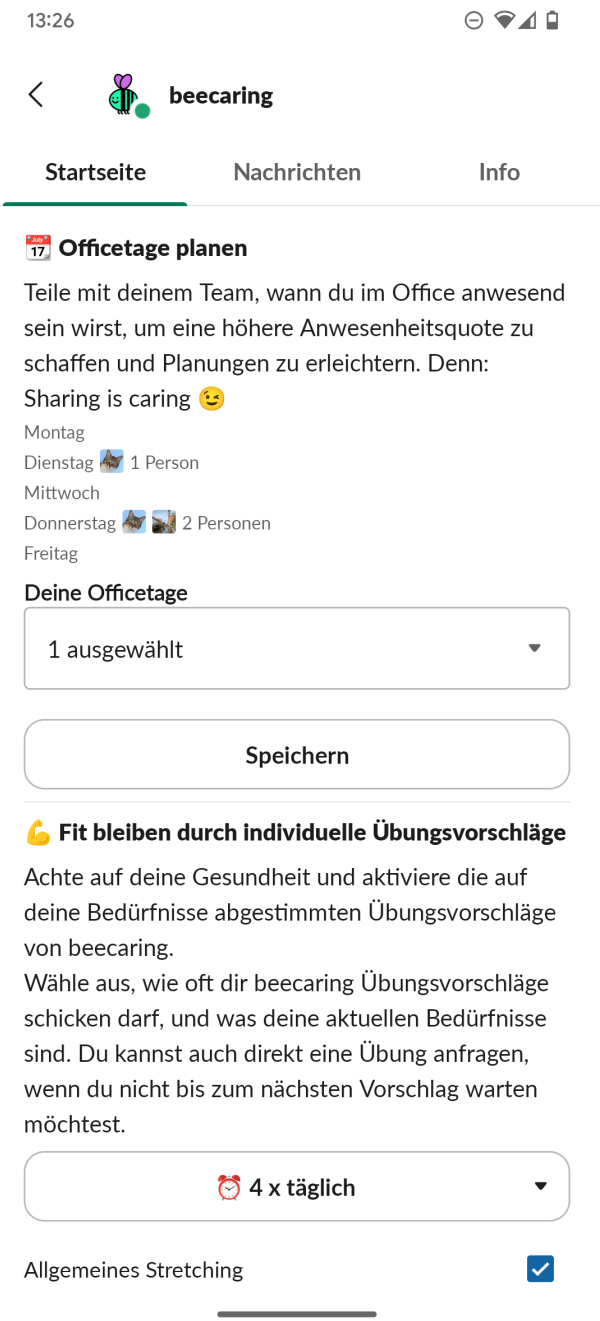

Abbildung B.8: beecaring Home-Tab in der Mobile-Ansicht - Teil 1 (Quelle: Eigene Darstellung)

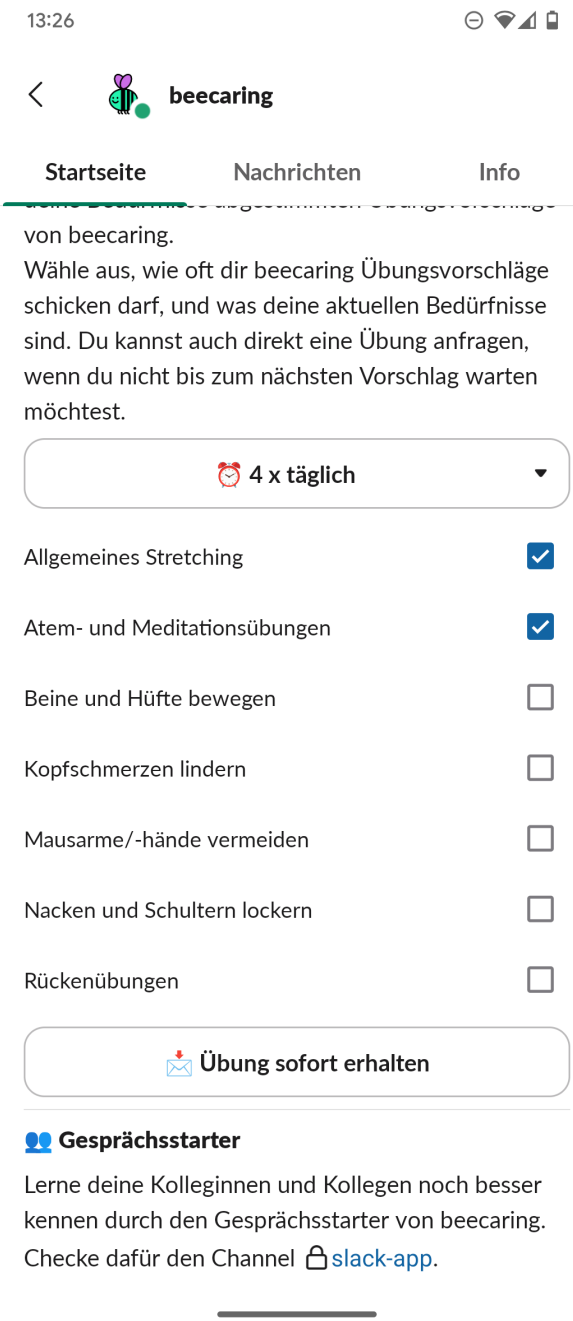

Abbildung B.9: beecaring Home-Tab in der Mobile-Ansicht - Teil 2 (Quelle: Eigene Darstellung)

Anhang C

beecaring Corporate Design

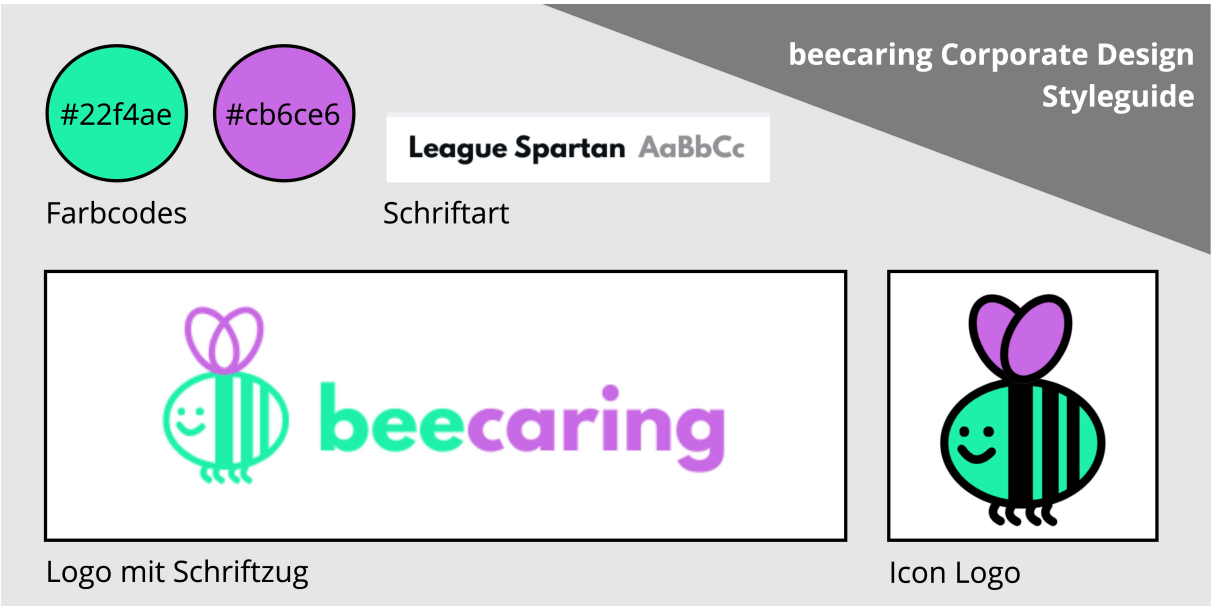

Abbildung C.1: beecaring Styleguide (Quelle: Eigene Darstellung)