Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Der Coaching Bot

The Coaching Bot

Maximilian Wellenhofer

Master-Projektstudium

Betreuer: Prof. Dr. Georg Schneider

Zürich, 28.03.2022

Kurzfassung

Provisionierung eines Bots, der die OnBoarding-Phase eines Coaching Programms automatisiert.

Am Schluss schreiben!

Abstract

The same in English.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1

Einleitung und Problemstellung

1.1 Motivation

Viele junge Menschen die nach einem abgeschlossenen Studium in die Arbeitswelt einsteigen möchten, haben keine oder wenig Erfahrung damit, wie sie sich vorbereiten sollen oder welche Schritte erforderlich sind, um einen erfolgreichen Start zu schaffen. Persönliche Beratungsleistungen und speziell EinzelCoaching können dabei helfen, sich effektiv vorzubereiten und einen erheblichen Wettbewerbsvorteil bieten, sind aber für Berufseinsteiger ohne signifikante finanzielle Mittel meist weder zugänglich noch erschwinglich. Aber auch Professionals mit hinreichenden Mitteln, haben oft Hemmungen und die Einstiegshürde ich hoch.

Um diese Zielgruppen unkompliziert und intuitiv anzusprechen und so die Einstiegshürde für derlei Services herabzusetzen, gewinnen digitale, mobile Messenger-Dienste und Social-Media-Kanäle zunehmend an Relevanz. Interaktionen mit jungen Absolventen auf klassischen Websiten stellen sich erfahrungsgemäß vor Allem im persönlichen Networking als wenig erfolgreich heraus. Daneben geht man als Coach mit einem Erstgespräch immer in eine relativ riskante Vorleistung, weil die erste Kennenlern-Session meist gratis angenoten werden muss, um überhaupt Neukunden zu gewinnen. Aus dem Bedürfnis, mit geringem kontinuierlichen Planungs- und Koordinationsaufwand ein breit gefächertes Klientel im Coaching-Bereich aufzubauen, hat man sich dazu entschieden, diverse Kanäle zu prüfen und ggf. Systeme und Technologien für einen standardisierten Onboarding-Mechanismus / Workflow zu etablieren, um zumindest den Prozess bis zum Kennenlernen von manuellem Aufwand zu abstrahieren und Kosten dahingehend auf ein Minimum zu reduzieren. Der Standardisierungscharakter ist deshalb sinnvoll, weil eine erste Kontaktaufnahme und Vorbereitung auf eine erste Sitzung erfahrungsgemäß sehr kongruent zueinander oder gar einem Skript folgend verlaufen.

Die Abwendung vom Web-Browser als Kommunikationsmedium ist keine Neuerung der letzten Jahre und schreitet mit der steten Optimierung von Messenger-Diensten wie WhatsApp, Signal, Telegram und Kommunikationsoptionen via Social Media Portalen wie Instagram, Facebook, TikTok, SnapChat, etc. weiter voran. So ist bspw. die Conversion Rate auf einem Web-Formular erheblich niedriger als

die auf einem auf der gleichen Website eingebundenen Chat-Bot.
Ziel des Projekts ist es, eine erste Version des Coaching Bots zu programmieren, die es ermöglicht, persönliche Angaben zu machen und den Nutzer zu einer Terminvereinbarung hinführt.

2

Verwandte Arbeiten

Ziel der Recherche war es, einen kleinen, selbst wartbaren open source Chatbot zu finden, der von angehenden Coaches mit minimalen Kenntnissen und einer einfach verständlichen Dokumentation ohne monetäre Mittel auf die eigenen Bedürfnisse angepasst, an einem beliebigen Ort gehostet und eine einfache, geskriptete OnBoarding-Phase durchlaufen kann. Analysiert man den Hintergrund der meisten Bots, so wird schnell klar: Die meisten Fraeworks sind zu mächtig, bieten zu umfangreiche und komplexe Feature-Sets, die für Enterprise-Grade-Software bestens geeignet sind, aber für die in der Problemstellung beschriebenen Zwecke zu weitreichend sind. Daneben sind ansonsten valable Optionen teilweise nicht direkt, aber in zweiter Instanz zu eng mit zu bezahlenden Systemen verknüpft, als dass diese ohne Weiteres genutzt werden könnten. Nach umfangreicher Recherche hat man sich ergo dazu entschieden, die Anforderungen selbst auf Basis der in 2.16 aufgeschlüsselten Systeme zu erfüllen.

Analysierte Frameworks:

1. Microsoft Bot Framework
2. Botkit
3. Botpress
4. Rasa
5. Wit.ai
6. OpenDialog
7. Botonic
8. Claudia Bot Builder
9. Tock
10. BotMan
11. Bottender
12. DeepPavlov
13. Golem
14. ParlAI by Facebook AI
15. Ana
16. Bot Libre

2.1 Microsoft Bot Framework

`https://github.com/microsoft/botframework-sdk` Als Corporate ist eine Nutzung des von Microsoft bereitgestellten Frameworks sicher aufgrund vielerlei Plug-and-play-Integrationen sinnvoll. Allerdings bindet man sich damit an die mit Kosten verbundene Cloud Plattform Azure. Das Open Source Kriterium ist also nicht erfüllt.

2.2 Botkit

`https://github.com/howdyai/botkit-cms` Botkit ist im Microsoft Bot Framework aufgegangen.

2.3 Botpress

`https://botpress.com/` Botpress ist ein sehr mächtiges Bot Framework, das grundsätzlich den oben genannten Anforderungen entspricht. Schnell wird aber klar, dass man als Laie umfangreiche Einarbeitung benötigt und, dass das Natural Language Understanding, das für fortgeschrittene Chatbos eines der Hauptfeatures ist, die hier geforderten Zwecke weit übersteigt.

2.4 Rasa

`https://github.com/RasaHQ/rasa` Rasa bietet mit seinem Story-Feature genau das, was man sich als Coach wünscht. Nämlich, den potenziellen Coachee mit auf seine persönliche Reise zu nehmen. Allerdings bedarf Rasa, um gut zu funktionieren eines umfangreichen Datensatzes, anhand dessen die AI lernen kann und das liegt uns leider nicht vor.

2.5 Wit.ai

`https://github.com/wit-ai` Wit.ai gehört Facebook (inzwischen Meta) und entspricht damit nicht unserer Vorstellung von freier Software.

2.6 OpenDialog

`https://www.opendialog.ai/` Open Dialog ist zwar open source, die Nutzung ist aber mit Lizenzgebühren verbunden.

2.7 Botonic

`https://github.com/hubtype/botonic` Botonic bietet genau das, was wir gesucht haben: Eine Kombination aus Text- und grafischen Schnittstellen. Allerdings sind wie auch für die Nutzung von Botonic, wie für Botpress, umfangreiche Vorkenntnisse erforderlich. Eine Weiterverwendung und Individualisierung durch weitere Coaches ist daher unwahrscheinlich.

2.8 Claudia Bot Builder

`https://github.com/claudiajs/claudia-bot-builder` Claudia Bot Builder reduziert die Komplexität, einen Bot selbst zu bauen und zu konfigurieren erheblich und bietet somit genau die Features, die eine einfache Adaption ermöglichen. Leider ist die Software aber ausschließlich auf AWS Lambda ausführbar und somit mit regelmäßigen Kosten verbunden.

2.9 Tock

`https://github.com/theopenconversationkit/tock` Tock ist eine valable Stand Alone Lösung für Chat Bots. Allerdings ist die Kompatibilität mit Plattformen, die ausschließlich kommerziellen Corporationen gehören, nicht mit den Zielen des Coaching Bots vereinbar.

2.10 BotMan

`https://github.com/botman/botman` Als das populärste Bot-Framework der Welt, stellt BotMan einen soliden Kandidaten für unseren Coaching Bot dar.

2.11 Bottender

`https://github.com/yoctol/bottender` Auch Bottender erfüllt auf den ersten Blick alle Anforderungen, die an das Framework gestellt wurden. Allerdings scheint es, als wären die Features, die für die Zwecke des Coaching-Bots benötigt werden, nicht einfacher zu implementieren als auf einer Chatbot Boiler Plate. Durch Bottender wären aber Komplexität und Gewicht der Applikation erheblich erhöht.

2.12 DeepPavlov

`https://github.com/deepmipt/deeppavlov` Das auf mächtige und qualitativ hochwertiges NLP ausgelegte Framework DeepPavlov ist weitaus zu mächtig und entspricht nicht dem geskripteten OnBoarding-Prozess, der für den CoachingBot verfolgt werden soll.

2.13 Golem

`https://github.com/prihoda/golem` Aus den gleichen Gründen wie bei Bottender und DeepPavlov ist uns auch Golem nicht dienlich. Weder werden für die erste Version des Bots NLU benötigt, noch bietet Golem mehr relevante Features, als die Vanilla-Version des Telegram-Bots.

2.14 ParlAI by Facebook AI

`https://ai.facebook.com/blog/state-of-the-art-open-source-chatbot/` Als Teil des Facebook- / Meta-Universums bietet ParlAI wahrscheinlich eines der besten NLPs, die aktuell verfügbar sind. Allerdings befindet sich das Framework noch in Produktion und das Feature wird nicht benötigt.

2.15 Ana

`https://www.ana.chat/` Ana bietet ein SDK, über das ein Chatbot in Applikationen integriert werden kann. Da wir aber bestehende Messenger Applikationen nutzen möchten, schließen wir Ana aus.

2.16 Bot Libre

`https://www.botlibre.com/` Bot Libre ist auf Android beschränkt. Ein dignifikanter Anteil aller Mobile-User wäre dadurch von unserer Zielgruppe ausgeschlossen.

3

Grundlagen

Die folgenden Sprachen und Systeme dienen als Grundlage für die im Rahmen dieses Projekts entwickelte Applikation. Eine Liste aller eingebundenen Bibliotheken kann dem Pipfile des Projets entnommen werden.

3.1 Python

Der größte Teil der Applikation ist in Python 3.8.6 [Pyt21d] geschrieben. Das war zum Stand des Entwicklungsbeginns 2021 die aktuell stabile Python-Version.

3.2 Telegram API

Die Applikation basiert auf der API des Instant Messaging Dienstes Telegram [ea21b] sowie deren Extension [eA21c]. Als Grundgerüst der für den Bot erforderlichen State Machine wurde der ConversationBot von Leandro Toledo et. al. genutzt. [ea21a] Das Repository enthält eine Vielzahl basaler Bot-Implementierungen, die als Startpunkt für jegliche Bot-Implementierung einen guten Überblick über das Grundgerüst und die Funktionsweise eines Bot geben.

3.3 Telegram Chat Bots

Telegram Chat Bots sind Applikationen, die auf einer quelloffenen API [ea21b] basieren, auf allen Komplexitätsstufen adaptierbar sind und es jedermann ermöglichen, einen Chatbot zu bauen. Die einzige suboptimale Einschränkung besteht im Vendor Lock-In der Telegram-App. (Der Bot ist nur in Verbindung mit der Telegram-App [Tel21b] nutzbar.) Aufgrund der technischen Vorteile, die dieses Framework bietet, ist dieser Nachteil jedoch in Kauf zu nehmen. Vor Allem im Hinblick auf einen Proof of Concept, ist ein Bot auf einer Plattform hinreichend. Sollte das Prinzip Erfolg versprechen, so kann die Logik auf andere Umgebungen erweitert werden.
Die meisten Menschen in Deutschland und der DACH-Region verwenden immer noch WhatsApp [Meh22] und doch bietet uns der populärste Messenger nicht

die Freiheiten und den Funktionsumfang, den wir uns für unseren CoachingBot wünschen. Telegram jedoch hält genau diese Offenheit für uns bereit. [Kre21]. So bietet der Dienst, die Möglichkeit, via einer API direkt in die Entwicklung einzusteigen und hält sogar basale State-Mashines für uns bereit, die uns die Komplexität für den Kern des Bots nicht komplett abnehmen, aber als Gerüst für einen ChatBot dienen können.

3.4 SQLite

Zur Speicherung von Nutzerdaten wird eine SQLite Datenbank [The21] genutzt.

3.5 SQLite3 API

Als Schnittstelle zwischen dem Python basierten Backend und der SQLite Datenbank nutzt die Applikation den sqlite3 Database-Connector [Pyt].

3.6 HMTL

HTML bietet uns die Möglichkeit die grafische Oberfläche in einem Standard Webbrowser auszugeben. [W3C19]

3.7 CSS

Zur Aufbereitung der Web-GUI wird ein CSS-File genutzt.

3.8 Flask

Die HTML-GUI wird via Flask [Pal22] an einen lokalen Web-Server übermittelt und kann so einfach mittels Python in gängigen Browsern präsentiert werden.

3.9 Google Calendar API

Die Applikation bindet die Google Calendar API [Goo22d] an, um es dem User zu ermöglichen, einen Termin mit dem Anbieter zu vereinbaren.

3.10 TheCoachingBot

Schließlich findet sich der gesamte Source-Code inklusive aller Abhängigkeiten in einem öffentlichen GitHub Repository: [Wel21]

4

Konzept

Zum Beispiel: - Blockbild der Architektur Ihrer Anwendung - Pseudocode für Algorithmen - mathematische Formeln - evtl. Diagramme auf hohem Abstraktionsniveau.

4.1 Grundkonzept

Um sich auf die in 1. gennanten Zielgruppen einzulassen, hat man sich nach einiger Analyse entschieden, in einem ersten Schritt einen Chat-Bot zu programmieren, der basale Informationen vom Nutzer abfragt und den Bewerber einen Termin vereinbaren lässt. Weitere Iterationen sind nach erfolgreicher Beta-Phase möglich. Weitere Informationen zu Erweiterungen und ein Ausblick in Kapitel 9.

Der Instant Messenger Telegram (`https://telegram.org/`) ist (neben vielen anderen) ein beliebtes Kommunikations- und Interaktionsmedium, das Funktionen weit über den einfachen Nachrichtenaustausch hinaus bietet.

Das zentrale Element der Applikation bildet der Telegram-Bot selbst. Er wird von einem Benutzer angesprochen und reagiert auf seine Eingabe. So können verschieden Funktionen ausgelöst werden. Bspw. werden Antworten zurückgegeben, Informationen gespeichert oder es wird ein Vorschlag gemacht und an den Nutzer zurückgegeben. Der Bot soll mit mehreren Benutzern gleichzeitig sprechen können. Das wird ermöglicht, weil alle Reaktionen des Bots mit der Kennung des jeweiligen Nutzers verknüpft sind. So spricht der Bot den Nutzer mit Namen an oder kann sich daran erinnnern, welche Fragen schon beantwortet wurden und welche nicht.

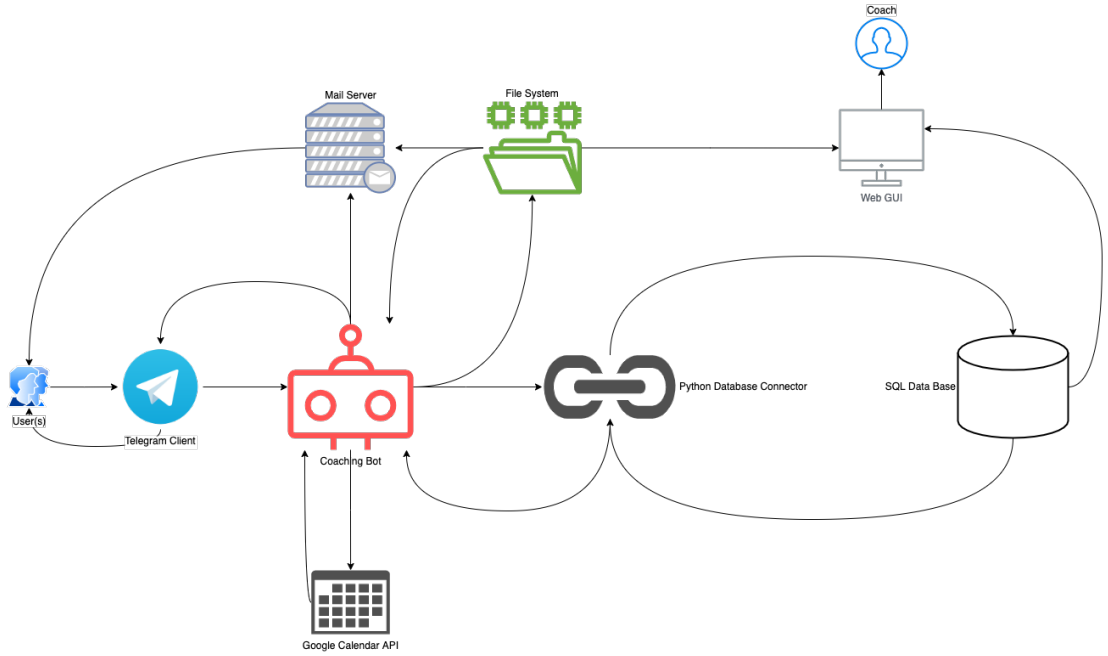

Abbildung 4.1: Abstrakte Architektur für das Projekt "Der Coaching Bot"

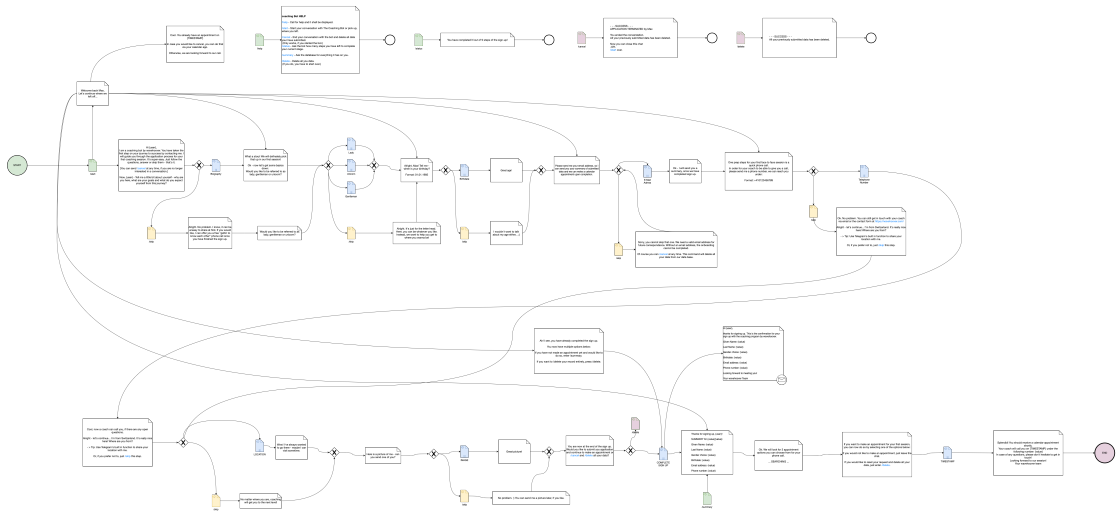

Abbildung 4.2: Konversationsfluss des Bots

5

Realisierung

Hier kommt hin, wie es gemacht haben.

5.1 Telegram Bot Framwork

5.1.1 Generierung Telegram Bot

Als ersten Schritt zur Erschaffung eines Telegram-Bots wird der Bot-Father (selbst ein Telegram-Bot [Tel21a]) konsultiert. Er erstellt das Framework, registriert den Bot und gibt ein API-Token zurück, das verwendet werden kann, um sich gegenüber der Telegram-Bot-API als Entwickler zu identifizieren. [ea21b]

5.2 Vanilla Bot Implementierung

Als Basis (boiler plate) für den Coaching Bot (fortan "Bot") nutzen wir die breit in der Community abgestütze Implementierung "Conversation Bot"[ea21a]. Sie stellt uns die basale Anbindung an die State Machine zur Verfügung und ist einfach genug, um als Einstieg in einen vordefinierten Bot zu fungieren. Im Gegensatz dazu ist der "Nested Converation Botßchon zu umfangreich und zu mächtig für unsere Zwecke.

Die Kommunikationslogik des Bots basiert auf einer State Machine. Die Zustände, in denen der Bot sich befinden kann, sind vordefiniert und immer mit einer Aktion und einer Reaktion verbunden. Aktionen werden meist von Seiten des Benutzers durch eine Eingabe oder einen Befehl ausgelöst. Reaktionen sind in Funktionen vordefiniert. Deren Umfang wird im Folgenden funktional und im Kapitel Implementierung"technisch beschrieben.

5.3 Meta-Funktionen

Neben den Hauptfunktionen des Bots (Funktionen, die zum Gesprächsfluss gehören), gibt es eine Reihe an Meta-Funktionen, die dem Nutzer zur Verfügung stehen, um eine Konversation zu starten, zu beenden, persönliche Daten zu löschen oder die Hilfe auszugeben.

5.3.1 Start: Eine Konversation beginnen

Der Bot kann gestartet werden (Aktion) und gibt eine Begrüßungsnachricht zurück. Gleichzeitig erfasst er grundsätzliche Informationen des Nutzers und schreibt diese in eine Datenbank. Aber diesem Zeitpunkt, kennt die Applikation den Benutzer und kann weitere Informationen über ihn speichern oder individuell auf Eingaben reagieren.

Die Loop-Back-Funktion

Eine der großen Herausforderungen für den Bot besteht darin, einen Nutzer wiederzuerkennen und ihn am richtigen Punkt zurück in den Konversationsfluss zu platzieren. Diese Erfahrung soll für den Nutzer nicht angestrengt wirken, sondern so, als würde der Bot ihn schon kennen und einfach da weitermachen, wo man aufgehört hat. Die technische Komplexität besteht darin, dass das Feature besonders dann funktionieren soll, wenn der Bot neu gestartet wurde.

5.3.2 Ende: Konversation manuell beenden

Hat der Nutzer eine Konversation gestartet, so kann er diese auch wieder beenden. Die Konversation muss nicht zuende geführt worden sein. Über einen kurzen Befehl /cancel"wird der Bot beendet und personenbezogene Daten werden aus der Datenbank gelöscht. Dabei ist darauf zu achten, dass der Nutzer nur seine eigenen Daten löschen kann. Hat der Nutzer seine Konversation bereits beendet, so ist /cancel"nicht mehr verfügbar. Möchte der Nutzer seine Daten dennoch löschen, so steht ihm stattdessen der Befehl /deleteßur Verfügung.

5.3.3 Persönliche Daten löschen

Zu jeder Zeit hat der Nutzer die Möglichkeit, die eigenen Daten via dem Befehl /deleteßu löschen. Die Funktion ist mit einem "Reset-Knopfßu vergleichen. Das Resultat ist nämlich, dass der Bot den Benutzer nicht mehr kennt. Er weiß nicht, dass er schon einmal da war und auch nicht, welche Angaben er gemacht hat oder nicht. So kann man den Bot nach fehlerhafter Eingabe oder, falls man neu anfangen möchte, einfach zurücksetzen.

5.3.4 Hilfe-Funktion aufrufen

Die Hilfe-Funktion gibt eine Beschreibung der Interaktionen-Optionen aus, die es gegeneüber dem Bot gibt. So werden alle Befehle einfach erklärt und können auch direkt aus der Hilfe heraus aufgerufen werden.

5.3.5 Überspringen

Die meisten Zustände des Bots erlauben es dem Benutzer, die aktuelle Frage zu überspringen. Vor allem, wenn es um personenbezogene oder private Informationen geht, die der Nutzer preisgibt, ist der Befehl /skip"verfügbar. Für jeden Zustand,

in dem /skip"verfügbar ist, ist eine individuelle Reaktion auf das Überspringen vorgesehen, die den Nutzer trotzdem abholt um in den nächsten Zustand leitet. Die einzelnen Übersprunkgsfunktionen werden im Kapitel Ïmplementierung"genauer erklärt.

5.4 Hauptfunktionen

5.4.1 Abfragen des Geburtsdatums

Um zu erfahren, wie alt der Bewerber ist, möchten wir das Geburtsdatum abfragen. Dabei ist wichtig, dass das Datum in einem sinnvollen Format eingegeben wird. (Siehe Eingabe-Validierung.)

5.4.2 Hintergrund des Nutzers

Für eine Coaching-Session ist es besonders wichtig, den Coachee besser kennenzulernen. Zu diesem Zweck hat der Nutzer die Möglichkeit, etwas über sich zu erzählen. Erwartet wird hier kein komplettes Motivationsschreiben, sondern einfache, kurz formulierte Beweggründe dafür, dass man gerne mit dem Personal Coaching beginnen möchte.

5.4.3 Abfragen des Geschlechts des Nutzers

Um den Nutzer in der Folegkommunikation korrekt anzusprechen, wird nach dem Geschlecht des Nutzers gefragt. Neben der Option, die Frage überspringen zu können, präsentiert der Bot den Nutzer mit mehr als 2 Optionen, um diversen Geschlechtern gerecht zu werden.

5.4.4 Abfragen der E-Mail Adresse des Nutzers

Um dem Nutzer eine E-Mail mit allen erfassten Daten zusenden zu können und dem eigentlichen Zweck des Bots nachzukommen - einen Termin vereinbaren zu können - benötigt der Bot eine valide E-Mail-Adresse des Nutzers. Um die Wahrscheinlichkeit zu erhöhen, dass bei dieser Eigabe keine Fehler passieren, ist auch hier eine Eingabe-Validierung hinterlegt.

5.4.5 Abfragen der Telefonnummer des Nutzers

Am Ende des Konversationsflusses hat der Nutzer die Möglichkeit, einen ersten Termin zu vereinbaren. Dabei handelt es sich um einen unverbindlichen Telefontermin. Um den Nutzer zu einer festgelegten Zeit erreichen zu können, wird hier die Telefonnummer des Nutzers erfasst. Da der Service aktuell nur in der DACH-Region angeboten wird, können hier nur Telefonnummern mit der Länderkennung Deutschland, Österreich und der Schweiz angegeben werden.

5.4.6 Abfragen des Standorts des Nutzers

Der Coaching-Service soll primär und vorerst nur in der DACH-Region angeboten werden. Daher soll der Standort des Nutzers abgefragt werden. Eine Geo-Fencing-Funktion würde für unseren Zweck hier zu weit gehen, weil wir auch Personen die Chance geben wollen, sich für den Dienst anzumelden, die aktuell im Ausland sind. So bietet die Telegram-App dem Nutzer die Möglichkeit, den Ort, den er teilen möchte, spotan selbst auszuwählen.

5.4.7 Abfragen des Bilds des Nutzers

Informationen aller Nutzer werden als Resultat der Teilnahme am On-Boarding in einer Web-GUI ausgegeben. Hier wird neben den Informationen zum Bewerber auch ein Bild angezeigt. So kann der Coach sich besser auf ein erstes Treffen einstellen.

5.4.8 Zusammenfassungs-Funktion

Ziel des Bots ist ein hohes Maß an Transparenz auf allen Seiten. Der Nutzer weiß nicht nur, dass seine Daten erfasst wurden, sondern am Ende des Konversationsflusses werden diese auch automatisch zurückkommuniziert. Dies passiert auf zweierlei Wegen. Neben einer Telegram-Nachricht wird dem Nutzer auch eine Zusammenfassung in Form einer E-Mail an die angegebene Adresse gesendet. Darüber hinaus hat der Nutzer die Möglichkeit, die Zusammenfassung jederzeit manuell abzurufen. So kann er jederzeit einsehen, welche Informationen bereits übergeben wurden und welche noch fehlen.

5.5 Support-Funktionen

5.5.1 Eingabe-Validierung

Bei einigen Angaben ist es besonders wichtig, dass Eingaben auf korrekte Formate geprüft werden. So müssen bspw. E-Mail-Adresse sowie Telefonnummer des Nutzers stimmen, um weitere Funktionen des Bots zu nutzen. Um die Wahrscheinlichkeit dafür, dass diese Eingaben korrekt sind, zu steigern, werden ausgewählte Eingaben auf Formatfehler geprüft und der Nutzer bei falscher Eingabe um eine erneute Eingabe gebeten.

5.5.2 E-Mail zusammenbauen

Die E-Mail, die am Ende des Konversationsflusses ausgegeben wird, wird separat aus verschiedenen Bausteinen zusammengesetzt. Dafür kommen Informationsabfragen gegen die Datenbank mit der Ansprache eines Mail-Servers zusammen.

5.6 Datenbank

Fast alle Informationen über den Nutzer werden in einer Datenbank gespeichert. Ausgenommen ist nur das Bild, das der Nutzer hochlädt. So können einzelne Werte jederzeit verwendet werden, um Nutzer-spezifische Reaktionen zu gestalten. Dem Nutzer stehen die meisten Datenbank-Operationen implizit und wenige explizit zur Verfügung. Daten werden implizit gespeichert und abgerufen. Explizit können Daten gelöscht werden. Zur Realisierung wird eine SLQlite Datenbank genutzt. Diese sehr einfache Datenbank ist für den Zweck des Coaching-Bots vollkommen ausreichend. Weder ist mit immensen Nutzerzahlen, noch mit vielen gleichzeitigen Operationen oder einer riesigen Datemenge zu rechen, was für mächtigere Lösungen sprechen würde. Da keine komplexen berechnungen auf den Daten ausgeführt werden, sondern nur basale CRUD-Operaionen geplant sind, gibt es nur eine Tabelle, in der alle Nutzerdaten gespeichert sind.

5.7 Anbindung Datenbank an Python

Um eine individuelle Implementierung eines Database-Connectors zu vermeiden, bedient die Applikation sich der sqlite3-Bibliothek. Sie ermöglicht es, klassische Datenbank-Operationen direkt aus einem Python-Skript heraus anzustoßen und dient hier als Database-Connector. Die Operationen selbst werden in handelsüblichem SQL formuliert und übergeben.

5.8 Kalender

Um am Ende des Konversationsflusses einen ersten Termin mit einem Coach vereinbaren zu können, muss der Nutzer einen freien Termin auswählen können und für diesen eine Einladung beantragen. Zu diesem Zweck wurde die Google Calendar API angebunden. Der Nutzer wird zunächst gefragt, ob er überhaupt einen Termin vereinbaren möchte. Daraufhin wird die API abgefragt und dem Nutzer werden drei Termine vorgeschlagen. Mit einem Klick kann der gewünschte Termin dann ausgewählt werden. Kurz darauf erhält der Nutzer eine Termineinladung an die zuvor angegebene E-Mail-Adresse und kann diese im persönlichen Kalender-Client annehmen oder ablehnen.
Eine Kennenlern-Sitzung dauert 50 Minuten. Vorschläge sollen über eine Spanne von 10 Tagen verteilt sein und nur an Wochentagen und zu Geschäftszeiten möglich sein. Da sich Geschäftszeiten ändern können und dazu keine Anpassung am Programmcode notwendig sein soll, werden Geschäftszeiten direkt im entsprechenden Kalender festgelegt.

6

Implementierung

Kann mit 5. Zusammenfallen. Manchmal eignet es sich, 2 Abstraktionsschritte zu machen (Realisierung und Implementierung getrennt).

Generell für 5. Und 6.: Wenig Quellcode (wenn überhaupt)! Maximal 2/3 Seite und immer begleitet von Erklärungen, was zu sehen ist. Kommentare im Quellcode sind nicht ausreichend. Dies gilt für UML Diagrammen analog.

6.1 Setup

6.1.1 Entwicklungsumgebung

6.1.2 Microsoft Visual Studio Code

6.1.3 pipenv Python Package Manager

Die Applikation nutzt den Package Manager pipenv. Dieser bietet die Möglichkeit, ein projektspezifisches Dokument über alle Abhängigkeiten hinweg zu erstellen und im Projekt selbst zu speichern. So können andere Entwickler Abhängigkeiten leicht installieren und müssen dies nicht auf Systemebene tun, wo es ggf. zu Konflikten mit anderen Projekten kommen könnte.
Um alle Abhängigkeiten einzusehen, pipenv [Pyt21c] installieren und das coaching_botPipfile entsprechend der Dokumentation nutzen, um alle automatisch zu installieren.

6.1.4 Konstanten und Schlüssel

Der Coaching Bot hat einige Abhängigkeiten zu Umsystemen, die Zugangsdaten voraussetzen. Diese sind im Repository [Wel21] aus Sicherheitsgründen abstrahiert und können durch kleine Anpassungen adaptiert werden.
Entsprechende Vorlagen sind unter *_constants* zu finden.

6.2 Coaching Bot Herzstück main.py State Machine

Die main.py ist das Herzstück des Coaching Bots.
Sie importiert alle HandlerFunktionen, authentifiziert sich durch den entsprechenden APISchlüssel und beinhaltet den Dispatcher, an dem wiederum Conversation

sowie CommandHandler hängen. Darüber hinaus startet sie den Bot und aktualisiert die Handler in regelmäßigen Abständen via dem Updater.

Im Folgenden werden die CommandHandler für den Coaching Bot kurz aufgeführt:

6.2.1 Dispatcher

Dispatcher liefern Nachrichten an den User aus. Pro Bot gibt es meist einen Dispatcher. Der Coaching Bot hat aber mehrere für mehrere Konversationsstränge. Command und ConversationHandler werden an diese übergeben.

6.2.2 ConversationHandlers

ConversationHandler kontrollieren den Konversationsfluss zwischen dem User und dem Bot. Pro Bot kann es mehrere ConversationHandler geben. Der CoachingBot hat aber nur einen den conv_handler. Der ConversationHandler koordiniert alle CommandHandler.

6.2.3 CommandHandler

CommandHandler nehmen Nutzereingaben via eines CallbackContexts entgegen, prüfen diese auf vordefinierte Kriterien und führen prädefinierte Funktionen sog. Handler Functions aus.

6.2.4 start und cancel Konversation starten und stoppen

Der in dieser Applikation umfangreichste ConversationHandler umfasst zwei Commands: /start und /cancel. Solange der Bot ausgeführt wird, lässt sich eine Konversation mit ihm über den Befehlt /start starten und via /cancel beenden. Zur Funktionsweise von /start und /cancel, siehe start.py und cancel.py unter Handler Funktionen.

6.2.5 delete Nutzerdaten löschen

Über den Befehl /delete wird der CommandHandler delete ausgeführt. Zur Funktionsweise von /delete, siehe delete.py unter Handler Funktionen.

6.2.6 help Hilfe ausgeben

Über den Befehl /help wird der CommandHandler help ausgeführt. Zur Funktionsweise von /help, siehe help.py unter Handler Funktionen.

6.2.7 summary Zusammenfassung ausgeben

Über den Befehl /summary wird der CommandHandler summary ausgeführt. Zur Funktionsweise von /summary, siehe summary.py unter Handler Funktionen.

6.2.8 status Status Quo ausgeben

Über den Befehl /status wird der CommandHandler status ausgeführt. Zur Funktionsweise von /status, siehe status.py unter Handler Funktionen.

6.3 Handler Funktionen

HandlerFunktionen (siehe coaching_bot/handler_functions in [Wel21]) sind Funktionen, die auf Eingaben reagieren, die via CallbackContext vom User an den Bot gesendet werden und bestimmten Kriterien entsprechen. Diese Kriterien werden direkt in der main.py in einem der CommandHandler definiert.

Im Folgenden gehen wir detailliert auf die einzelnen HandlerFunktionen ein, beschreiben deren Umfang und Aufbau und erklären ihre Funktionsweise.

6.3.1 start.py

Die Methode start in der start.py fungiert als Eingangstor für jeden User. Wann immer der Befehl /start an den Bot schickt wird, löst der CommandHandler die Methode start aus.

Zunächst wird geprüft, ob es eine Datenbank gibt. Ist dies der Fall, wird geprüft, ob der Nutzer, der die Methode ausgelöst hat, bereits in der Datenbank existiert.

Ist dies der Fall, gibt die Methode eine WillkommenzurückNachricht aus und differenziert zwischen unterschiedlichen Reaktionen auf verschiedene Zustände:

1. Befindet der Nutzer sich im Zustand *SUMMARY*, hat also bereits alle Fragen beantwortet, aber noch keinen Termin vereinbart, so werden in diesem Zustand sinnvolle Optionen empfohlen. Der Nutzer kann sich den Status seiner Bewerbung ausgeben lassen, die Zusammenfassung erneut beantragen oder alle seine Daten löschen.
2. Befindet der Nutzer sich im Zustand *APPOINTMENT*, hat aber noch keinen Termin vereinbart, die Zusammenfassun aber bereits erhalten, erhält er zusätzlich zur Option, sich die Zusammenfassung erneut ausgeben zu lassen und so die Terminfindung zu starten, nur die Status und LöschOptionen. Natürlich kann der Nutzer auch manuell alle Befehle jederzeit eingeben, aber die Tastatur ist so für Optionen vordefiniert, dass der Nutzer in eine bestimmte Richtung gelenkt wird. Nach der Ausgabe dieser Nachricht, beendet der Conversation-Handler die Kommunikation.
3. Befindet der Nutzer sich im Zustand *APPOINTMENT* und hat bereits einen Termin vereinbart, so werden Informationen zu dem Termin aus der Datenbank abgerufen und direkt ausgegeben. Auch in diesem Fall wird die Konversation nun beendet, da keine weiteren Interaktionen mit dem Nutzer vorgesehen sind.

Egal welche Option die Methode wählt, der Nutzer wird immer in den Konversationsfluss zurückgeführt und zwar genau vor der Frage, die zuletzt nicht beantwortet wurde. Eine Frage zu überspringen gilt dabei auch als Beantwortung. Dazu wird die Datenbank abgefragt und der Wert aus *state* für die entsprechende UserID an den ConversationHandler weitergegeben. Dieser präsentiert als Antwort darauf die nächste Frage im Konversationsfluss.

Treffen all diese Konditionen nicht zu, wurde der Nutzer also nicht in der Datenbank gefunden, so startet der Bot ganz normal mit einer Begrüßung, nachdem initiale Daten von der TelegramInstanz des Nutzers abgefragt und in die Datenbank geschrieben wurden.

Ist die Nachricht an den Nutzer ausgeliefert, aktualisiert der Bot den Zustand für den Nutzer in der Datenbank, damit der Bot weiß, welche Fragen der Nutzer schon beantwortet hat und er den Nutzer bei einer Rückkehr wieder am richtigen Punkt in den Konversationsfluss einfügen kann.

Bevor der Bot den Nutzer zur nächsten Stufe weiterleitet, speichert er noch einen Zeitstempel, damit man nachvollziehen kann, wann der Nutzer seinen Prozess begonnen hat.

6.3.2 bio.py

Methode bio

Die Methode bio in der bio.py speichert die TextEingabe eines Nutzers als erste Nutzereingabe nach dem /startBefehl. Sie repräsentiert besonders gut den Aufbau der HandlerFunktionen, weil sie über das Speichern und weiterleiten keine weiteren Features besitzt. Daher erläutern wir hier den Aufbau der HandlerFunktionen hier beispielhaft für alle anderen HandlerFunktionen:

```
# Stores the information received and continues on
↪  to the next state
def bio(update: Update, context: CallbackContext)
↪  -> int:

    user_id = update.message.from_user.id
    bio_message = update.message.text

    logger.info(f'+++++ Bio of user {user_id}:
    ↪  {bio_message} +++++')

    # write bio to DB
    insert_update(user_id, 'bio', bio_message)

    # reply keyboard for next state
```

```
        update.message.reply_text(
            'What a story! We will definately pick that
            ↪  up in our first session!\n\n' + \
            'Ok - now let\'s get some basics down: \n'
            ↪  + \
            states.MESSAGES[states.GENDER],

            ↪  reply_markup=states.KEYBOARD_MARKUPS[states.GENDER],
            )

        # save state to DB
        insert_update(user_id, 'state', states.GENDER)
        return states.GENDER
```

Zunächst werden ein Update und ein CallbackContextObjekt an die Handler-Methode übergeben. Zurückgegeben wird der Datentyp int, da die StateMachine am Ende der Methode wissen muss, in welchen Zustand der Nutzer als nächstes geschickt werden soll.
Innerhalb der Methode werden user_id und bio_message aus dem UpdateObjekt gespeichert, da man diese beiden informationen gleich weiterverwenden möchte. Die bio_message ist in diesem Fall die TextEingabe, die an den Bot nach der letzten Stufe (start) übermittelt wurde. Die user_id ist die TelegramID des jeweiligen Nutzers.
Nach der Ausgabe eines einfachen LogEintrags dazu, welcher Nutzer gerade welche Nachricht gesendet hat, wird der Datenbankeintrsg des Nutzers um die soeben empfangene Nachricht erweitert. (Funktionalität der inser_update Methode folgt unter Abschnitt insert_update.py) Nun kann die für den Nutzer sichtbare Reaktion auf die Nachricht erfolgen. Die update.message.reply_text Methode erlaubt es uns, dem Nutzer einen beliebigen String sowie eine für diese Nachricht individuelles AntwortTastatur auszugeben. Die zu übergebenden Parameter sind für die meisten reply_textInstanzen in der states.py zentral gespeichert, um sich innerhalb der einzelnen HandlerFunktion von möglichst viel Inhalt zu abstrahieren.
Ist die Ausgabe an den Nutzer erfolgt, bleibt noch die Aktualisirung des Zustands des Nutzers in der Datenbank, gefolgt von der Übergabe des nächsten Zustands an den ConversationHandler.

Der Aufbau aller weiteren HandlerFunktionen ähnelt der Methode bio sehr stark. Auf Erweiterungen und Anpassungen wird in den entsprechenden Abschnitten eingegangen. Die Methode leitet in den Zustand GENDER.

Methode skip_bio

Die Methode skip_bio in der bio.py wird durch den Befehl /skip ausgelöst. Dieser Befehl ist in jedem Zustand spezifisch für den CommandHandler einer Stufe definiert und hat in jedem Zustand einen anderen Effekt. In diesem Fall, wird die Methode skip_bio aus der bio.py aufgerufen. Auch für die Methode skip_bio gilt, dass sie den Aufbau der skipMethoden gut repräsentiert. Daher auch hier wieder

eine detaillierte Erklärung:

```
    # Skips this information and continues on to
    ↪  the next state
def skip_bio(update: Update, context:
↪  CallbackContext) -> int:

    user_id = update.message.from_user.id

    logger.info(f'00000 No bio submitted by
    ↪  {user_id} 00000')

    # alternative message
    update.message.reply_text(
        'Alright. No problem. I know, it can be
        ↪  uneasy to share at first. If you would
        ↪  like, I can offer you a free "gettin to
        ↪  know each other" phone call once you
        ↪  have finished the sign up.',
        reply_markup=ReplyKeyboardRemove(),
        )

    # reply keyboard for next state
    update.message.reply_text(
        states.MESSAGES[states.GENDER],

        ↪  reply_markup=states.KEYBOARD_MARKUPS[states.GENDER],
        )

    # save state to DB
    insert_update(user_id, 'state', states.GENDER)
    return states.GENDER
```

Der Aufbau ähnelt der bioMethode. Allerdings liegt hier ein reduzierter Umfang und natürlich eine andere Nachricht an den Nutzer vor. So gibt der Logger nur aus, dass keine Nachricht eingegangen ist. Ein Update der Datenbank fällt weg, da der Nutzer keine neuen Informationen angegeben hat. Hier werden zwei reply_textMethoden verwendet.
Die Erste dient dazu, eine auf diese skipMethode individuelle Nachricht zu übermitteln. Die Zweite ähnelt der Methode aus der bioFunktion. Sie übermittelt die Aufforderung zur Eingabe der Information für die nächste Stufe und zeigt die entsprechende Tastatur an. Der Rest der Methode gleicht ihrer Schwester.

6.3.3 gender.py

Methode gender

Die einzige Besonderheit der genderMethode aus gener.py liegt in der Differenzierung der Datenbankoperationen, die als Resultat der vordefinierten Antwort des Nutzers ausgelöst werden. Die Optionen "Gentleman", "Ladyünd Ünicorn"resultieren in einem nüchternen Datenbankeintrag: "male", "female", "diverse". Die Methode leitet in den Zustand "BIRTHDAY".

Methode skip_gender

Keine Besonderheiten.

6.3.4 birthdate.py

Methode birthdate

Der Bot arbeitet hier erstmals mit InputValidierung. Dazu wird die Nutzereingabe zunächst an die Methode validate_birthdate übergeben und auf die Bewertung des Inputs gewartet. Entspricht der Input dem prädefinierten Format, fährt der Bot wie gewöhnlich fort und übergibt den nächsten Zustand zurück an den ConversationHandler. Ist dies jedoch nicht der Fall, so wird eine entsprechende Nachricht an den Nutzer ausgegeben. Da der ConversationHandler erst dann zur nächsten Stufe geht, wenn er von der Methode birthdate den entsprechenden Zustand zurückerhalten hat, entsteht hier ein loop, der entweder durch eine gültige Eingabe oder eine der MetaFunktionen gebrochen werden kann.

6.3.5 email.py

Methode email

Wie die Methode auch schon, nutzt birthdate Input-Validation - dieses mal, um zu prüfen, ob eine gültige EMailAdresse eingegeben wurde.

Methode skip_email

Die Methode email ist die einzige Methode, die nicht übersprungen werden kann. Ohne eine gültige EMailAdresse des Nutzers können wichtige Folgefunktionen des Bots nicht genutzt werden und der Sinn und Zweck (Terminvereinbarung) ist nicht möglich. Daher ist die Methode skip_email so gestaltet, dass sie keinen Zustand zurückgibt, sondern den Nutzer im aktuellen Zustand belässt, bis dieser entweder eine gültige Adresse eingegeben oder einen alternativen Befehl abgesetzt hat, der ebenfalls das Ende der Konversation zufolge hat. So steht es dem Nutzer frei, die Konversation jederzeit zu beenden.

6.3.6 telephone.py

Methode telephone

Die Methode telephone funktioniert exakt gleich wie die Methode email.

Methode skip_telephone

Die Methode skip_telephone bietet dem Nutzer an, den Kontakt mit dem Anbieter alternativ via dem auf der Internetseit verfügbaren Webformular zu suchen. Dies ist generell für alle Informationen möglich, die an den Bot übergeben werden (allerdings lag der Beweggrund für die Erstellung des Bots darin, eine Alternative zum klassischen Kommunikationsmedium WebFormular zu bieten).

6.3.7 location.py

Methode location

Der Nutzer hat hier die Möglichkeit, seinen Standort anzugeben. Dazu wird die bereits in Telegram vorhandene Funktion zur Standortfreigabe genutzt.
Am Ende der Methode, bevor der Bot zur nächsten Stufe *PHOTO* weitergeht, sendet der Bot ein Bild von sich selbst, um den Nutzer dazu anzuregen, auch sein Bild von sich zu teilen. Dazu wird ein einfaches JPG verwendet, das im Repository des Bots gespeichert ist. Der Pfad kann aber auch leicht an jede andere Ressource angepasst werden.

Methode skip_location

Keine Besonderheiten.

6.3.8 photo.py

Methode photo

Entscheidet der Nutzer sich, ein Bild mit dem Bot zu teilen, so nimmt die Methode photo dieses entgegen und speichert es in einem Ordner, der je nach System gewählt werden kann. Hier wurde ein Ordner im gleichen Verzeichnis gewählt, in dem der Bot existiert. Um Bilder später wieder zuordnen zu können, wird der Dateiname jedes Bildes auf die ID des jeweiligen Nutzers gesetzt, bevor es gespeichert wird.

Methode skip_photo

Keine Besonderheiten.

6.3.9 summary.py

Methode summary

In der Methode summary kommt alles zusammen. Der Nutzer hat nun alle Angaben gemacht oder übersprungen. Die Methode beginnt damit, eine Reihe von Informationen von der Datenbank abzufragen und in variablen zu speichern. Es werden nur Informationen abgefragt, die auch in der auszugebenden Nachricht genutzt werden sollen. Direkt darauf wird der String für die Nachricht zusammengebaut und gespeichert. Es folgt eine einfache Danke-Nachricht an den Nutzer,

bevor die eigentliche Logik der Methode beginnt.

Nun gibt es mehrere Szenarien aus Nutzersicht: Der Bot prüft, ob der Nutzer bereits einen Termin vereinbart hat.

1. Option A: Der Nutzer ist bis zum Zustand *SUMMARY* gekommen, hat die Zusammenfassung ausgegeben bekommen, dann aber keinen Termin vereinbart und den Chat verlassen. Der Nutzer kehrt nun zum Chat zurück und gibt erneut /start ein, um seine Konversation wieder aufzunehmen. Der Bot findet den Nutzer in der Datenbank und leitet an die Stufe *SUMMARY* weiter.
   Diesen Fall behandelt der Bot wie folgt:
   a) Ist dem nicht der Fall, möchte er dem Nutzer jetzt drei mögliche Terminvorschläge unterbreiten und startet dazu die Terminfindung (siehe Abschnitt 6.5).
   b) Sobald die Termine zurückkommen, präsentiert der Bot diese dem Nutzer in Form eines entsprechenden Tastatur-Layouts. Das Layout ist dabei dynamisch und generiert sich bei jeder Abfrage neu.
2. Option B: Der Nutzer ist bis zum Zustand *SUMMARY* gekommen und hat bereits einen Termin vereinbart. Diesen Fall behandelt der Bot wie folgt:
   Der Bot fragt den Termin von der Datenbank ab und gibt ihn in einer Nachricht an den Nutzer zurück. Gleichzeitig schlägt er dem Nutzer weitere mögliche Befehle vor, die an dieser Stelle Sinn machen und beendet die Konversation.

Schließlich wird die Methode confirmation_mail aufgerufen, die die gleiche Zusammenfassung nochmals per EMail an die Adresse des Nutzers sendet (siehe Abschnitt confirmation_mail.py) und die Information darüber, dass an diesen Nutzer bereits eine EMail gesendet wurde wird neben den üblichen Abschlussbefehlen in der Datenbank gespeichert.

6.3.10 confirmation_mail.py

Methode confirmation_mail

Um dem Nutzer die Zusammenfassung in Form einer E-Mail zukommen zu lassen, muss diese zunächst zusammengesetzt werden. Die Möglichkeit, dies zu bewerkstelligen, bietet die Bibliothek mime. [Pyt21b] Daneben wird die smtplib-Bibliothek genutzt, um eine sichere Verbindung zu einem Mail-Server aufzubauen, über den die fertige E-Mail versenden werden kann. [Pyt21e]

Erforderliche Zugangsdaten werden außerhalb der Methode confirmation_mail aus den constants abgefragt und für die Verwendung innerhalb der Methode gespeichert. So werden diese nicht bei jedem Methodenaufruf erneut abgerufen.

Die Methode bekommt EmpfängerName sowie Adresse und die Zusammenfassung aus der Methode summary übergeben. Über die smtplib wird ein ServerObjekt erstellt. Gegenüber diesem Server authentifiziert sich der Bot nun via Benutzername

und Passwort.
War die Authentifizierung erfolgreich, wird die eigentliche Nachricht zusammengesetzt. Dazu benötigt werden vier Bauteile: SenderAdresse, EmpfängerAdresse, der Betreff und die Nachricht selbst.
Die Nachricht wird zuerst via der Methode attache() zusammengesetzt, um dann aus dem vordefinierten String ein messageObjekt zu konstruieren.
Schließlich kann die EMail via der Methode sendmail() unter Verwendung des zuvor beschriebenen MailServers versendet werden.

Schließlich wird die Verbindung zum Server wieder getrennt.

6.3.11 appointment.py

Methode appointment

Ziel der Methode appointment ist es, den Zeitstempel vom Nutzer entgegenzunehmen, ein Kalender-Event zu bauen und dieses an die Methode make_appointment zu übergeben.
Dazu fragt sie zunächst alle erforderlichen Informationen bei der Datenbank ab. Der erhaltene Zeitstempel für den Beginn des Zeitfensters wird dann in ein Format übersetzt, das die Google Calendar API akzeptiert:
(`%Y-%m-%dT%H:%M:%S+01:00`)
Das Ende des Zeitfensters wird auf 50 Minuten nach dem Start gesetzt und die beiden korrekt formatierten Zeitstempel in das Event verbaut. (Format und Aufbau des Events können via dem Google APIs Explorer getestet werden. [Goo22a])
Ist der Aufruf an die Methode make_appointment abgesetzt und ohne Fehler zurückgekehrt, so wird die Datenbank entsprechend um den StartZeitstempfel erweitert und eine Bestätigung auf der Konsole ausgegeben. Der Nutzer wird außerdem über den Abschluss seiner Anmeldung informiert.

Methode skip_appointment

Überspringt der Nutzer diesen letzten Schritt, wird ihm lediglich eine Nachricht ausgegeben, die die Option offen lässt, auf anderem Weg mit dem Anbieter in Kontakt zu treten.

6.3.12 help.py

Methode help

Die Methode help setzt ein Dictionary aus einer Liste an Befehlen zusammen, das dann ausgegeben werden kann. Ein Dictionary bietet die Möglichkeit, die Hilfe jederzeit einfach anzupassen, um Elemente zu erweitern oder zu reduzieren, ohne die Logik, über die die Hilfe ausgegeben wird, zu beeinflussen. Dazu wird die collectionsBibliothek eingebunden, die es erlaubt, ein geordnetes Dictionary zu erstellen. Nachdem der String für die Hilfe zusammengesetzt ist, wird dieser einfach via der Methode send_message() ausgegeben.)

6.3.13 states.py

STATES

Die State Machine muss zu jeder Zeit wissen, welche Zustände es gibt und in welcher Reihenfolge diese existieren. Dazu nutzt der pythonconversationbot [ea21a] ein Array aus Konstanten. So lässt sich die Reihenfolge der States auch ganz leicht ändern. Soll der Bot bspw. EMail und Telefonnummer am Afang abfragen oder sollen einige Schritte aus dem Konversationsfluss genommen werden, so sind diese hier einfach zu entfernen und die Nachrichten in den eizelnen Stufen leicht anzupassen.

MESSAGES

Um Nachrichten an den Nutzer zentralisiert zu verwalten, verweisen Handler-Methoden wo immer möglich auf eine Konstante aus dem *MESSAGES* Dictionary. So wird vermieden, dass Strings bei Anpassungen der Zustände oder deren Reihenfolge in mehreren Dateien angepasst werden müssen.

KEYBOARD_MARKUPS

Gleiches gilt für individuelle Tastaturen.

6.3.14 status.py

Methode status

Zu jedem Zeitpunkt, kann der Nutzer seinen aktuellen Status abfragen. Dazu prüft die Methode status zunächst, ob der Nutzer überhaupt in der Datenbank existiert. Hat der Nutzer seine Informationen nämlich gelöscht, existiert er für den Bot nicht. Zwei Szenarien:

1. Der Bot findet den Nutzer, gibt den aktuellen Status zurück und beendet die Konversation.
2. Der Bot findet den Nutzer nicht und zeigt dem Nutzer Optionen an, fortzufahren - namentlich die Hilfe aufzurufen oder eine neue Konversation mit dem Bot zu starten.

6.3.15 validation.py

Alle InputValidation Methoden sind ähnlich mit einem try/except oder if/else Block aufgebaut.

validate_birthdate

Die Methode bekommt die Nutzereingabe übergeben und vergleicht diese via der Methode strptime aus der datetimeBibliothek [Pyt21a] mit dem in der DACH-Region gängigen DatumsFormat: *TT.MM.JJJJ*
Stimmt die Eingabe mit dem definierten Format überein, gibt die Methode True zurück. Ansonsten wird ein ValueError geloggt und die Methode gibt False zurück.

validate_email

Diese Methode bedient sich eines relativ einfach regulären Ausdrucks, um zu prüfen, ob die Eingabe eine EMail sein könnte:

```
[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}
```

(Ein vollumfänglicher regulärer Ausdruck, ein externder Dienst oder gar das versenden einer TestEMail wurden aus PerformanceGründen ausgeschlossen.)
Ist der Vergleich erfolgreich, gibt die Methode True zurück, ansonsten False.

validate_telephone

Auch Telefonnummern werden via regulärem Ausdruck geprüft: `^\+4[139]\d{9,12}$`
Zugelassen sind so alle Telefonnummern aus Deutschland, Österreich und der Schweiz.
Ist der Vergleich erfolgreich, gibt die Methode True zurück, ansonsten False.

6.3.16 cancel

Methode cancel

Wurde eine Konversation mit dem MainConversationHandler gestartet, so kann diese auch manuell wieder beendet werden. So hat ein Nutzer, wann immer er sich im Konversationsfluss befindet, die Möglichkeit den Befehl /cancel abzusetzen. Da dieser Befehl im MainCommandHandler als Fallback definiert ist, kann der Befehl nur abgesetzt werden, solange dieser aktiv ist. Wird die Methode cancel aufgerufen, so wird ein Log-Eintrag über den Abbruch der Konversation abgesetzt. Direkt darauf werden alle Daten des Nutzers aus der Datenbank gelöscht und eine Bestätigung an den Nutzer ausgegeben. Sollte es bei diesem Vorgang zu einem Fehler kommen, so wird der Nutzer auch darüber benachrichtigt und es werden sowohl der Fehler, als auch ein Log-Eintrag in der Konsole ausgegeben. Diese Ausnahme tritt meist dann auf, wenn der Nutzer seine Daten bereits gelöscht und den Bot noch nicht neu gestartet hat - es ihn also in der Datenbank gar nicht gibt oder (selten), falls die SQL-Operation nicht erfolgreich war. Schließlich beendet der Bot die Konversation.

Methode delte

Grundsätzlich handelt es sich bei der Methode delete um fast den gleichen Funktionsumfang, wie bei der Methode cancel. Allerdings ist sie nicht Bestandteil des MainConversationHandlers, sondern in ihrem eigenen Handler definiert und kann somit zu jederzeit über den Befehl /delte aufgerufen werden. So ist dafür gesorgt, dass der Nutzer seine Daten auch löschen kann, wenn die Konversation mit dem Bot aus irgendeinem Grund unterbrochen oder bereits beendet wurde.

6.4 Datenbank

In diesem Abschnitt wird die Funktionsweise des DatabaseConnectors erleutert. Alle Methoden sind ähnlich aufgebaut. Zunächst wird eine Verbindung zur Datenbank geöffnet, es finden diverse Prüfungen statt, eine CRUD-Operaion wird abgesetzt und die Antwort entweder innerhalb der Methode analysiert und ein Boolean oder der Wert aufbereitet und im entsprechenden Format zurückgegeben.

6.4.1 create_db.py

Methode create_db

Es wird versucht, eine Verbindung zur Datenbank (db) aufzubauen. Ist dies erfolgreich, wird der cursor erstellt, über den alle folgenden Operaionen an die Datenbank kommuniziert werden. Ist dies nicht erfolgreich, wird eine neue Datenbank, mit dem vordefinierten Namen erstellt.
Ist die Datenbank verfügbar und wurde eine Verbindung aufgebaut, so wird zunächst geprüft, ob es die Tabelle users schon gibt. Ist dem so, wird die Methode mit einem Commit und dem Schließen der Verbindung zur Datenbank, beendet. Ist dem nicht so, wird die Tabelle für die Benutzer instanziiert. Aufgrund der Simplizität der gespeicherten Daten kommt der Bot mit einer Tabelle aus.

6.4.2 select_db.py

Methode user_search

An usersearch bekommt eine NutzerID übergeben und prüft, ob ein Nutzer in der Datenbank existiert. Falls ja, wird True zurückgegeben - ansonsten False.

Methode get_all_data

An get_all_data wird ebenfalls eine NutzerID übergeben und direkt eine Abfrage für alle Informationen abgesetzt, die es über diesen Nutzer gibt. Die Methode iteriert über alle Einträge, die gefunden wurden und gibt die Daten als Liste und in der Konsole zurück.

Methode get_customers

Die Methode hat keine Input-Parameter, sondern gibt die gesamte Nutzertabelle aus und diese als Liste zurück.

get_value

An get_value werden eine NutzerID sowie eine Spaltenbezeichnung übergeben. So kann ein spezifischer Wert aus der Datenbank abgerufen werden.

6.4.3 insert_value_db.py

Methode insert_update

Um sicherzustellen, dass eine Datebank existiert, bevor man in sie hineinschreibt, wird zunächste die Methode create_db() aufgerufen. Diese kommt schnell zurück, da die Datenbank in den meisten Fällen bereits existiert. Nun wird in einem try/exceptBlock versucht, einen existierenden Datenbankeintrag zu aktualisieren. Das funktioniert meistens, weil der Datensatz für einen Nutzer bereits bei der Eingabe von /start passiert. So kann ein Eintrag bereits von Beginn an immer weiter angereichert werden. Nachdem der Eintrag erfolgreich aktualisiert wurde, gibt es noch einen LogEintrag auf der Konsole.

6.4.4 insert_update_db.py

Methode insert_update

Die Methode funktioniert ähnlich wie die Methode insert_update aus der insert_value_db.py. Sie unterscheidet sich darin, dass sie alle Parameter für den gesamten Datenbankeintrag eines Nutzers übergeben bekommt. So ist es möglich, einen Nutzer auf einmal in die Datenbank einzufügen, ohne den Bot jedes Mal zu durchlaufen. Vor allem zu Testzwecken ist diese Methode hilfreich.

6.4.5 delete_record.py

Methode delete_record

Die Methode bekommt eine Nutzer-ID übergeben und prüft zunächst via der Methode user_search(), ob es den Nutzer mit der angegebenen Nutzer-ID überhaupt gibt. Falls ja, wird in einem try/exceptBlock versucht, alle Informationen eines Nutzers via SQL-Befehl zu löschen.

Methode delete_value

Die Methode ist mit der Methode delete_record fast identisch. Hier wird zusätzlich eine Spaltenbezeichnung übergeben, die es ermöglicht, einen einzelnen Wert zu löschen.

6.5 Kalender

Google Calendar API

Zur Terminfindung ist die Google Calendar API angebunden. [Goo22c] Sie erlaubt es dem aufrufenden System, abzufragen, ob eine Zeitspanne verfügbar ist und Termine zwischen einer festgelegten VeranstalterAdresse und beliebig vielen TeilnehmerAdressen zu erstellen und zu versenden. [Goo22d] Im Folgenden werden die dazu erforderlichen Methoden und Schritte detailliert erklärt.

6.5.1 calendar_manager.py

Der Kalender Manager basiert auf der quickstart.py, die von Google als Starter-Kit für einige gängige Programmiersprachen angeboten wird. Sie stellt den Rahmen für die Authentifizierung gegenüber dem Google Open-Authorization (oauth2) Protokoll und bindet erste Bibliotheken ein. Eine genaue Dokumentation zu Aufbau und Nutzung der quickstart.py findet sich hier: [Goo22b]
Daneben nutzt der Kalender Manager python native Bibliotheken zum Zusammenfügen von Dateipfaden sowie die get_value aus dem Database Connector.

SCOPE

Die API kann auf verschiedene Scopes (z.dt. Umfang oder Reichweite) eingestellt werden. So wird festgelegt, welche Rechte dem Kalender Manager gegenüber der API zur Verfügung stehen und welche Methoden, die die API bietet, genutzt werden können. So wäre bspw. der Scope ".../readonly" verfügbar, über den ein Kalender nur abgefragt, aber keine Termine erstellt werden können.
Der Bot nutzt den umfangreichsten Scope: "https://www.googleapis.com/auth/calendar". Über ihn stehen alle Operationen der API zur Verfügung.

Zugangsdaten

Um sich via OAuth zu authentifizieren, bedarf es folgender Schritte.

1. Erstellung eines Google Accounts
2. Registrierung dieses Accounts als Google Developer Account
3. Anlegen eines Projekts in diesem Google Developer Account
4. Deklaration des Projekts als Testprojekt
5. Eintragen eines Testers (das Gleiche oder ein anderes Google-Konto kann verwendet werden.)
6. Generierung eines Schlüsselpaares zur Authentifizierung
7. Freigabe der Redirect-URI für dieses Schlüsselpaar
8. Generierung und Herunterladen der Zugangsdaten (credentials.json)
9. Installation der quickstart.py im eigenen Repository
10. Anpassung der quickstart.py (Angabe des Pfads zum credentials.json)
11. Ausführen der quickstart.py zur Generierung des lokalen PartnerTokens für die Authentifizierung
12. Anpassung der quickstart.py (Angabe des Pfads zum SicherheitsToken)
13. Transformation der quickstart.py zum Kalender Manager

Sind diese Schritte erfolgreich durchgeführt, so kann mit der eigentlichen Implementierung begonnen werden.

Methode main

Die mainMethode des Kalender Managers authentifiziert sich gegenüber der Google Calendar API und versucht daraufhin, die nächsten zehn Elemente des Kalenders auszugeben. Ist die Authentifizierung nicht erfolgreich, wird ein HTTPError ausgegeben.

Methode authenticate

Die Methode ist eine reduzierte Version der mainMethode. Sie gibt ein s̈ervice¨-Objekt zurück, das genutzt werden kann, um die Methoden der Google Calendar API anzusprechen und auszuführen.

Methode check_availability

Die Methode check_availability nimmt einen Start und einen Endzeitpunkt entgegen und formatiert diese so um, dass sie dem RFC3339Format entsprechen. Das Format setzt sich zusammen wie folgt:

YYYY-MM-DD, Buchstabe T́, HH:MM:SS.ms, Buchstabe Ź´

Beispiel für 10:05 Uhr vormittags am 28.02.2022, koordinierte Weltzeit (UTC):

```
2022-02-28T10:05:00.00Z
```

Weitere Informationen zum RFC3339Format finden sich im offiziellen Standard: [eaK02]

Die beiden Zeitstempel werden in einer Anfrage an die Calendar API eingebaut, an diese übergeben. Die Methode versucht darauf, die Methode freebusy() der API abzufragen. Ist dies erfolgreich, gibt die API eine Antwort im JSON-Format zurück, das Python als Dictionary interpretiert und in mehreren geschachtelten Schleifen auslegen kann. Ist das Feld b̈usy: []l̈eer, so wird True zurückgegeben. Das Zeitfenster ist frei. Ist das Feld nicht leer, gibt es einen Terminkonflikt. Die Methode gibt False zurück.

Erzeugt dieser Vorgang einen Fehler, wird ein HTTP-Error auf der Konsole ausgegeben.

Methode find_slots

Die Methode find_slots sucht nach drei Zeitfenstern innerhalb der vordefinierten Geschäftszeiten und gibt diese, in Form einer Liste zurück.

Die Suche für Termine startet um 08:00 Uhr am kommenden Arbeitstag. Es wird zwischen heute und morgen 08:00 Uhr unterschieden. Ist der Zeitstempel zur Zeit der Ausführung der Methode vor 08:00 Uhr, so wird die Suche heute um 08:00 Uhr begonnen. Ist es bereits nach 08:00 Uhr, so wird die Suche am nächsten Tag begonnen. DAs Resultat ist der Zeitstempel von heute oder morgen 08:00 Uhr, der als Startzeit für die Suche verwendet wird.

Nun wird die Google Calendar API für das erste Zeitfenster abgefragt. Ist das Fenster frei, übernimmt die Methode den Slot und fährt mit der Suche fort. Der

nächste Termin wird drei Tage später gesucht, um mehrere unterschiedliche Termine anbieten zu können.

Ist ein Fenster belegt, so wird der Slot zur nächsten vollen Stunde geprüft und dies solange, bis drei Termine gefunden sind. Neben der Rückgabe als Liste erfolgt eine Ausgabe auf der Konsole.

Wochenenden und Zeiten vor 08:00 sowie nach 18:00 Uhr werden übersprungen, da die Methode check_availability für diese Zeiten immer False zurückgibt.

Methode make_appointment

Der Nutzer hat die Möglichkeit, einen der drei Vorschläge auszuwählen. Dieser Vorschlag wird neben der Nutzer-ID in zweierlei Form an die Methode üerbgeben einmal als String und einmal als KalenderEvent.
Das Event wird inkl. der Kalender-ID an die API via der Methode insert() übergeben und so in den entsprechenden Kalender eingefügt.
Falls bei dieser Operation ein Fehler passiert, wird ein HTTP-Error auf der Konsole ausgegeben.

6.5.2 send_invitation.py

Methode make_appointment

6.5.3

Google Cloud Console: https://console.cloud.google.com/apis/credentials/consent?project=coachin
bot-339115 Calendar API Python Quickstart Guide: https://developers.google.com/calendar/api/qu
Verify own website: https://www.google.com/webmasters/verification/home?hl=en

Tests

Das HauptTestInstrument ist die TelegramApp selbst. Daneben wurden 2 Testskripte zum einfachen Testen verschiedener Teile der App geschrieben. Tests sind getrennt und entsprechend ein oder auskommentiert.

7

Beispiele

Der User startet den Bot via dem Klick auf einen Link, den er auf einer Website findet oder der ihm zugesandt wird.

/start

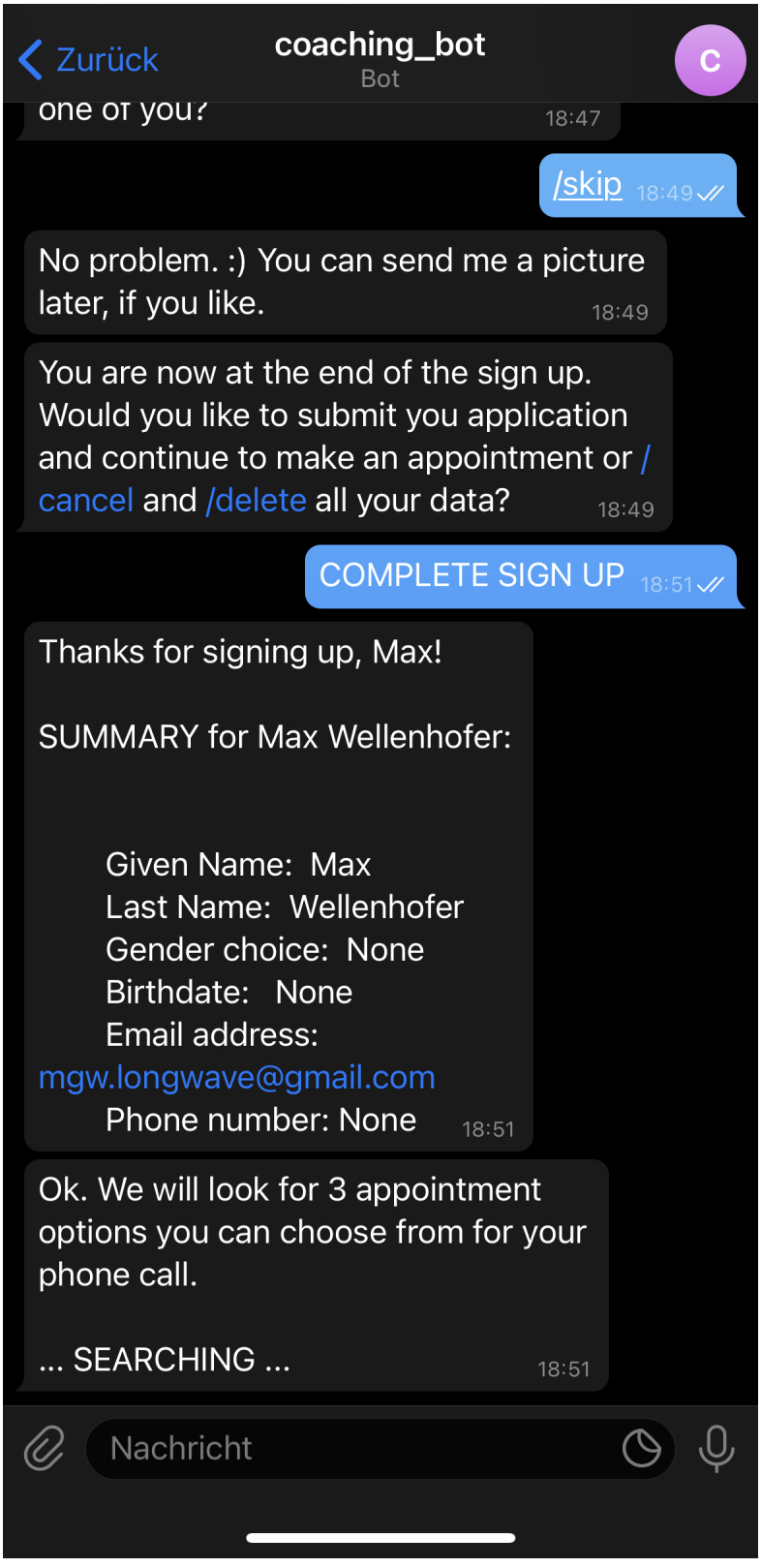

Abbildung 7.1: Bezeichnung der Abbildung

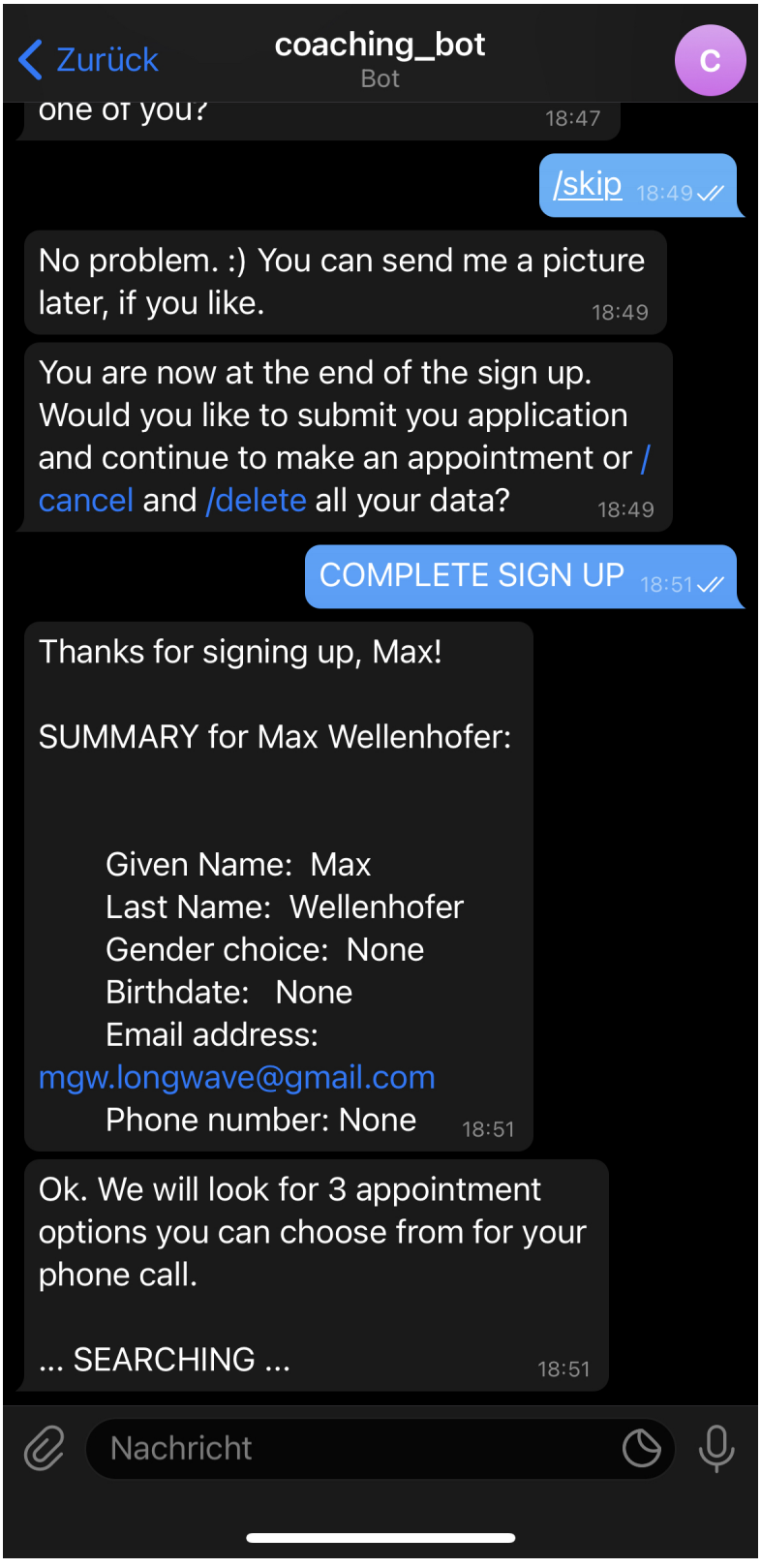

Abbildung 7.2: Bezeichnung der Abbildung

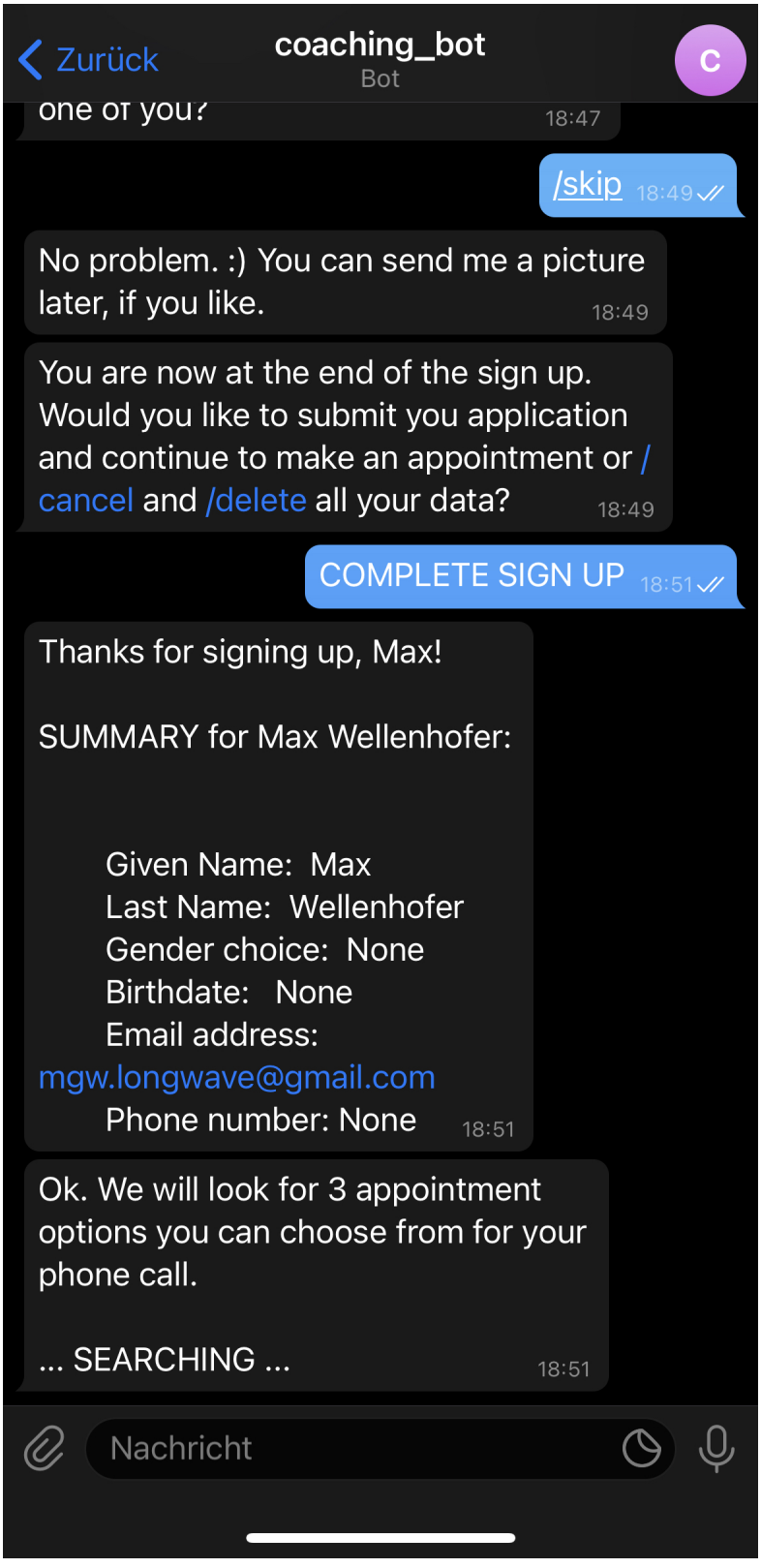

Abbildung 7.3: Bezeichnung der Abbildung

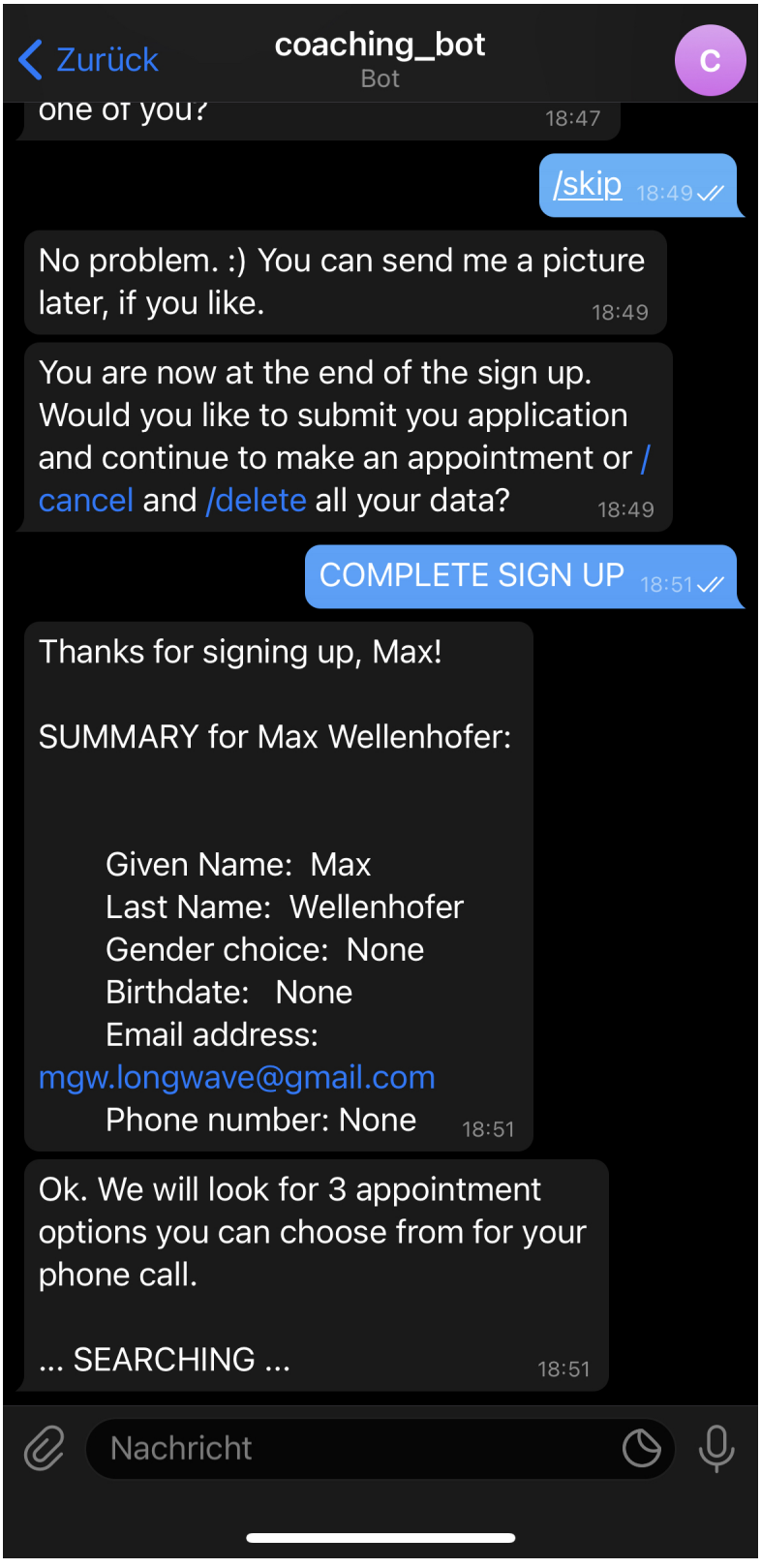

Abbildung 7.4: Bezeichnung der Abbildung

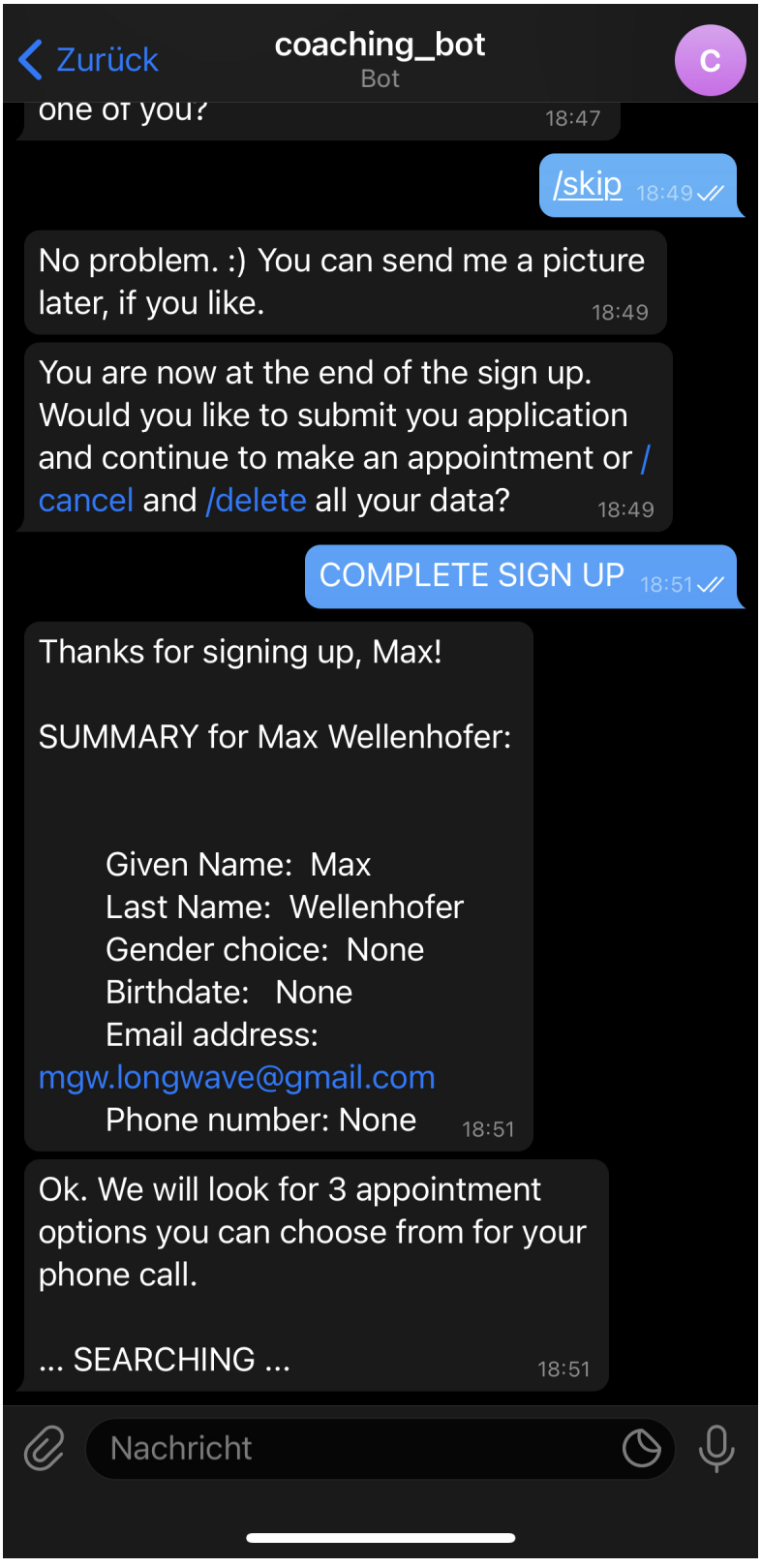

Abbildung 7.5: Bezeichnung der Abbildung

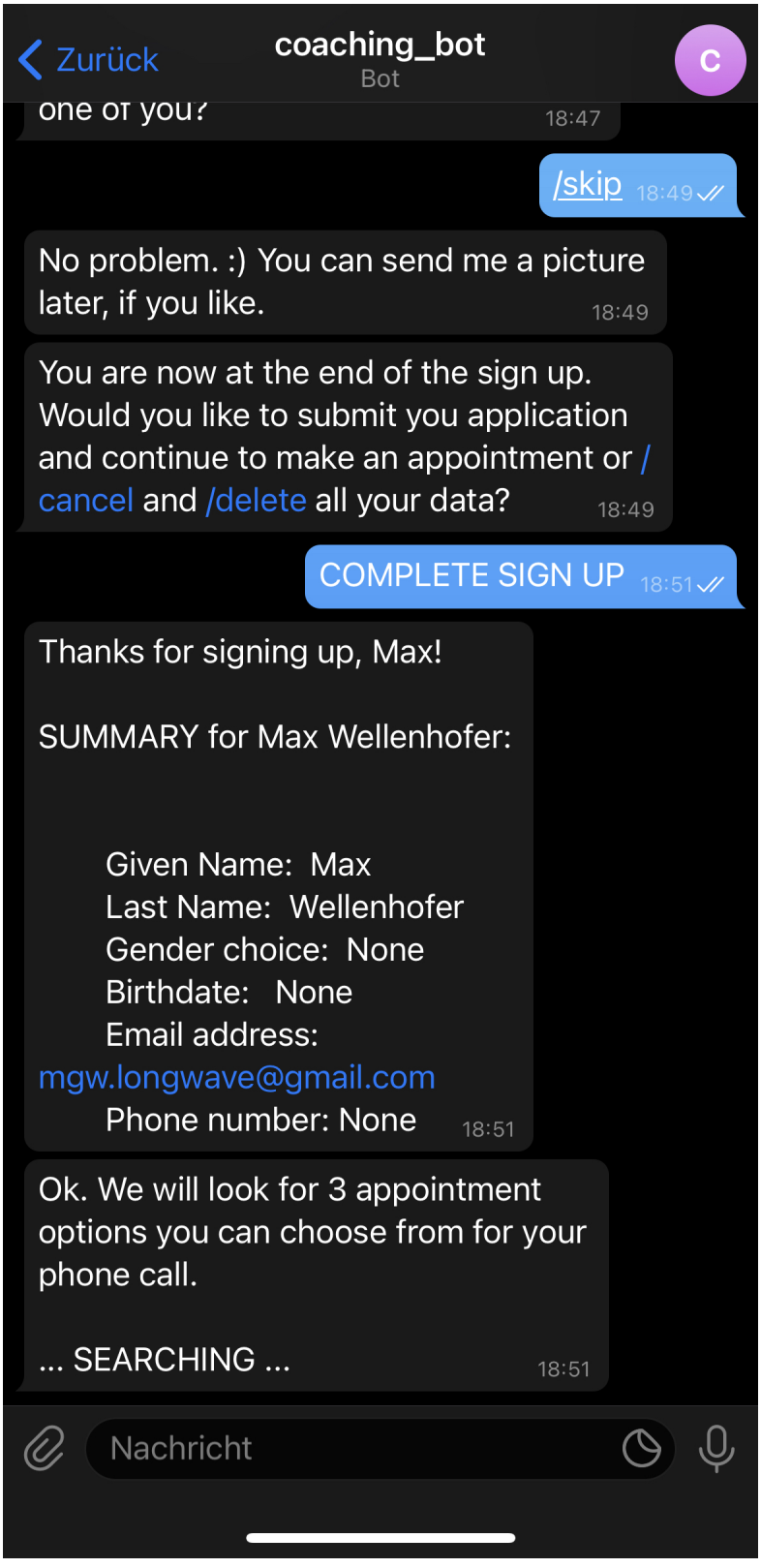

Abbildung 7.6: Bezeichnung der Abbildung

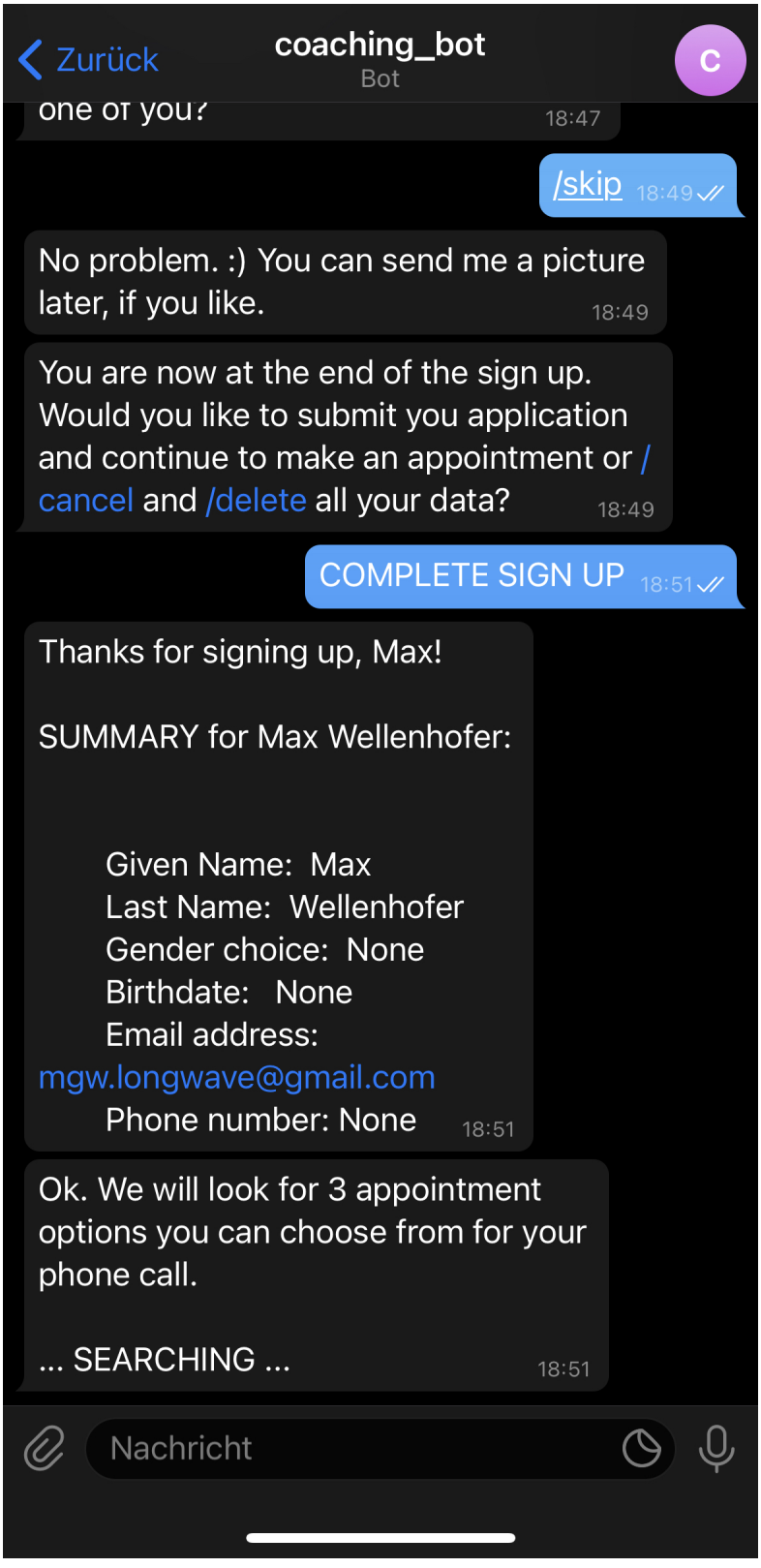

Abbildung 7.7: Bezeichnung der Abbildung

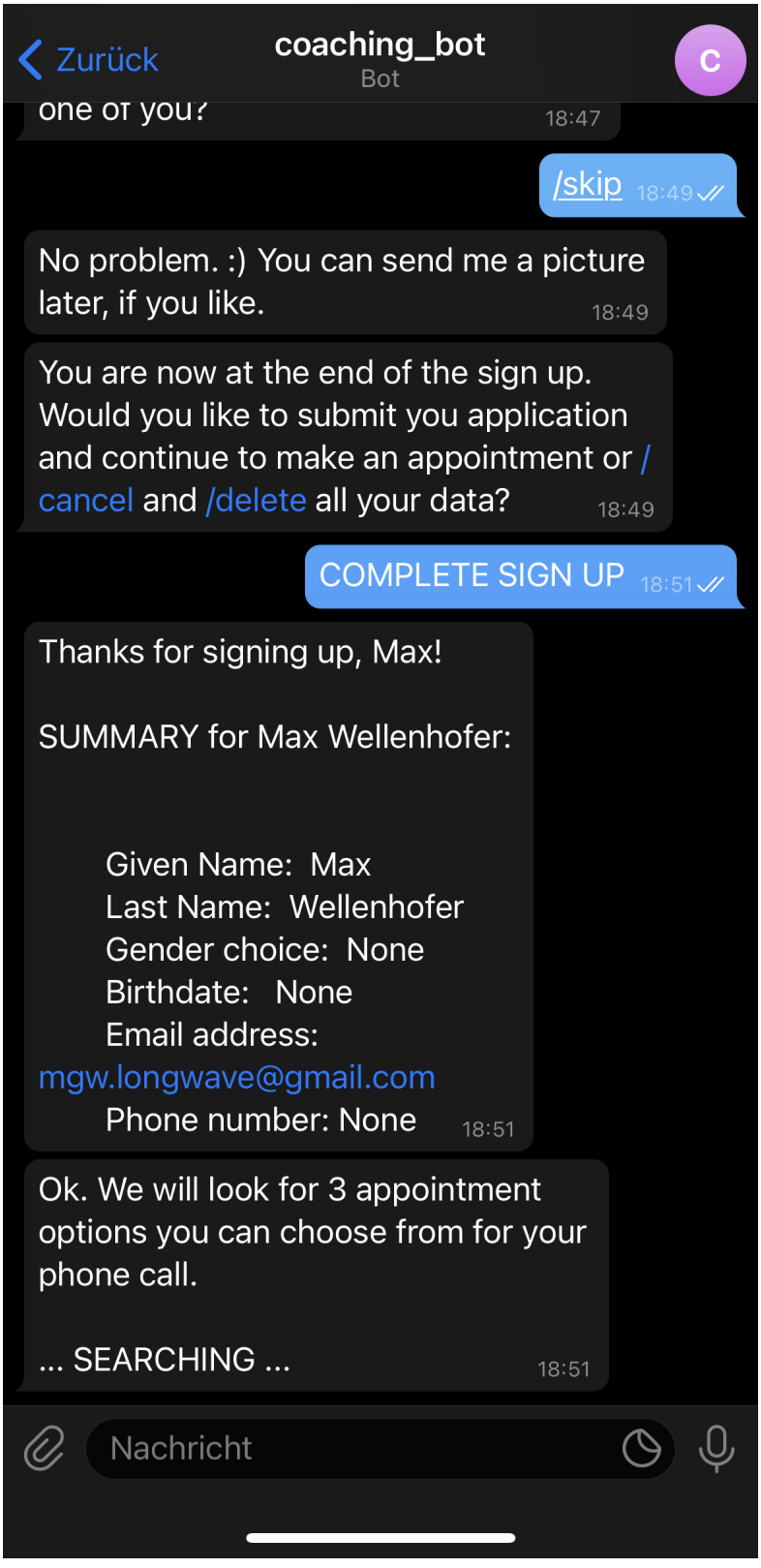

Abbildung 7.8: Bezeichnung der Abbildung

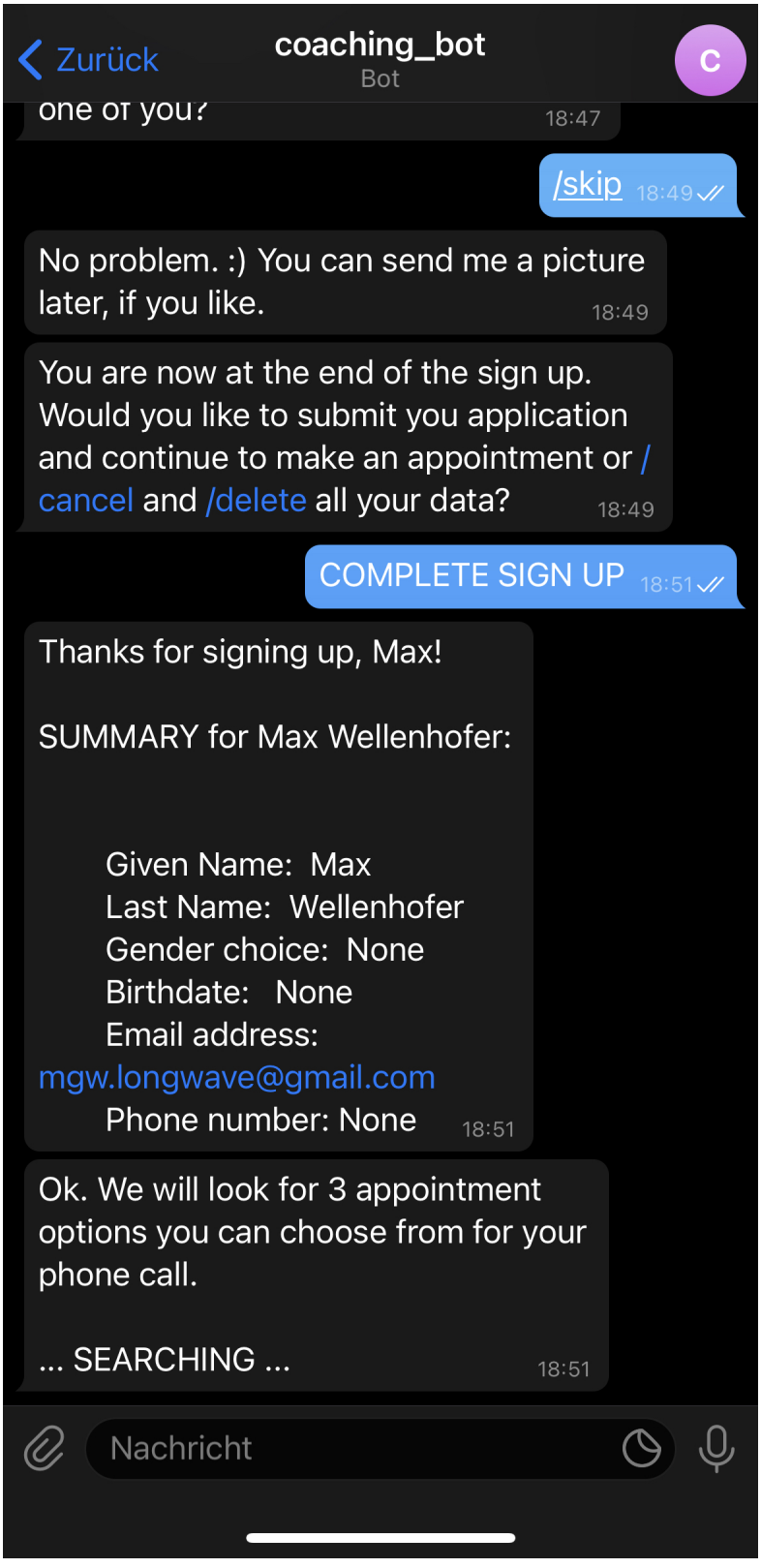

Abbildung 7.9: Bezeichnung der Abbildung

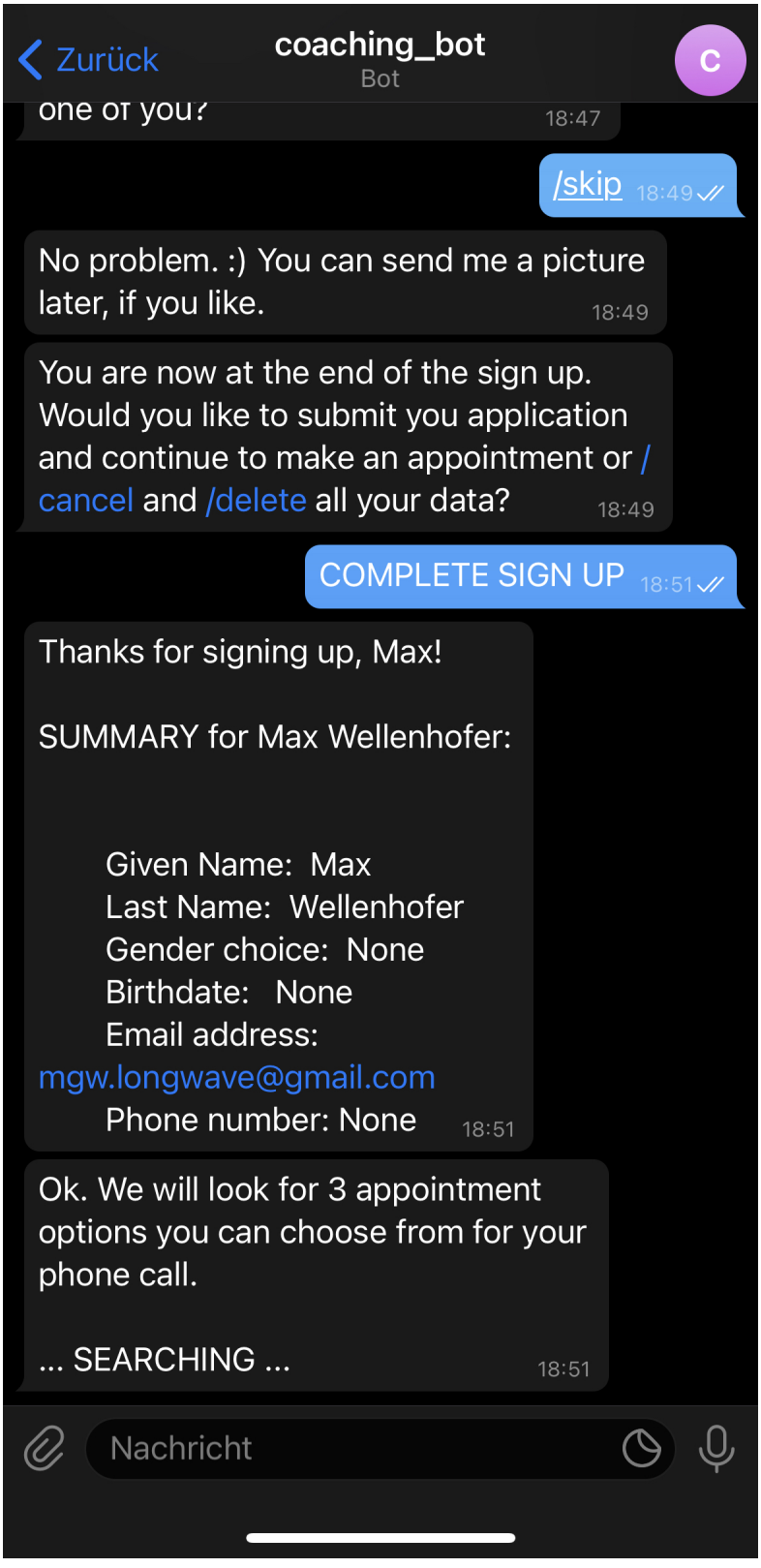

Abbildung 7.10: Bezeichnung der Abbildung

8

Anwendungsszenarien

Der Coaching Bot kann in allerlei Szenarien angewandt werden. Die in den letzten Jahren stark angewachsene Zahl an Personal Coaches kann den Bot mit basalen Programmierfähigkeiten an die eigenen Bedürfnisse anpassen. Vor allem für Nebenerwerbstätige Coaches mit einem kleinen Kundenportfolio stellt der Coaching Bot eine einfache Möglichkeit dar, Neukunden onzuboarden. Der Prozess ist unkompliziert, unverbindlich und einfach zu adaptieren.

Ausbaupotenzial

Sollte der Coaching Bot über die ersten Monate vielversprechende Ergebnisse liefern, sind folgende Ausbaustufen geplant:

1. Verteilung und Verlagerung in die Cloud für konstante und hohe Verfügbarkeit 2. Skripten und Automatisierung weiterer Coaching-Stufen. i.e. könnte die erste Session, die oft ähnlich abläuft auch vom Bot abgehandelt werden. 3. Ton-Aufnahmen für das Biography-Modul

9

Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal kurz zusammengefasst werden. Insbesondere sollen die wesentlichen Ergebnisse Ihrer Arbeit herausgehoben werden. Erfahrungen, die z.B. Benutzer mit der Mensch-Maschine-Schnittstelle gemacht haben oder Ergebnisse von Leistungsmessungen sollen an dieser Stelle präsentiert werden. Sie können in diesem Kapitel auch die Ergebnisse oder das Arbeitsumfeld Ihrer Arbeit kritisch bewerten. Wünschenswerte Erweiterungen sollen als Hinweise auf weiterführende Arbeiten erwähnt werden.

Nochmal kurz sagen, was Sie gemacht haben - am besten die Ziele der Arbeit aus 1.2. nochmals nennen und kurz erklären, wie sie das in Ihrem System realisiert haben. (So was wie ein "management summary)

Literaturverzeichnis

ea21a. AL., TOLEDO ET.: *python-telegram-bot*. https://github.com/python-telegram-bot/python-telegram-bot/blob/master/examples/conversationbot.py, 2021.

ea21b. AL., TOLEDO ET.: *Telegram API Documentation*. https://python-telegram-bot.readthedocs.io/en/stable/index.html#, 2021.

eA21c. AL., TOLEDO ET.: *telegram.ext package*. https://python-telegram-bot.readthedocs.io/en/stable/telegram.ext.html, 2021.

eaK02. KLYNE ET. AL.: *Date and Time on the Internet: Timestamps*. 2002.

Goo22a. GOOGLE LLC: *Google APIs Explorer overview*. https://developers.google.com/explorer-help/, 2022.

Goo22b. GOOGLE LLC: *Google Calendar API - Python Quickstart*. https://developers.google.com/calendar/api/quickstart/python, 2022.

Goo22c. GOOGLE LLC: *Google Cloud Platform*. https://console.cloud.google.com/, 2022.

Goo22d. GOOGLE LLC: *GoogleCalendarAPI*. https://googleapis.github.io/google-api-python-client/docs/dyn/calendar_v3.html, 2022.

IBM20. IBM: *Application Programming Interface (API)*. https://www.ibm.com/cloud/learn/api, 2020.

Kre21. KREMMING, KATHARINA: *Telegram Messenger – Alles was Du wissen musst!* https://www.messengerpeople.com/de/messaging-apps-brands-der-telegram-messenger/, 2021.

Meh22. MEHNER, MATTHIAS: *Nutzerzahlen Messenger Apps Deutschland und Weltweit*. https://www.messengerpeople.com/de/weltweite-nutzer-statistik-fuer-whatsapp-wechat-und-andere-messenger/, 2022.

Pal22. PALLETS: *Flask Documentation*. https://flask.palletsprojects.com/en/2.0.x/, 2022.

Pyt. PYTHON SOFTWARE FOUNDATION: *sqlite3 — DB-API 2.0 interface for SQLite databases*. https://docs.python.org/3/library/sqlite3.html.

Pyt21a. PYTHON SOFTWARE FOUNDATION: *datetime — Basic date and time types*. https://docs.python.org/3.8/library/datetime.html, 2021.

Pyt21b. PYTHON SOFTWARE FOUNDATION: *email.mime: Creating email and MIME objects from scratch*. https://docs.python.org/3.8/library/email.mime.html, 2021.

Pyt21c. Python Software Foundation: *pipenv*. https://pypi.org/project/pipenv/, 2021.

Pyt21d. Python Software Foundation: *Python 3.8.6*. https://docs.python.org/3.8/, 2021.

Pyt21e. Python Software Foundation: *smtplib — SMTP protocol client*. https://docs.python.org/3.8/library/smtplib.html#module-smtplib, 2021.

Tel21a. Telegram Messenger Inc.: *Bots: An introduction for developers*. https://core.telegram.org/bots, 2021.

Tel21b. Telegram Messenger Inc.: *Telegram App*. https://telegram.org/, 2021.

The21. The SQLite Consortium: *SQLite*. https://sqlite.org, 2021.

W3C19. W3CSchools: *HTML Documentation*. https://www.w3schools.com/html/html_intro.asp, 2019.

Wel21. Wellenhofer, Max: *The Coaching Bot Repository*. https://github.com/mwel/coaching_bot, 2021.

appendix

A

Glossar

API	Anwendungsprogrammierschnittstellen (APIs) vereinfachen Softwareentwicklung und -innovation, indem sie Anwendungen den einfachen und sicheren Austausch von Daten und Funktionen ermöglichen. [IBM20]
Bot	hier: ChatBot - Computerprogramm, das mit einem Nutzer interagieren und vordefinierte Aufgaben selbstständig erledigen kann.
PoC	Proof of Concept: z. dt. Machbarkeitsstudie
DACH	Deutschland, Österreich, Schweiz bzw. deutschsprachige Region Europas
Vendor Lock-In	hier: Beschränkung der Anwendbarkeit eines Systems auf einen Anbieter - in diesem Fall 'Telegram'
GUI	Graphical User Interface: z.dt. Benutzeroberfläche
CSS	Cascading Style Sheets: Stylesheet-Sprache, die zur optimsch ansehnlicheren Aufbereitung von Benutzeroberflächen genutzt wird
Geo-Fencing	Virtuelle Abgrenzung eines festgelegten Gebiets oder einer Region
CRUD	CREATE, READ, UPDATE, DELETE: Standard-Operationen, die auf einer Datenbank durchgeführt werden
SQL	Search Query Language: Sprache zur Definition von Datenstrukturen in relationalen Datenbanken

B

Selbstständigkeitserklärung

☐ Diese Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

☐ Diese Arbeit wurde als Gruppenarbeit angefertigt. Meinen Anteil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser:

Meine eigene Leistung ist:

Datum

Unterschrift der Kandidatin/des Kandidaten