Idee

EDEKA DIGITAL ist weit mehr als www.edeka.de oder vielleicht noch die EDEKA-App. Unter Anderem wird durch EDEKA DIGITAL die Plattform Bringmeister gepflegt, mit der in verschiedene Städten Lebensmittel bestellt werden können.
Der Alexa Skill soll bei Aufforderung nach Produkten im Bestand von Bringmeister suchen und diese in einen Warenkorb hinzufügen. Bei schönem Wetter soll dem Kunden jedoch nahe gelegt werden doch selbst einkaufen zu gehen

Beispiele für Interaktionen wären also:

“Alexa, Ich brauche Tomaten”
“Soll ich EDEKA Selektion Cherry Rispentomaten in deinen Warenkorb legen?”
“Ja.”
“Ist erledigt.”

“Alexa, Was ist in meinem Warenkorb?”
“Du hast 12 Produkte im Wert von 32 Euro und 22 Cent in deinem Warenkorb. Soll Ich sie auflisten?”

“Alexa, Entferne Bananen aus meinem Warenkorb”

“Alexa, Liefere Mittwoch um 10 Uhr”

Installation

Um diesen Skill zu erstellen, müssen folgende Schritte durchgeführt werden.

Endpoint
Java 8, Maven

$ cd Bringmeister/endpoint
$ make build
$ cd test
$ java -jar Bringmeister-voice-1.0-SNAPSHOT.jar
z.B.: $ ngrok http 8080 (um Tunnel zu erstellen und lokalen Port SSL-Verschlüsselt freizugeben)

Alexa Developer Console

1. Projekt erstellen (Custom, Self-Hosted)
2. Custom → Invocations → JSON editor -> de_DE.json hochladen
3. Custom → Endpoint → HTTPS -> Default Region: https://tunnel/endpoint & “My development endpoint is a subdomain …”
4. Models → Account Linking
    a. “Do you allow users to create an account to an existing account with you?” aktivieren
    b. Auth Code Grant: Your Web Authorization URI: https://tunnel/account/login
    c. Auth Code Grant: Access Token URI: https://tunnel/account/token
    d. Your Client ID: Bringmeister

e. Your Secret: <Secret> (ist letztlich irrelevant)

Erste Überlegungen

Der erste Schritt überhaupt war es mittels Analyse der Netzwerk-Daten mithilfe der Chrome Dev Tools zu ermitteln, welche Schnittstellen das Frontend der Webseite www.bringmeister.de nutzt. Es wurde klar, dass für einiges eine klassische REST API verwendet wird. Für andere Dinge - v.a. für das Speichern der Kundendaten - wird GraphQL genutzt. Das heißt, dass das Backend des Alexa Skills auf diese Schnittstellen zugreifen muss.

Einrichtung

Um den Alexa Skill generell aufrufen zu können, wurde in der Alexa Developer Konsole ein Projekt angelegt. Es wird sich für eine Self-Hosted-Methode entschieden. Anschließend wurde mittels des Spring Boot Frameworks in Java eine Schnittstelle aufgebaut, auf den später der Skill zugreifen kann. Dabei hilft das eigens von Amazon entwickelte Alexa SDK. Mithilfe des Spring Frameworks wird nun ein Alexa Servlet unter /endpoint/ eingestellt. Unter Endpoint in der Konsole kann diese (z.B. über Tools wie ngrok SSL-verschlüsselte und öffentliche) Adresse mit dem Skill verbunden werden. In der Alexa App kann dieser Skill nun aktiviert werden.

Account Linking

Eine zweite Überlegung war es zu klären, wie überhaupt der Alexa Skill auf das Bringmeister-Kundenprofil zugreifen soll. Und was passiert, wenn noch überhaupt kein Profil existiert? Alexa bietet das sogenannte Account Linking. Selbstverständlich kann meine Software nicht direkt auf Bringmeister zugreifen. Zu Demonstrationszwecken ist daher ein eigenes Account Managing eingerichtet worden. Das heißt: Es wird ein kleiner Umweg genommen und meine Software wird mit dem Alexa Skill verbunden. Da dieses Account Linking lediglich zu Demonstrationszwecken dienen soll, handelt es sich lediglich um eine rudimentäre Benutzeroberfläche.

Gemäß des Aufbaus des Account Linkings wird ein Anmeldeformular über Spring Boot, welches auch im Späteren selbst auch mit der Alexa SDK verwendet wird, erstellt unter /account/login. Gefordert werden die Parameter state und redirect_uri. Aktiviert der Nutzer nun diesen Skill so wird er durch die Alexa App auf diese Seite gelenkt. Gibt der Nutzer dort seine Bringmeister-Anmeldedaten an, so wird auf Bringmeister GraphQL zugegriffen. Der erhaltene API-Token wird nun symmetrisch verschlüsselt als Code an die von Alexa geforderte Redirect-URI gesendet. Fordert Alexa jetzt nun über /account/token den Access Token, so wird dieser Code entschlüsselt und freigegeben.

Dieser Workaround entspricht dem Aufbau des vorgegebenen Alexa Account Flows. Nun kann mein Alexa Skill auf Bringmeister zugreifen. Nirgends wird das angegebene Passwort durch meine Software gespeichert.

Services

Bevor die eigentliche Kommunikation mit Alexa selbst implementiert wird, werden Schnittstellen mit den Bringmeister-Servern aufgebaut und Daten werden im Backend selbst dargestellt.

Nähere Erläuterungen zum Aufbau der Software selbst, findet sich in den JavaDocs.

UserService
Dieser Service dient ausschließlich zur Kommunikation über GraphQL mit den Bringmeister-Servern. Hier werden Benutzerkonto-bezogene Aktionen durchgeführt und verwaltet wie das Anmelden selbst oder das Hinzufügen und Löschen von Produkten im Warenkorb. Dazu wird ein GraphQL Spring Boot Client verwendet. So wird in der Methode *getToken(User user)* die Query aufgerufen, um den Token zu erhalten. Weitere Methoden fügen Produkte in den Warenkorb hinzu oder geben die Auslieferungsadresse des Nutzers zurück.

DatServe
Dieser Service dient zur Kommunikation mit öffentlichen und der mehr oder weniger statischen REST APIs wie OpenWeatherMap oder dem Teil des GraphQL-Servers, der keiner Autorisierung bedarf. So werden hier beispielsweise Produkte gesucht oder aktuelle Angebote werden geladen.

Intents

Statische Intents
Es gibt vier statische Anfragen an Alexa, die immer das gleiche zurückgeben: Die Begrüßung, die Verabschiedung, die Hilfe-Intent und die Unhandled-Intent, welche nichts zurückgibt im Falle, das eine Anfrage nicht verarbeitet wird.

AddCartItemIntent
Diese Intent ist die zentrale Intent des Skills. Diese soll ein Produkt in den Warenkorb hinzufügen können. Dazu sind verschiedene Eingaben möglich. Es wird der Parameter *query* benötigt, der die Suchanfrage an die Bringmeister-Server darstellt. Ist diese Query angegeben, so kann die Anfrage verarbeitet werden. Es wird mit hilfe des DataService nach Produkten gesucht, die der Suchanfrage entsprechen. Es wird ebenfalls die Postleitzahl des Nutzers in der Suchanfrage angegeben gemäß des Aufbaus der Bringmeister API, damit nur Produkte übermittelt werden, die überhaupt möglich sind für den Nutzer. Wurde nun das Produkt mit der größten Übereinstimmung gefunden, so werden Name (inkl. Packungstypen), sowie interne Daten wie sku und unitId gesichert. Der Nutzer wird nun gefragt, ob das gefundene Produkt in den Warenkorb hinzugefügt werden kann. Darauf kann er mit *Ja* und *Nein* antworten. Aufgrund dessen, dass eine spezifische Confirmation Prompt verwendet werden soll (in der Prompt ist der Name des Produktes enthalten) kann nicht das klassische Dialog System Alexas verwendet werden. Daher wird das gefundene Produkt in den Session Attributen zwischengespeichert. Die YesIntent und die NoIntent (die als Subklassen umgesetzt wurden) werden nun freigeschaltet. Antwortet der Nutzer mit *Nein* so wird eine statische Ausgabe gesendet und der Zwischenspeicher wird geleert. Antwortet er mit *Ja* so wird das Produkt mithilfe des UserService in den Warenkorb gelegt.

Mögliche Fehler, die während dieses Prozesses auftreten können wie z.B. fehlende Postleitzahl seitens des Nutzers oder eine nicht erfolgreiche Suche werden durch Exceptions aufgegriffen.

An dieser Stelle wird auch die Einbindung der Wetterdaten implementiert. Dazu wird die Methode *isGoodWeather(city: String)* des DataService aufgerufen. Diese Methode greift auf OpenWeatherMap Daten zurück und gibt an, dass gutes Wetter ist, wenn der Wetter-Code mit 8 beginnt (siehe OpenWeatherMap Dokumentation). Nach Rückfrage mit dem IT-Talents-Team wird diese Methode nicht mit dem tatsächlichen Standort der Alexa Anfrage aufgerufen, sondern mit dem Standort, der bei Bringmeister hinterlegt ist.

RemoveCartItemIntent
Parallel zu dem AddCartItemIntent existiert eine Aufforderung, die ein Produkt sucht und aus dem Warenkorb entfernt. Auch hier gibt es wieder eine Query, die über den DataService gesucht wird. Zusätzlich wird auch geprüft, ob das Produkt im Warenkorb ist. Nun wird der Nutzer gefragt, ob dieses Produkt entfernt werden soll. Ist dies der Fall, so wird eine GraphQL Request gestellt.

CartIntent
Ruft der Nutzer diese Intent auf, so werden ihm Infos über seinen Warenkorb genannt. Er kann auswählen, ob er seinen Warenkorb aufgelistet hören möchte. Auch hier wurde wieder das Prinzip der YesIntent und der NoIntent genutzt. Daten werden auch hier wieder über den UserService erhalten und als Objekt der Klasse Cart gespeichert. Wird die Intent ausgelöst, wird sein Warenkorb geladen und eine Ausgabe wird produziert (z.B. ein Produkt, kein Produkt, N Produkte im Warenkorb). Ist mehr als ein Produkt im Warenkorb wird die Frage gestellt, ob der Warenkorb aufgelistet werden soll. Über Session Attribute werden die Produkte und die gestellte Frage gesichert, sodass sie bei Aufrufen der YesIntent aufgelistet werden können.

TimeSlotIntent
Auch diese Methode arbeitet mit der YesIntent und NoIntent. Die Intent wird mit den Daten day und time aufgerufen. Im Backend wird dann über GraphQL eine Zeitschiene zur Lieferung gesucht, die den angegebenen Daten nahe liegt. Diese wird dann ausgegeben mit zugehörigem Lieferpreis. Der Nutzer kann nun entscheiden, ob er diese Zeitschiene auswählen möchte. Ist dies der Fall, so wird wie bekannt über den Sessionstorage die ID des Slots aufgerufen und im Backend wird diese Zeitschiene gesetzt.