Universität Potsdam

Department Linguistik

Dokumentation des Skills “Mensa-Auskunft”

Projektbericht über eine Funktionserweiterung von Amazons digitalem Assistenten *Alexa*

Modul: ANW A / AM 4 – Anwendungen der CL
Seminar: Praktische Dialogmodellierung: Ein Dialogsystem erstellen
Semester: Sommersemester 2019
Dozent: Prof. Dr. David Schlangen

Autoren: Bogdan Kostić, MATRIKELNUMMER
Maria Lomaeva, MATRIKELNUMMER
Monique Noss, MATRIKELNUMMER
Olha Zolotarenko, MATRIKELNUMMER

30. Oktober 2019

Inhaltsverzeichnis

1 Zusammenfassung

...

2 Projektziele und Funktionen

Im Nachfolgenden werden die erreichten Projektziele und Anforderungen erläutert. Darüber hinaus werden die Funktionen des Skills anhand von Beispieldialogen gezeigt.

2.1 Anwendungsbereich und Zielgruppe

Bei einem Mensa-Skill, dessen Hauptaufgabe es ist, über den Essensplan einer Mensa zu informieren, sind vor allem Studierende, aber auch Dozierende die Zielgruppe. Eine kleine, aber keinesfalls unwichtige Teilgruppe dieser doch recht großen Zielgruppe sind blinde Studierende und Beschäftigte der Universitäten. Dieser Skill ermöglicht es Menschen mit Sehbehinderung leichter an Informationen über das Essensangebot an deutschen Hochschulen zu kommen und macht damit die deutsche Hochschullandschaft noch ein wenig barriefreier. Eine weitere kleine Zielgruppe dieses Skills könnten Beschäftigte eines Studienakkreditierungsinstituts sein, die oft durch ganz Deutschland reisen und viel Zeit an den unterschiedlichsten Hochschulen verbringen. Dementsprechend könnten diese Beschäftigte an den Speiseplänen vieler verschiedener Universitäten und Fachhochschulen interessiert sein. Da wir versucht haben, die ganze Bandbreite der OpenMensa API zu nutzen und somit Informationen für 490 verschiedene Mensen bereitstellen können, ist dieser Skill dazu in der Lage, diesem Interesse entgegenzukommen.
Das Informationsbedürfnis dieser Zielgruppen ist es einerseits, eine grobe Übersicht über das Angebot zu erhalten, andererseits aber auch nach bestimmten Zutaten gefilterte Ergebnisse zu erhalten, falls Allergien oder besondere Ernährungsweisen vorliegen (z.B. Veganismus). Auch der Preis eines bestimmten Gerichtes könnte für eine finale Auswahl entscheidend sein, da das Budget eines Studierenden selten großzügig ausfällt. Diese und weitere verschiedene Funktionen wurden eingebaut, um die in der OpenMensa API angebotenen Daten ausgiebig zu nutzen.
Dazu gehören auch die folgenden Lokationsfunktionen:

- Adresse von Mensen
- Koordinaten, um die nächste Mensa zu finden
- Mensen nach Stadt finden

Neben den oben genannten Funktionen, die durch selbsterstellte Intents des Skills umgesetzt wurden, bietet Amazon den Skill-Entwicklern an, dynamische, auf den Benutzer zugeschnittene Funktionen einzubauen. Ein Beispiel hierfür sind die Persist Attributes"(= "bleibende Attribute"). Persist Attributes dienen dazu, bestimmte Daten in einer Session zu speichern, damit sie in der nächsten wieder abgerufen werden können. So müsste ein Benutzer nicht bei jedem Launch mit erwähnen, für welche Mensa ein Essensplan ausgegeben werden soll. Während die Implementierung und Einbettung über AWS DynamoDB einfach vorgenommen werden kann, könnten diese beim Entwickler Kosten verursachen, weshalb wir in diesem Skill auf die Benutzung dieser Attribute verzichtet haben. Auch könnten User Accounts angelegt werden, um persönliche Vorlieben des Users speichern zu können.
Eine weitere mögliche Verbesserung haben wir während der zweiten Projektphase, also nach der ersten Vorstellung der Projekte, umsetzen können. Es handelt sich hierbei um syntaktisches Parsing der Gerichte, um nur den Kopf der "Gerichtsphrase" auszugeben und somit die Antworten des Skills zu kürzen, um den Nutzer nicht mit Informationen zu überhäufen. Aus demselben Grund werden nun immer nur vier Gerichte in einem Prompt vorgelesen. Möchte der Nutzer weitere Gerichte erfahren, kann einfach "weiter"gescrollt werden. Mit einem Intent für Details können dagegen Details zu einem Gericht erfragt werden.
Für uns war es wichtig, die gesamte Breite der API zu nutzen, um dem Nutzer so viele Funktionen wie möglich zu bieten. Nichtsdestotrotz findet die In-App-Kommunikation schnell ein Ende, wenn der Benutzer ein mögliches Ziel erreicht hat oder ein Fehler aufgetreten ist. Der Skill kann dann in einem One-Shot erneut gelaunched werden.

2.1.1 Essensplan

Gibt den Essensplan einer bestimmten Mensa für ein bestimmtes Datum aus. Optional kann eine Zutat angegeben werden, die enthalten sein soll.
Beispieläußerungen:

> "lies mir den plan für {date} vor"
> "gibt es {date} {ingredient} gerichte in der {mensa_name}"
> "was gibt's in der mensa"

2.1.1.1 Gerichte ohne Zutat

Sucht Gerichte in einer bestimmten Mensa für ein bestimmtes Datum ohne bis zu zwei Zutaten.

Beispieläußerungen:

> “suche für {date} {synonyms_gericht} ohne {ingredient} in {mensa_name}”
> “gibt es {date} {synonyms_gericht} ohne {ingredient}”
> “nach {synonyms_gericht} ohne {ingredient} bitte”

2.1.1.2 Gerichte mit Zutat

Sucht Gerichte in einer bestimmten Mensa für ein bestimmtes Datum mit bis zu zwei Zutaten.
Beispieläußerungen:

> “suche für {date} {synonyms_gericht} mit {ingredient} in {mensa_name}”
> “gibt es {date} {synonyms_gericht} mit {ingredient}”
> “nach {synonyms_gericht} mit {ingredient} bitte”

2.1.2 Preis der Gerichte

Der Skill gibt den Preis für ein Gericht zurück.
Beispieläußerungen:

> “preis für nummer {number}”
> “wie viel kostet das {number} für {user_group}”
> “wie teuer ist das {number} gericht”

2.1.3 Adresse der Mensa

Die Adresse einer Mensa wird vorgelesen.
Beispieläußerungen:

> “zeige mir die adresse der {mensa_name}”
> “standort {mensa_name}”
> “adresse {mensa_name}”

2.1.4 Mensas in einer Stadt

Listet die Mensas in einer genannten Stadt auf.
Beispieläußerungen:

“welche mensas gibt es in {city}”
“suche mensas in {city}”
“gibt es mensas in {city}”

2.1.5 Mensa in der Nähe

Findet die Mensa, die dem Benutzerstandort am nächsten ist.
Beispieläußerungen:

“wo ist die nächste mensa”
“ich bin hungrig”
“welche mensa ist in der nähe”

2.2 Szenarien und Beispieldialoge

Im nachfolgenden Absatz werden Beispieldialoge für jeden Intent aufgelistet, die Skillerfolge und -misserfolge darstellen sollen, um die Funktionalität des Skills zu verdeutlichen. Diese werden jeweils anhand von spezifischen Szenarien veranschaulicht und erklärt.

2.2.1 Beispieldialoge: Essensplan und Preise erfahren

Die am häufigsten genutzte Funktion eines Mensa-Skills wird voraussichtlich das Abfragen des Essensplans für einen bestimmten Tag in der eigenen Mensa sein. Hierbei kann der Nutzer Zutaten inkludieren, um die Auswahl von Anfang an einzuschränken. Wenn eine Auswahl getroffen wurde, wird der Benutzer gefragt, ob er Details oder den Preis eines der Gerichte erfahren möchte. Der Benutzer kann an dieser Stelle auch verneinen, woraufhin der Skill geschlossen wird.
Szenario 1: Benutzer möchte den Essensplan für eine bestimmte Mensa, für einen bestimmten Tag erfragen.
Wann? Jederzeit, wann immer der User in der Mensa essen möchte.

Situation 1.1:
Morgen ist ein Werktag.

U: Alexa, öffne Mensaauskunft.

A: Willkommen bei der Mensaauskunft! Wenn du Hilfe bei der Bedienung brauchst, sag bitte HILFE. Was möchtest du wissen?

U: Was gibt es morgen in der Mensa?

A: In welcher Mensa möchtest du essen?

U: Mensa Golm.

A: Es gibt 5 Gerichte zur Auswahl: 1. Eintopf aus Kichererbsen, Linsen, Gemüse, Kartoffeln, 2. Eintopf aus Kichererbsen, Linsen, Gemüse,Kartoffeln, 3. Gulasch und 4. Hähnchenbrust „Piccata Milanese“. Möchtest du mehr Gerichte hören oder Details?

U: Mehr!

A: 5. Griechisches Kartoffelgericht. Möchtest du Details zu einem dieser Gerichte erfahren? Sag zum Beispiel: Details. oder: Wie viel kostet Gericht Nummer 2 für Studenten.

U: Wie viel kostet Gericht Nummer 2 für Studenten?

A: Das Gericht Eintopf aus Kichererbsen, Linsen, Gemüse,Kartoffeln und Rindfleisch kostet 1,6 Euro für Studenten.

— SKILL SCHLIEßT SICH —

Situation 1.2:

Morgen ist Wochenende oder Feiertag.

U: Alexa, frag Mensaauskunft, was gibt es morgen in der Mensa Golm?

A: Sorry, für den ausgewählten Tag 2019-10-20 gibt es leider keinen Essensplan für mensa golm.

— SKILL SCHLIEßT SICH —

Situation 1.3:

Die Mensa wurde nicht richtig verstanden (ASR Error) oder ist nicht in der Datenbank.

U: Alexa, frag Mensaauskunft, was gibt es morgen in der Mensa Golm?

A: Sorry, Essenspläne für mensa golum habe ich leider nicht im Angebot.

— SKILL SCHLIEßT SICH —

Situation 1.4:

Der Benutzer sucht nach Gerichten mit einer bestimmten Zutat.

– Dialog 1.4.1 –

U: Alexa, frag Mensaauskunft nach veganen Gerichten.
A: Für wann?
U: Morgen.
A: In welcher Mensa möchtest du essen?
U: Mensa Golm.
A: Es gibt 2 Vegan Gerichte zur Auswahl: 1. Vegane Knusperkroketten und 2. Glasnudel-Bowl. Möchtest du Details zu einem dieser Gerichte erfahren? Sag zum Beispiel: Details. oder: Wie viel kostet Gericht Nummer 2 für Studenten.
U: Stopp.
A: Guten Hunger!
— SKILL SCHLIEßT SICH —

– Dialog 1.4.2 –

U: Alexa, frag Mensaauskunft nach Gerichten mit Sellerie.
A: Für wann?
U: Morgen.
A: Leider gibt es keine passenden Gerichte mit sellerie. Kann ich sonst noch helfen?
U: Nein.
A: Okay, tschüss!
— SKILL SCHLIEßT SICH —

2.2.2 Beispieldialoge: Die nächste Mensa finden

Ein wenig komplexer ist die Funktionalität, eine Mensa in der Nähe des Standortes des Benutzers zu finden. Aus dem Intent können daher daher unterschiedliche Reaktionen hervorgehen, je nach Status der Standortfreigabe für das benutzte Gerät und den Skill selbst.

Szenario 2: Benutzer möchte eine Mensa in seiner Nähe finden und befindet sich im Ortsteil Golm in Potsdam.
Wann? Höchstwahrscheinlich nur ein Mal, wenn der User wissen möchte, welche Mensa die nächste in der Umgebung ist.

Situation 2.1:
Der User hat den Gerätestandort nicht freigegeben.

U: Alexa, frag Mensaauskunft, wo die nächste Mensa ist.

A: Um die nächste Mensa zu finden, benötige ich Deinen Standort. Bitte öffne die Alexa-App, um deinen Standort mit mir zu teilen.

— SKILL SCHLIEßT SICH —

Situation 2.2:
Der User hat den Gerätestandort freigegeben.

U: Alexa, frag Mensaauskunft, wo die nächste Mensa ist.

A: Die nächste Mensa ist Mensa Golm in Karl-Liebknecht-Str. 24/25, 14476 Potsdam OT Golm.

— SKILL SCHLIEßT SICH —

2.2.3 Beispieldialoge: Die Adresse einer Mensa erfahren

Die Adresse der Mensa kann erneut erfragt werden oder es kann auch nach Adressen anderer Mensen gesucht werden.

Szenario 3: Benutzer möchte die Adresse einer Mensa erfahren.
Situation 3.1:
Die Adresse der Mensa kann gefunden werden, sie ist also in der Datenbank vorhanden.

U: Alexa, frag Mensaauskunft nach der Adresse der Mensa Golm.

A: Die Adresse der mensa golm lautet Karl-Liebknecht-Str. 24/25, 14476 Potsdam OT Golm.

— SKILL SCHLIEßT SICH —

Situation 3.2:
Die Adresse der Mensa kann nicht gefunden werden, sie ist also nicht in der Datenbank vorhanden. Möglich ist auch, dass ASR den User falsch verstanden hat.

U: Alexa, frag Mensaauskunft nach der Adresse der Mensa Golm.

A: Die Adresse der angefragten Mensa mensa ball um konnte leider nicht gefunden werden.

— SKILL SCHLIEßT SICH —

Szenario 4: Benutzer möchte erfahren, welche Mensen es in einer bestimmten Stadt gibt.

Situation 4.1:

Der Nutzer fragt nach Mensen in einer deutschen Stadt, zum Beispiel Potsdam.

U: Alexa, frag Mensaauskunft, welche Mensas gibt es in Potsdam?

A: Es gibt die folgenden Mensas in potsdam: Mensa Am Neuen Palais, Mensa Kiepenheuerallee, Mensa Golm, Mensa Griebnitzsee, Bistro Tasty Studio Babelsberg, Ulf's Café (HPI Cafeteria), Erfrischungshalle.

— SKILL SCHLIEßT SICH —

Situation 4.2:

Der Nutzer fragt nach Mensen in einer Stadt, die nicht in Deutschland liegt, zum Beispiel Tokyo.

U: Alexa, frag Mensaauskunft, welche Mensas gibt es in Tokyo?

A: Leider keine Mensas in Tokyo gefunden. Du kannst eine andere Stadt in Deutschland auswählen.

— SKILL SCHLIEßT SICH —

3 Projektorganisation

3.1 TeilnehmerInnen & Aufgabenverteilung

Das Team setzt sich aus vier Personen zusammen: Monique Noss, Bogdan Kostić, Maria Lomaeva und Olha Zolotarenko. Jedem Teammitglied wurde ein Aufgabenbereich zugeteilt, für den er oder sie zuständig war. Die Aufteilung der Intents im Team war wie folgt:

- Bogdan: IngredientIntent, GetNearestMensaIntent, WithoutIntent, ListDishesIntent(+WithoutIntent), NextIntent
- Olha: AddressIntent, Chunking
- Maria: ListMensasIntent, Chunking
- Monique: ListDishesIntent, PriceIntent, DetailsIntent, NextIntent, NoIntent

Neben dem Erstellen des Codes für das Backend der Intents sollte sich auch um die jeweiligen *sample utterances*, das Testen sowie das Dokumentieren des

jeweiligen Intents gekümmert werden. Die Koordination der verschiedenen Bestandteile des Skills im Code hat Monique Noss übernommen.

3.2 Planungsdokumente & Milestones

Die Planungsdokumente zusammen mit den ursprünglichen Entwicklungsideen (`ideas.md`) und den dazugehörigen Milestones sind unter `mensa-skill/material/` zu finden.
Die erste Version des Skills und der Intents wurde in der Datei `meeting1.md` dokumentiert. Diese Version wurde in der Vorstellung der Projekte am Ende des Sommersemesters 2019 getestet und anschließend präsentiert. Die wichtigsten Verbesserungsvorschläge, die dabei resultiert sind, waren dabei die folgenden:

- Aufzählungslisten verkürzen (siehe dazu auch `chunking.md`)
- Die Möglichkeit, nach mehr als einer Zutat im Gericht zu suchen
- Nach einer Mensa in der Nähe fragen zu können, statt nur alle Mensen in der Stadt aufzulisten
- Mehr natürliche *sample utterances* einfügen

Die zweite Version des Skills Mensa-Auskunft wurde nach einer ausführlichen Besprechung der nötigen Verbesserungen implementiert, welche oben kurz zusammengefasst wurden. Näheres kann man der Datei `meeting2.md` entnehmen. Schließlich wurde der Code organisiert und kommentiert, um die Lesbarkeit zu erhöhen. So entstand auch eine separate Datei `lambda_utility.py`, die alle Hilfsfunktionen für `lambda_function.py` beinhaltet.

4 Entwurf des Systems, Dokumentation

4.1 Entwicklungsumgebung

Abbildung 1 zeigt, mit welcher Software der Skill erstellt wurde. Der Skill läuft unter der Pythonversion 3.6 und importiert einige Bibliotheken, die einerseits native Python-Bibliotheken sind und andererseits von dem Alexa Skills Kit zur Verfügung gestellt wurden oder installiert werden müssen. Diese Bibliotheken müssen im selben Ordner der `lambda_function.py` liegen, damit diese auch von AWS Lambda eingelesen werden können. Für die Installation wurde `pip` und das Kommando `pip3 install -r requirements.txt -t .` benutzt. Zu den Requirements gehören:

- ask-sdk

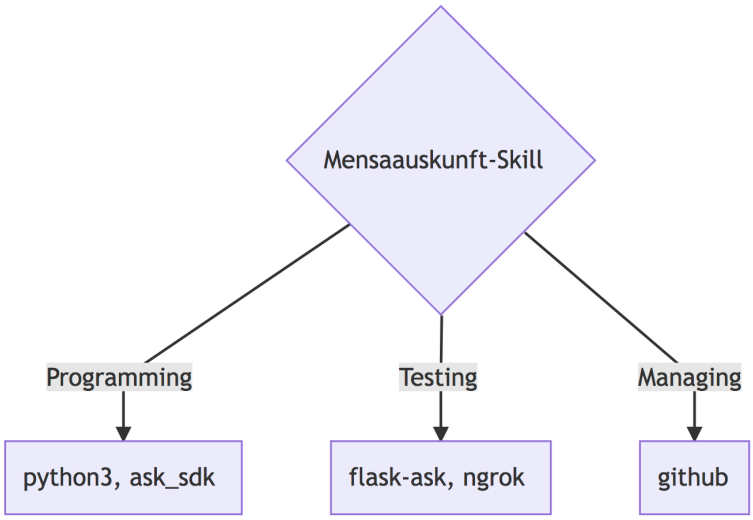

Abbildung 1: Benutzte Software während der Entwicklung des Skills

- flask
- flask-ask-sdk
- haversine

`ask-sdk` wird benötigt, um die Kommunikation mit Alexa herzustellen. Das ganze Skill-Building basiert auf den Bibliotheken dieser Software, dazu gehören die Intent-Klassen und Funktionen sowie das Verarbeiten des Sprach-Outputs (Prompts).
`flask` und `flask-ask-sdk` wird zum Aufsetzen eines lokalen Servers verwendet. Wenn der Server gestartet ist, kann das Alexa Skills Kit mit dem auf dem Computer gespeicherten Code kommunizieren, wenn dieser lokal ausgeführt wird. Dazu wird außerdem das Tool `ngrok` benötigt, um den Port des Computers nach außen zu leiten. Diese Tools waren essenziell für das Testen unseres Skills.
`haversine` ist eine Bibliothek, die die Distanz zwischen zwei Punkten auf der Erde mithilfe des Breiten- und Längengrades kalkuliert. Diese wird benötigt, um die nächste Mensa in der Nähe des Benutzerstandortes zu finden.

4.2 Tests

Getestet wurde der Skill mithilfe von *flask* und *ngrok* lokal auf unseren eigenen Rechnern, um Status- und Fehlermeldungen angezeigt bekommen zu können, sodass sie direkt nachvollzogen werden konnten.

4.3 Intents

Dieser Skill verwendet insgesamt 13 Intents, davon sind sechs Intents *Custom Intents* sowie sieben Intents *Built-In Intents*, die von Amazon zur Verfügung gestellt werden und teilweise über eigene *sample utterances* erweitert wurden. Außerdem benutzt der Skill acht verschiedene Slot-Types, von denen vier zu den *Amazon Built-In Slots* gehören.

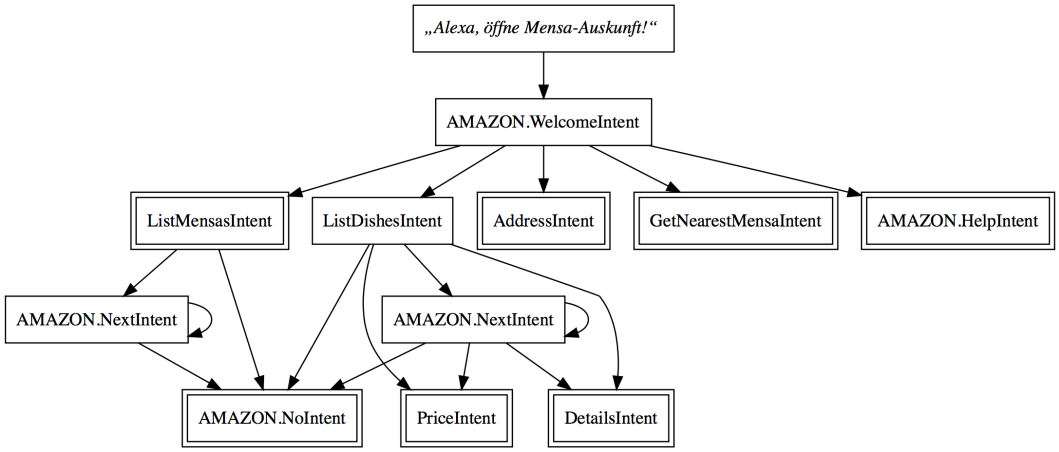

Abbildung 2: Intentabfolge von gelungenen Dialogen mit dem Skill

Abbildung 2 zeigt die Intentabfolge von gelungenen Dialogen mit dem Skill. Der Intent `AMAZON.WelcomeIntent` kann mit einer One-Shot-Äußerung des Benutzers übersprungen werden. Weicht der Nutzer von dieser Intentreihenfolge ab, so erhält er einen Hinweis, dass zunächst eine entsprechende Suche gestartet werden muss.
Im Anhang dieses Berichts befindet sich eine Dokumentation des Backends des Skills, aus der die verschiedenen Intents und Slot-Types entnommen werden können. Diese Dokumentation wurde mithilfe des Tools *Sphinx* aus den Docstrings extrahiert.

5 Projektabschluss, Evaluation

- Versuchsanordnung: VPs, Material, Methode (Fragebogen, Erfolg)
- Auswertung
- Ausblick, Verbesserungsmöglichkeiten

Anhang: Dokumentation des Backends

KAPITEL 1

Lambda Function

lambda_function.**details_intent_handler**(*handler_input*)

Der Intent listet die Details zu einem bestimmten Gericht auf.

(Alle verfügbaren Informationen der OpenMensa API.)

Dazu gehören:

- Der vollständige Titel des Gerichts
- Die Preise für Studenten, Angestellte und Andere
- Die Kategorie des Gerichts
- Die zusätzlichen Notes

Der Benutzeranfrage und den bereitgestellten Slot Values werden folgende Daten entnommen:

- Gerichtnummer

Über die Session-Attributes müssen folgende Daten abgerufen werden:

- **Daten zu den gespeicherten Gerichten:**
  - ‚name'
  - ‚price'
  - ‚category'
  - ‚notes'

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**list_dishes_intent_handler**(*handler_input*)

Der Intent listet alle verfügbaren Gerichte auf, die nach verschiedenen Mustern gesucht werden:

1. Zunächst wird eine Mensa und ein Datum festgelegt. Die Slots werden elizitiert, wenn die Nutzeranfrage diese noch nicht enthält. 2. Die Suchanfrage kann spezifiziert werden, indem der User zusätzlich bis zu zwei Zutaten mit angeben kann, die entweder im Gericht enthalten sind oder nicht im Gericht enthalten sein sollen.

Der Benutzeranfrage und den bereitgestellten Slot Values werden folgende Daten entlockt:

- Tag bzw. Datum der Suchanfrage (erforderlich)
- Mensaname (erforderlich)
- (nicht) erwünschte Zutaten (optional)

Anschließend wird eine Suchanfrage mithilfe der Mensa-ID und dem Datum gestartet und die Daten von der OpenMensa API erfragt. Diese werden ebenfalls in den Session-Attributes gespeichert.

Gibt es zusätzlich Zutaten in der Nutzeranfrage, werden die API-Daten gefiltert.

Zuletzt werden die Ergebnisse in einen String überführt, für den Chunking benutzt wird, um die Antwort von Alexa kurz zu halten (siehe lambda_utility.py).

Außerdem werden bei jedem Turn nur vier Gerichte ausgegeben. Will der Nutzer mehr Gerichte erfahren, muss er nach „weiteren Gerichten“ fragen.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**price_intent_handler**(*handler_input*)

Der Intent gibt den Preis für ein bestimmtes Gericht zurück. Der Benutzer muss dabei die Nummer des Gerichts angeben und kann optional eine Zielgruppe (Studenten, Angestellte, Andere) definieren.

Der Benutzeranfrage und den bereitgestellten Slot Values werden folgende Daten entlockt:

- Nummer des Gerichts (erforderlich)
- Zielgruppe (optional)

Über die Session-Attributes müssen folgende Daten abgerufen werden:

- **Daten zu den gespeicherten Gerichten:**
  - ‚name‘
  - ‚prices‘

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**address_intent_handler**(*handler_input*)

Der Intent gibt die Adresse einer Mensa zurück. Benötigt wird der Name der Mensa. Ist dieser im Katalog, wird die Adresse zurückgegeben. Ist dieser nicht vorhanden, wird eine Fehlermeldung zurückgegeben.

Der Benutzeranfrage und den bereitgestellten Slot Values werden folgende Daten entlockt:

- Name und ID der Mensa (erforderlich)

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**list_mensas_intent_handler**(*handler_input*)

Der Intent gibt eine Liste mit Mensen in einer Stadt zurück. Benötigt wird der Name der Stadt. Sind Mensen vorhanden, werden diese zurückgegeben; gibt es keine oder ist die Stadt nicht in der Datenbank, wird eine Fehldermeldung zurückgegeben.

Bei jedem Turn werden nur vier Mensen ausgegeben. Will der Nutzer mehr Mensen erfahren, muss er nach „weiteren Mensen“ fragen.

Der Benutzeranfrage und den bereitgestellten Slot Values werden folgende Daten entlockt:

- Stadt (erforderlich)

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**get_nearest_mensa_intent_handler**(*handler_input*)

Der Intent gibt die vom Standort des Nutzers aus nächste Mensa und ihre Adresse zurück. Dafür muss der Benutzer seinen Standort für den Skill freigegeben haben.

Um die nächste Mensa berechnen zu können, werden die Koordinaten des Nutzers extrahiert. Anschließend wird mithilfe der Haversine-Formel die zum Benutzer nächstgelegene Mensa berechnet. Es wird also die Mensa mit der kleinsten Luftlinie zum Nutzer zurückgegeben.

Die Koordinaten des Nutzer werden folgendermaßen extrahiert:

a) Falls der Benutzer ein mobiles Gerät verwendet, werden die aktuellen Koordinaten aus dem GPS-Sensor des Geräts des Nutzers aus dem von Alexa an das Backend gesendete Request-JSON-Objekt extrahiert.

b) Falls der Benutzer ein stationäres Gerät verwendet, wird die vom Benutzer in der Alexa-App bzw. angegebene Adresse aus der Alexa-API extrahiert. Anschließend werden die Koordinaten der extrahierten Adresses mithilfe der Nominatim-API ermittelt.

Um dies zu ermöglichen, muss der Nutzer Zugriff auf seinen aktuellen Standort bzw. auf seine Adressdaten in der Alexa-App erlauben.

Wenn der Benutzer seinen Standort nicht freigegeben hat oder dieser gerade nicht extrahierbar ist, weil z.B. kein GPS-Signal verfügbar ist, werden je nach Errortype unterschiedliche Fehlermeldungen ausgegeben.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**log_response**(*handler_input*, *response*)

Response logger.

lambda_function.**log_request**(*handler_input*)

Request logger.

lambda_function.**launch_request_handler**(*handler_input*)

Der Intent wird beim Öffnen des Skills durch Modal-Launchphrasing getriggert. Er gibt eine Willkommensnachricht zurück.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**no_intent_handler**(*handler_input*)

Der Intent wird ausgelöst, wenn der Benutzer eine Rückfrage des Skills verneint. Er gibt eine Verabschiedung zurück und beendet den Skill.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**next_intent_handler**(*handler_input*)

Der Intent führt den ListDishesIntent weiter und listet weitere Gerichte auf. Die benötigten Daten werden aus den Session-Attributes entnommen.

Jeder Turn gibt nur vier Gerichte aus. Will der Nutzer mehr Gerichte erfahren, muss er nach „weiteren Gerichten“ fragen.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**help_intent_handler**(*handler_input*)

Der Intent unterstützt den Nutzer bei der Bedienung des Skills. Er informiert den Nutzer über den Umfang des Skills und gibt Beispielanfragen zurück.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**exit_intent_handler**(*handler_input*)

Der Intent stoppt den Skill mit einer Verabschiedung.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**session_ended_request_handler**(*handler_input*)

Der Intent stoppt den Skill ohne Verabschiedung.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die response ohne Antwort zurück

Rückgabetyp Response

lambda_function.**fallback_intent_handler**(*handler_input*)

Der Intent informiert den Benutzer darüber, dass der Skill die gewünschte Funktionalität nicht besitzt und beendet ihn.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**all_exception_handler**(*handler_input*, *exception*)

Der Intent ist ein Fallback für alle Exceptions. Er informiert darüber, dass der Skill die gewünschte Funktionalität nicht besitzt und beendet ihn.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**unhandled_intent_handler**(*handler_input*)

Ein Intent für alle „Unhandled requests“. Er informiert darüber, dass der Skill die gewünschte Funktionalität nicht besitzt und beendet ihn.

Parameter **handler_input** (*(HandlerInput) -> Response*) – HandlerInput

Rückgabe Gibt die vollständige Skill-Antwort zurück

Rückgabetyp Response

lambda_function.**invoke_skill**()

POST Methode, die den Skill als FLASK app startet.

Rückgabe Startet den Skill

Rückgabetyp dispatch_request

KAPITEL 2

Lambda Utility

lambda_utility.**create_mensa_url**(*mensa_id*, *date*)
Baut den String für die OpenMensa API Anfrage.

Parameter

- **mensa_id** (*str*) – die Mensa-ID als String
- **date** (*str*) – das Datum als String

Rückgabe Gibt die Webseite der API als String zurück

Rückgabetyp str

lambda_utility.**random_phrase**(*str_list*)
Gibt einen zufälligen String aus einer String-Liste zurück.

Parameter **str_list** (*List[str]*) – Liste mit Strings

Rückgabe Gibt einen String zurück

Rückgabetyp str

lambda_utility.**http_get**(*url*, ***kwargs*)
Erfragt Daten mithilfe des requests-Moduls und fängt Error ab.

Parameter **url** (*str*) – die API url, die angefragt wird

Verursacht

- **ValueError** – ValueError, wenn die Response leer ist
- **[ErrorType]** – Je nach Statuscode, den die Response enthält

Rückgabe Gibt eine Antwort des requests-Moduls als json zurück.

Rückgabetyp json

lambda_utility.**http_get_iterate**(*url*)
Erfragt Daten mithilfe des requests-Moduls und fängt Error ab. Iteriert über alle Seiten einer API

Parameter **url** (*str*) – die API url, die angefragt wird

Verursacht **[ErrorType]** – Je nach Statuscode, den die Response enthält

Rückgabe Gibt die Antworten des requests-Moduls als json zurück.

Rückgabetyp json

lambda_utility.**chunking**(*meal*)
Hilfsfunktion, die naives Chunking auf die Gerichte anwendet, für die es keine ähnlichen Gerichte in der globalen Liste gibt. Gesplittet wird an Präpositionen.

Parameter **meal** (*str*) – Name des Gerichts, auf den Chunking angewendet wird.

Rückgabe Gibt den verkürzten String zurück.

Rückgabetyp str

lambda_utility.**find_difference**(*duplicates*, *all_dishes*)
Hilfsfunktion für Chunking, die ähnliche Gerichte in der Liste aller Gerichte sucht und diese so verkürzt, dass nach der Präposition „mit“ der Teil dieser ähnlichen Gerichte steht, der sie unterscheidet.

Parameter **duplicates** – Liste ähnlicher Gerichte

Rückgabe Gibt die Liste der ähnlichen Gerichte zurück, wenn diese vorkommen

Rückgabetyp List

lambda_utility.**make_chunking**(*all_dishes*)
Hauptfunktion des Chunkings der Gerichte. Die Länge der Namen der Gerichte in der endgültigen Liste wird durch den parameter cutoff von der overlap-Funktion kontrolliert.

Parameter **all_dishes** (*List*) – Liste aller Gerichte.

Rückgabe Gibt eine Liste mit verkürzten Namen der Gerichte zurück

Rückgabetyp List

lambda_utility.**build_mensa_speech**(*mensalist*, *start_idx*)
Baut den String, der anschließend für den Prompt des ListMensasIntents benutzt wird. Die Liste ist möglicherweise sehr lang. Deshalb wird nur das Element am Startindex und drei weitere Elemente in den Prompt gebaut, um zu lange Antworten zu vermeiden.

Der letzte Index wird gemerkt und zusammen mit dem String zurückgegeben. Dieser fungiert beim nächsten Durchlauf als Startindec, damit die Liste an derselben Stelle fortgesetzt wird.

Parameter

- **mensalist** (*List[str]*) – Eine Liste mit Mensen als Strings
- **start_idx** (*int*) – Der Index, bei dem angefangen werden soll zu suchen.

Rückgabe Gibt den fertigen String un den letzten Index für den nächsten Durchlauf zurück.

Rückgabetyp str, int

lambda_utility.**build_dish_speech**(*dishlist*, *start_idx*)
Baut den String, der anschließend für den Prompt des ListDishesIntents benutzt wird. Die Liste ist möglicherweise sehr lang. Daher wird nur das Element am Startindex und drei weitere Strings in den Prompt gebaut, um zu lange Antworten zu vermeiden.

Der letzte Index wird gemerkt und zusammen mit dem String zurückgegeben. Dieser fungiert beim nächsten Durchlauf als Startindex, damit die Liste an derselben Stelle fortgesetzt wird.

Parameter

- **dishlist** (*List[str]*) – Eine Liste mit Gerichten als Strings
- **start_idx** (*int*) – Der Index, bei dem angefangen werden soll zu suchen.

Rückgabe Gibt den fertigen String und den letzten Index für den nächsten Durchlauf zurück

Rückgabetyp str, int

lambda_utility.**build_preposition_speech**(*ingredients*)
Baut einen zusätlichen String, der Zutaten enthält, um dem Nutzer mitzuteilen, wonach er gesucht hat.

Parameter **ingredients** (*dict*) – Ein Dictionary mit Zutaten-Strings

Rückgabe Gibt den fertigen String zurück

Rückgabetyp str

lambda_utility.**build_price_speech**(*price*, *user*)
Baut die Prompt für den Preis.

Parameter

- **price** (*str*) – Der Preis als String
- **user** (*str*) – Die Zielgruppe als String

Rückgabe Gibt den fertigen String zurück

Rückgabetyp str

lambda_utility.**get_resolved_value**(*request*, *slot_name*)
Die Funktion löst einen slot name mithilfe von slot resolutions auf, um einen „dahinterliegenden“ Wert zu finden.

„Resolve the slot name from the request using resolutions.“

Parameter

- **request** (*IntentRequest*) – Die Anfrage des Intents
- **slot_name** (*str*) – Name des Slots

Verursacht

- **AttributeError** –
- **KeyError** –
- **ValueError** –
- **IndexError** –
- **TypeError** –

Rückgabe Gibt entweder den resolved slot value zurück oder None

Rückgabetyp str, None

lambda_utility.**get_slot_values**(*filled_slots*)
Extrahiert zusätzliche Informationen zum Slot filling, wie z.B. die ID eines slots, ob der Slot in den Slot values des Interaction Models vorhanden ist und seinen resolved value.

„Return slot values with additional info.“

Parameter **filled_slots** (*Dict[str, Slot]*) – Ein Dictionary mit allen Slots

Verursacht

- **AttributeError** –
- **KeyError** –
- **ValueError** –
- **IndexError** –
- **TypeError** –

Rückgabe Gibt die extrahierten Values mit zusätzlichen Infos zurück

Rückgabetyp Dict[str, Any]

lambda_utility.**find_matching_dishes**(*api_response*, *ingredients={'first': None, 'second': None}*)
Filtert die aus der API-Antwort erhaltenen Gerichte nach bestimmten Zutaten. Hat der User keine Zutaten angebeben, sind diese „None“ und es wird die komplette API-Antwort zurückgegeben.

Parameter

- **api_response** (*json*) – Die json response der OpenMensa API

- **ingredients** (*Dict[str, [None, str]] (Default: {'first' : None, 'second' : None})*) – Dictionary mit Zutaten-String

Rückgabe Liste mit den erwünschten Gerichten

Rückgabetyp List[str]

lambda_utility.**ingredient_in_dish**(*ingredient*, *dish*)

Checkt, ob eine bestimmte Zutat in einem Gerichte-String oder in den Notes enthalten ist.

Parameter

- **ingredients** (*Dict[str, [None, str]]*) – Dictionary mit Zutaten-String
- **dish** (*Dict[str, Any]*) – Dictionary mit allen Daten zu einem Gericht

Rückgabe Gibt True zurück, wenn die Zutat gefunden wurde, sonst False

Rückgabetyp Boolean

lambda_utility.**ingredient_fleisch**(*dishlist*, *first_prep*, *second_prep*, *is_first_ingredient=True*)

Filtert die erhaltenen Gerichte nach veganen und vegetarischen Gerichten, da dies nicht aus den Daten der API ersichtlich ist.

Parameter

- **dishlist** (*List[str]*) – Liste mit Gerichten
- **first_prep** (*str*) – Die erste Präposition
- **second_prep** (*str*) – Die zweite Präposition
- **is_first_ingredient** (*Boolean (Default: True)*) – TODO Docu

Rückgabe Liste mit Gerichten mit/ohne Fleisch

Rückgabetyp List[str]

lambda_utility.**calculate_nearest_mensa**(*user_coordinates*, *mensa_list*)

Berechnet die zum User nächstgelegene Mensa (Luftlinie) mithilfe der Haversine-Formel.

Parameter

- **user_coordinates** (*Tuple[float]*) – Paar mit Koordinaten des Users
- **mensa_list** (*List[dict]*) – Liste aller Mensen

Rückgabe Paar mit Name und Adresse der nächsten Mensa

Rückgabetyp Tuple[str]

lambda_utility.**convert_acronyms**(*speech*)

Konvertiert Akronyme in einem Speech-Output in eine Form, die Alexa zeigt, dass das Akronym nicht als Wort sondern Buchstabe für Buchstabe ausgesprochen werden soll. Diese Funktion ist wichtig, da Alexa ansonsten Universitäten wie „FU Berlin“ als [fuː bɛʊˈliːn] aussprechen würde.

Parameter **speech** (*str*) – Speech-Output als String

Rückgabe Gibt einen umgewandelten Speech-String zurück

Rückgabetyp str