

Hochschule für angewandte Wissenschaften Coburg  
Fakultät Elektrotechnik und Informatik

Dokumentation

Communications Engineering - Project

Alexa Skill in Verbindung  
mit einer API

Timo Purzner

Abgabe der Arbeit: 30. 07 2017

Betreut durch:  
Prof. Dr. Matthias Mörz, Hochschule Coburg

# Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc482701656)

[Abkürzungsverzeichnis 4](#_Toc482701657)

[1 Einleitung 5](#_Toc482701658)

[1.1 Vorstellung Alexa 5](#_Toc482701659)

[1.2 Projektbeschreibung 5](#_Toc482701660)

[2 Komponenten für den Alexa-Skill 6](#_Toc482701661)

[2.1 Nodejs 6](#_Toc482701662)

[2.2 Express 6](#_Toc482701663)

[2.3 Yeoman 6](#_Toc482701664)

[2.4 Alexa-App 6](#_Toc482701665)

[2.5 Postman 7](#_Toc482701666)

[3 Veröffentlichen eines Alexa-Skills ohne AWS 8](#_Toc482701667)

[3.1 Veröffentlichung eines Skills bei Amazon 8](#_Toc482701668)

[3.2 Zertifizierung 8](#_Toc482701669)

[4 Aufbau eines Alexa-Skill 9](#_Toc482701670)

[4.1 Inhalt Hauptteil (index.js) 9](#_Toc482701671)

[4.2 Intents 9](#_Toc482701672)

[4.2.1 Slots 9](#_Toc482701673)

[4.2.2 Utterances 9](#_Toc482701674)

[5 Implementierung des Skills 10](#_Toc482701675)

[5.1 Konzept 11](#_Toc482701676)

[5.1.1 Programmablauf 11](#_Toc482701677)

[5.1.2 Programmaufbau 11](#_Toc482701678)

[5.2 Hauptprogramm 11](#_Toc482701679)

[5.3 Intent StartGame 11](#_Toc482701680)

[5.3.1 Slots 11](#_Toc482701681)

[5.3.2 Utterances 11](#_Toc482701682)

[5.4 Intent GameRound 11](#_Toc482701683)

[5.4.1 Slots 11](#_Toc482701684)

[5.4.2 Utterances 11](#_Toc482701685)

[5.5 Intent Help 11](#_Toc482701686)

[5.5.1 Slots 11](#_Toc482701687)

[5.5.2 Utterances 11](#_Toc482701688)

[5.6 API Funktionen 11](#_Toc482701689)

[6 Fazit 12](#_Toc482701690)

[Glossar 13](#_Toc482701691)

# Abkürzungsverzeichnis

AWS Amazon web Service

API Application programming interface

AVS Alexa Voice Service

NPM Node Package Manager

# Einleitung

Der Begriffe Alexa beschreibt den von Amazon angebotenen Sprachassistenten, dabei muss zwischen der Software (Alexa) und der Hardware (Echo) unterschieden werden. Dieses Kapitel bezieht sich auf [Amazon 2017].

Der Alexa Voice Service (AVS) ist ein Cloud Service, welcher von Amazon bereitgestellt wird, um von Anwendern Gesprochene Anfragen zu bearbeiten, dazu gehört Musik abspielen, Fragen beantworten, Nachrichten vorlesen und Smarthome Geräte steuern. Dies alles geschieht ausschließlich über eine Eingabe mit der Stimme.

Amazon bietet dabei nicht nur die Software (AVS) sondern auch die Hardware (Echo) an, welche die Stimme aufnimmt und an den AVS sendet. Alexa kann aufgrund dieser Trennung auch auf eigner Hardware, wie einem Raspberry-Pi betrieben werden. Der Vorteil der von Amazon gestellten Hardware ist, dass der Echo mit 7 Mikrophonen ausgestattet ist so das gesagtes über den ganzen Raum hinweg von Alexa gehört werden kann.

Amazon bietet dabei zwei verschiedene Modelle des Echos an, zum einen den „normalen“ Echo und zum anderen den Echo dot. Diese Beiden Geräte unterscheiden sich lediglich in Größe, Sound Qualität und dem Preis. Der Echo dot ist deutlich kleiner und hat somit weniger Platz für einen großen Lautsprecher welcher zum Musikhören geeignet ist. Da allerdings die Alexa in der Cloud bei Amazon läuft, haben beide Modelle denselben Funktionsumfang was den Umgang mit dem Sprachassistenten betrifft.

Neben Alexa gibt es noch weitere Sprachassistenten von verschiedensten Herstellern in der folgenden Tabelle wird ein kleiner Überblick über alternativen gegeben.

| **Hersteller** | **Amazon** | **Google** | **Apple** | **Samsung** |
| --- | --- | --- | --- | --- |
| Name des Assistenten | Alexa | Google Assistent | Siri | Bixby |
| Wake-up Wort | Alexa | Ok Google | Hey Siri | Nur über einen Button |
| Smart Speaker | Ja | Ja | Angekündigt | Nein |
| Handy Assistent | Nein | Ja | Ja | Ja |
| Hardware gebunden | Nein | Ja | Ja | Ja |
| Software gebunden | Nein | Ja | Ja | Ja |
| Apps von 3 Anbietern | Ja | Ja | Ja | Ja |
| Besonderes | Kann unabhängig von Hardware laufen |  | Auf allen Apple Geräten verfügbar |  |

Tab. Übersicht Smart Assistenten [android 2017]

In diesem Projekt wird ein Skill für Alexa entwickelt, dieser Skill wird nicht auf dem Amazon Servern gehostet, sondern auf einem Privaten Server. Dabei wird in dieser Dokumentation darauf eingegangen welche Bedingungen erfüllt sein müssen, damit der Skill auf dem eigenen Server laufen kann und welche Einstellungen auf der Developer-Seite von Amazon zu machen sind. Es wird auch beschrieben wie ein Alexa-Skill aufzubauen ist (s. 4) und wie dieser lokal auf dem Rechner getestet werden kann (s. 2.4). Als letztes wird der entwickelte Skill beschrieben (s. 5). Dabei wird vorausgesetzt das Programmierkenntnisse vorhanden sind.

# Tools für den Alexa-Skill

Hier werden alle Komponenten beschrieben welche für die Erstellung des Skills genutzt wurden. Dazu gehört Nodejs als Programmiersprache und die in Nodejs verwendeten Module. Der in diesem Projekt erstellte Alexa-Skill wurde aus einem Template generiert welches von Yoeman bereitgestellt wird. Als letztes wird das Framework Alexa-App beschrieben welches genutzt wurde um den auf Anfragen von Alexa zu Reagieren. Um Lokal auf dem Rechner den eigenen Code testen zu können wird das Tool Postman verwendet.

## Nodejs

Der Alexinator-Skill wird mit Nodejs geschrieben, Nodejs ist eine asynchrone, Event-basierte Laufzeitumgebung welche auf der JavaScript Engine basiert. Nodejs bietet den Node Package Manager (NPM) um einfach zusätzliche Module dem Projekt hinzuzufügen. Welche Module Installiert sind kann in der Datei package.json eingesehen werden, dort wird Name und Versionsnummer der Module im JSON-Format gespeichert.

Da der Skill auf dem eignen Server laufen soll wird ein Framework gebraucht um HTTP-Anfragen zu bearbeiten, dazu wird das Framework *Expressjs* verwendet. AVS erwartet von dem Server auf dem der Skill läuft, HTTP-Antworten mit bestimmten JSON Format, diese werden Generiert von dem Framework Alexa-App, dieses Framework Arbeitet mit Expressjs um die HTTP-Anfragen und –Antworten zu bearbeiten.

In diesem Projekt wurde ein vorgeneriertes Template von Yeoman (s. 2.2) genutzt, welches auf den verwendeten Frameworks aufbaut.

Es werden neben Alexa-App und Expressjs weitere Module verwendet. Unteranderem das NPM Modul Request. Module können mit npm install Name installiert werden, also wird Request mit npm install request installiert. Mithilfe dieses Moduls werden Anfragen an bestimmt URLs gesendet, im Falle dieses Projekts werden Anfragen an die Akinator-API geschickt und anschließend ausgewertet.

Nodejs ist asynchron, doch an einigen Stellen im Projekt ist es wichtig Synchronität zu erreichen, dazu werden Promises verwendet [MDN 2017]. Zum Beispiel, wenn das Framework Alexa-App warten soll bis eine funktion durchlaufen wurde bevor eine Antwort an AVS zurücksendet. Promieses werden im Projekt verwendet, wenn mit einer API Kommuniziert wird, da darauf gewartet werden muss das diese API eine Antwort gesendet hat.

Um einen Node-Server zu starten muss im Verzeichnis des Projekts der Befehl npm start ausgeführt werden. Um alle Module zu installieren, die in der Datei package.json stehen muss der Befehl npm install ausgeführt werden.

## Yeoman

Yeoman ist eine Hilfe im neue Projekte zu starten, es gibt Strukturen des Projekts vor, diese müssen anschließend noch an das eigene Projekt angepasst werden (s. 5.7) [Yeoman 2017]. Mithilfe von Yeoman können von Nutzer bereitgestellte Templates für das eigene Projekt verwendet werden. In diesem Projekt wird das Template [Jseijas 2017] genutzt.

Um Yeoman zu nutzen wurde NPM (s. 2.1) verwendet. Zuerst muss Yeoman installiert werden, anschließend muss das Template bekannt gemacht werden und im letzten Schritt muss dieses ausgeführt werden. Daraus folgt es muss folgender Code in der Konsole ausgeführt werden [Yeoman 2017].

npm install –g yo

npm install –g generator-GENERATOR\_NAME

yo GENERATPR\_NAME

Code Getting Startet with Yeoman

Für dieses Projekt gemachte Anpassungen werden unter 5.7 beschrieben.

## Alexa-App

Alexa-App ist ein Nodejs-Framework um Anfragen von AVS zu Verarbeiten und zu beantworten. Dieses Framework ist dabei ausgelegt um sowohl auf AWS zu funktionieren, als auch ohne auf dem eignen Server. Wenn das Framework auf dem eignen Server läuft nutzt es Expressjs, in Kombination mit AWS wird Expressjs nicht gebraucht [Alexa-App 2017]. Da in diesem Projekt der Skill auf einem eigenen Server laufen soll wird nicht das von Amazon bereitgestellte Alexa-SDK verwendet, da dieses nur mit AWS Funktioniert.

Das Framework erstellt die JSON-Antwort welche an AVS gesendet werden muss und liest die Antwort aus und ruft automatisch den Richtigen Intent auf und stellt alle Informationen bereit. Alexa-App stellt auch Funktionen z.B. für die Verwaltung von Sessions, Karten erstellen und Audiostreaming bereit.

## Postman

In diesem Kapitel wird das Tool „Postman“ beschrieben, welches in diesem Projekt zum Testen des Alexa-Skills dient. In diesem Kapitel wird beschrieben was Postman ist und welche Funktionen es bietet um den Alexinator-Skill zu testen. Postman dient dazu den AVS-Dienst zu simulieren, so das der Skill Entwickelt und getestet werden kann ohne ihn nach jeder Änderung auf den Server laden zu müssen. Es wird nicht beschrieben wie Postman grundlegend funktioniert dazu kann in der Doku von Postman nachgelesen werden [Postman 2017]. Wie der Alexa-Skill auf diese Anfragen reagiert wird unter 2.4 beschrieben.

Der AVS-Dienst sendet HTTP-Anfragen an den Server auf dem der Alexa-Skill läuft, in dem er den Body des Requests mit einem JSON Ausdruck füllt. Diese HTTP-Anfragen können mit Postman simuliert werden.

Postman sendet HTTP-Anfragen an einen Server, dabei unterstützt Postman viele Übertragungs-Methoden für unseren Skill ist nur die POST Methode wichtig. Da der Alexa-Skill lokal getestet werden soll wird Postman die Anfragen an localhost senden.

Dabei bietet Postman mehrere nützliche Features, Postman zeigt die HTTP-Antwort des Servers an auf dem der Alexinator-Skill läuft. So kann getestet werden ob der Skill die Antwort an Alexa zurücksendet, welche vom Programmierer vorgesehen wurde. In Postman können auch Tests definiert werden welche bei jedem Aufruf der HTTP-Anfrage durchlaufen werden. In diesen Testest wird geprüft ob in der HTTP-Antwort die richten Attribute vorhanden sind. Postman bietet ein Feature um Dynamischen Inhalt in verschiedenen HTTP-Anfragen zu verwenden, so kann die Antwort einer Anfrage gespeichert werden und in der Nächsten Anfrage wiederverwendet werden [Postman 2017]. Beispiele für die Anwendung dieser Funktionen sind unter 5.7 zu finden.

# Veröffentlichen eines Alexa-Skills ohne AWS

In diesem Kapitel wird beschrieben wie man einen eigenen Alexa-Skill ohne AWS veröffentlichen kann, dabei ist das Ziel, dass dieser aus dem Amazon Store für Alexa-Skills auf das eigene Alexa-Gerät heruntergeladen werden kann. Dazu wird der Alexa-Skill auf einem eigenen Server gehostet, dieser Server wird bei Amazon eingetragen und Amazon leitet anfragen welche an den Skill gehen an den eigenen Server weiter.

Damit der Skill auf einem eigenen Server laufen kann müssen einige Kriterien erfüllt werden, das bekannt machen des Server bei der Erstellung des Skills bei Amazon und der Server muss über ein gültiges SSL-Zertifikat verfügen.

## Veröffentlichung eines Skills bei Amazon

In diesem Abschnitt wird beschrieben wie ein Skill auf der Amazon Developer Seite angelegt und Regestiert werden kann. Dieses Kapitel bezieht sich auf den in diesem Projekt Implementierten Alexinator-Skill. Dazu muss auf der Amazon Developer Seite ein neuer Skill Registriert werden. Danach erscheint eine Seite mit folgenden auszufüllenden Punkten.

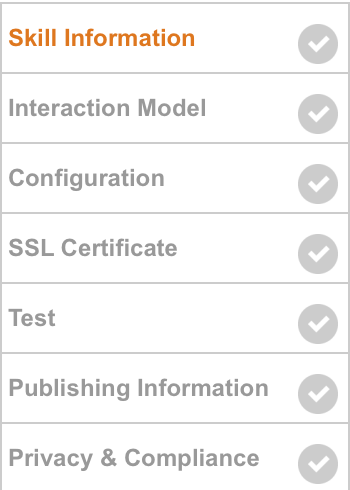


Abb. Information für einen Skill [Amazon 2017]

Skill Information:

Unter diesem Punkt werden Allgemeine Innformationen zu dem Skill angeben.

In diesem Projekt wird ein selbsterstellter deutscher Skill geschrieben daraus folgt folgende Einträge:

Skill Type:

Als Skill Type wird Custom ausgewählt.

Language:

Als Sprache für diesen Skill wird German gewählt.

Name:

Als Name welcher dem Nutzer Angezeigt wird, wird Alexinator verwendet.

Invocation Name:

Als Befehl welcher deinen Skill Startet wird Alexinator gewählt.

Audio Player:

Da keine Audio Streams gemacht werden wird „No“ ausgewählt.

Interaction Model

Unter diesem Punkt werden die Intents (4.2), Utterances (4.2.1) und Slots (4.2.2) festgelegt. In diesem Projekt wird der von Amazon bereit gestellte Dienst Builder welcher sich im Moment (14.06.2017) in der Beta befindet.

In diesem Builder sind bereits die Intents Cancel- Help- und StopIntent erstellt, welcher jeder Skill haben muss. Es müssen noch zusätzliche die Intents StartGame (s. 5.3) und GameRound (s. 5.4) angelegt werden. Für den Intent GameRound, wird ein Slot mit dem Namen „answers“ erstellt, welcher die Antworten des Spieles darstellt.

* Was muss auf der Amazon Developer Seite angegeben werden
* Evtl. mit Bildern eine Art Anleitung schreiben
* Für Intents und Slots verweis auf 4.2 und 4.3 da diese erst bei Programmieren entstehen

## Zertifizierung

* Was ist SSL
* Wozu SSL
* Verweis auf Lets Encyrpt

# Aufbau eines Alexa-Skill

In diesem Kapitel wird der Aufbau eines Alexa-Skills beschrieben, ohne dabei auf die Implementierung eines bestimmten Skills einzugehen. Dabei werden die einzelnen Bestandteile eines Normalen Alexa-Skills beschrieben. Zu diesen Bestandteilen gehören die Funktionen welche beim Starten des Servers ausgeführt werden und die Funktionen, welche beim Starten der App ausgeführt werden (s. 4.1). Desweitern sind Bestandteile des Skill die Intents, eine Skill besteht aus mehreren Intents. Ein Intent stellt die Reaktion auf eine bestimmte Spracheingabe des Nutzers da. Dabei wird für verschiedene festgelegte Sätze (s. 4.2.2) ein Intent aufgerufen. Ein Satz besteht immer aus einem statischem Teil und Optional aus einem Slot (s. 4.2.1), dieser Slot ist wie eine Variable zu sehen. Das bedeutet zum Beispiel, wenn der Nutzer nach dem Wetter frägt ist der Statische teil „Wie wird das Wetter in“ und der Slot dann der Städtename. Daraufhin wird der Intent aufgerufen, welcher für diesen Satz eingetragen wurde (3.1) und als Slot wird der Städtename mit übergeben.

## Inhalt Hauptteil (index.js)

Die Index.js wurde von Template bereitgestellt (s. 2.2) und besteht aus den Funktionen um den Server zu initialisieren, eine Funktion welche ausgeführt wird, wenn der Skill aufgerufen wird und eine Funktion welche die Intents lädt.

Konstruktor:

Der Konstruktor wird aufgerufen, wenn der Server startet dort wird der Port und der Name der App gesetzt, anschließend werden die anderen Funktionen in der index.js aufgerufen.

Server Initialisieren:

In der Funktion initializeServer() wird aufgerufen beim Start des Servers. In dieser Funktion wird Expressjs als Server gesetzt und der Port geladen, dies ist vom Template so vorgeneriert. Es musste zusätzlich gesetzt werden welche Fehlernachricht ausgegeben wird, wenn kein Passender Intent gefunden wurde.

this.app.messages.NO\_INTENT\_FOUND= "Selbsterstelle Fehlernachricht";

Code Intent nicht gefunden Nachricht setzen

Skill Aufruf:

Jedes Mal wenn ein Nutzer den Skill aufruft wird die Funktion initializeApp() aufgerufen. Hier wird die Funktion festgelegt welche ausgeführt werden soll, wenn der Skill aufgerufen wird. In dieser Funktion können erste Session Attribute gesetzt werden oder die Verbindung zu einer API hergestellt werden. Es muss hier allerdings nichts passieren doch sollte ein Satz für Alexa zum Ausgeben zurückgegeben werden, dass der Nutzer erkennen kann das die App gestartet ist.

initializeApp() {

    this.app = new alexa.app(this.name);

    this.app.launch(function(req, res) {

        var session = req.getSession();

        res.shouldEndSession(false);

        res.say('Willkommen bei Alexinator, dem Spiel das deine Gedanken lesen kann');

        session.set('status','launch');

    });

  }

Code Funktion bei Aufruf des Skills

Intents laden:

Zum Start des Server werden die Intents geladen dazu dient die Funktion loadIntents(). Diese lädt alle JavaScricpt Dateien aus dem Ordner „Intents“ und Macht sie der App bekannt. Dazu wird das Modul Glob verwendet welches alle Dateien eines bestimmten Patterns lädt und in einem Array namens files bereitstellt.

 loadIntents() {

    var intentPath = './intents'; // Legt den Pfad fest

    glob('\*\*.js', {cwd: './intents'}, function (er, files) {

      for (var i = 0; i < files.length; i++) {

        var currentIntent = new (require(intentPath+'/'+files[i]))();

        this.app.intent(currentIntent.name, {

          slots: currentIntent.slots,

          utterances: currentIntent.utterances

        },currentIntent.execute.bind(currentIntent));

      }

    }.bind(this));

  }

Code Intents laden

## Intents

Ein Intent stellt die Antwort, des Servers, auf eine bestimmte Anfrage da [Amazon 2017]. Ein Intent besteht dabei aus mehreren Komponenten welche zusammen einen Intent darstellen.

Ein Intent besteht aus 3 Komponenten:

* Slots
* Utterances
* Funktion

Slots und Utterances werden jeweils im Kapitel 4.2.1 und 4.2.2 erklärt, der dritte Punkt Funktion stellt die Logik dar welche ausgeführt wird, wenn der Intent aufgerufen wird. In dieser Funktion stehen die 2 Objekte Request und Response zur Verfügung. Diese Objekte werden vom Framework Alexa-App bereitgestellt und verfügen über alle Methoden dieses Frameworks [Alexa-App 2017].

Daraus folgt ein Intent wird im Code folgendermaßen umgesetzt, hier als Beispiel eines Wetter Intents:

module.exports = class Intent {

  constructor() {

    this.name = 'Wetter'; // Name des Intents

    this.slots = { // Slots des Intents

      'city': 'city'

    };

    this.utterances = ['Wie ist das Wetter in{city}'],

                      ['Ist das Wetter in {city} schön'];

  }

  execute(req, res) {

        // Hier kommt die Logik des Programms

        wetter = getWetter(req.slot('city’)).toString(); //Gibt Wetter für die gesagte Stadt zurück

        res.say(wetter);

      }

    );

  }

}

Code Beispiel Intent Aufruf

Daraus folgt, wenn der Nutzer nach starten des Wetter-Skills zu Alexa sagt „Wie wird das Wetter in Coburg“, wird der Skill mit dem Namen Wetter aufgerufen und als Wert des Slots wird „Coburg“ übergeben.

### Utterances

* Was ist das
* Wozu ist das
* Wie wird es im Template umgesetzt

### Slots

Slots geben die Möglichkeit einen Utterance (s. 4.2.1) mit einem Variablen Wort zu belegen. Das heißt ein Slot ist eine Liste von Wörtern oder Ausdrücken welche von Amazon in verschiedenen Varianten vorgegeben wird oder selbst definiert werden kann. Slots werden in einem Utterance mit geschweiften Klammern angegeben z.B. {slotName}. [Amazon 2017].

Der Vorteil eines Slots ist, es müssen weniger Utterances definiert werden. Zum Beispiel die Frage nach dem Wetter „Wie wird das Wetter in Coburg“, dabei müsste ein eigner Utterance für jede Verfügbare Stadt angelegt werden. Wenn der Stadtname durch einen Slot ausgetauscht wird bleiben die Utterances übersichtlich. Desweitern müsste für jede Variation des Satzes und jede Stadt ein eigener Utterance festgelegt werden. Mithilfe eines Slots namens „city“ könnten folgende Utterances entstehen:

* „Wie wird das Wetter in {city}
* „Brauche ich in {city} heute einen Regenschirm“
* „Wird das Wetter in {city} heute schön“

Ein weiterer Vorteil ist, ein Slot steht durch das Alexa-App Framework als Variable zu Verfügung mit req.slot(slotname) kann dieser ausgelesen werden. Dies vereinfacht die Verarbeitung von User eingaben deutlich da kein String durchsucht werden muss.

Die Werte die Slots einnehmen können werden, über Listen definiert dabei bietet Amazon selbst viele vordefinierte Listen oder diese Listen können selbst definiert werden (s. 3.1) [Amazon 2017].

# Implementierung des Skills

Unter Kapitel 2 wurde beschrieben welche Tools gebraucht werden um diesen Alexa-Skill zu entwickeln. In Kapitel 3 wurde beschrieben wie ein Alexa-Skill veröffentlicht werden kann. Kapitel 4 hat beschrieben wie ein Skill aufgebaut werden kann. In diesem Kapitel wird beschrieben wie der Alexa-Skill „alexinator“ entwickelt wurde. Dazu gehört welche Vorüberlegungen getroffen wurden um den Alexa-Skill zu entwickeln, und welche Intents erstellt wurden um den Skill dann umzusetzen. Dazu gehört auch welche Slots und Utterances für die jeweiligen Intents gebraucht werden. Es wird auch erklärt welche Änderungen in der „index.js“ Datei vorgenommen wurden welche als Startpunk der App dient. Als letzten Schritt wird erklärt wie vorgegangen wurde um den die Verbindung zu Akinator-API herzustellen.

## Konzept

### Programmablauf

### Programmaufbau

## Hauptprogramm

## Intent StartGame

### Slots

### Utterances

## Intent GameRound

### Slots

### Utterances

## Intent Help

### Slots

### Utterances

## API Funktionen

## Template

Als Vorlage wurde ein von Yeoman (s. 2.2) bereitgestelltes Template verwendet. Hier soll kurz beschrieben werden welches Template verwendet wurde und welche Anpassungen gemacht werden mussten.

Um das Template für dieses Projekt zu nutzen musste folgender Code in der Konsole ausgeführt werden [Jseijas 2017]:

npm install –g yo

npm install –g generator-alexa

yo alexa

Code Installieren des Templates

Da das Template nicht alle aktuellen Versionen der Verwendetenten Modulle Enthält müssen in der Datei „package.json“ noch Änderungen vorgenommen werden. In der Datei müssen unter dem Punkt „dependncies“ die Versionen für alexa-app (auf den momentan aktuellsten Stand von 2.3.4 auf 4.0.0) aktualisiert werden. Außerdem sind müssen in der Index Datei folgende Zeilen in der Funktion initializeServer() geändert werden.

Löschen:

this.server.use(bodyParser.urlencoded({extended: true}));

this.server.use(bodyParser.json());

Code Template Änderung löschen

Hinzufügen:

var router = express.Router();

this.app.express({ router: router, checkCert: false });

Code Template Änderung hinzufügen

Alle weiteren Anpassungen werden in 5.2 aufgezählt.

## Testen

In diesem Kapitel wird behandelt wie der Alexa-Skill während der Entwicklung getestet wurde. Dies wurde mit dem Programm „Postman“ (s. 2.5) gemacht, in diesem Kapitel wird beschrieben wie mit Postman ein Alexa-Skill lokal getestet werden kann. Da in diesem Skill mit der Akinator-API kommuniziert wird, müssen Variablen gesetzt werden, welche sich von Request zu Request ändern, wie beispielsweise die Frage welche von der Akinator-API gegeben wird. Für die Anwendung von Postman wird sich auf die Dokumentation von Postman berufen [Postman 2017].

Für die Kommunikation zwischen AVS und dem Skill welcher auf dem Privaten Server läuft werden HTTP-Requests genutzt welche JSON Objekte austauschen. Um den Skill auf dem eigenen Rechner zu entwickeln muss der AVS-Dienst simuliert werden d.h. die JSON-Pakete welche AVS an den Server schickt werden von Postman simuliert.

Als erstes muss in Postman eine neue Umgebung (Environment) angelegt werden, diese wird hier Alexinator genannt, in dieser müssen Folgende 5 Variablen angelegt werden.

| **Name** | **Wert** |
| --- | --- |
| url | localhost:5000/echo/alexinator  Enthält die URL an die gesendet wird |
| id | EMPTY  Enthält die ID welche von der Akinator-API zu gewiesen wird |
| sig | EMPTY  Enthält die Signatur welche von der Akinator-API zugewiesen wird |
| qu | EMPTY  Enthält die Frage welche die Akinator-API sendet |
| st | EMPTY  Enthält die Schritt Nummer welche von Akinator-API gesendet wird. |
| stat | EMPTY Enthält den Status des Laufenden Spiels |
| nameGuess | EMPTY Enthält den Namen der Person welcher geraten wurde. |
| picURL | EMPTY Enthält die URL zu dem Bild der Person die erraten wurde. |

Tabelle Postman Variablen

Um Variablen zu nutzen müssen diese im Format {{name}} geschrieben werden. Postman ersetzt automatisch diesen Ausdruck durch den Gesetzten Wert.

Im Folgenden wird der Request für den LaunchRequest dargestellt und ausführlich beschrieben auch wie Tests für diesen Request aussehen und wie Variablen genutzt werden welche in den folgenden Requests wiederverwendet werden können.

In einen neuen Tab muss in der URL Zeile POST als HTTP-Methode ausgewählt werden und das Textfeld mit {{url}} befüllt werden. Als nächstes muss der Body des Requests gesetzt werden. In diesem müssen Session Daten gesetzt werden (hier rein fiktive Werte, da diese Sonst von AVS gesetzt werden) in der „request“ Sektion wird festgelegt welche Anfrage an den Server gestellt wird. Der Launch Request für den Alexa-Skill sieht wie folgt aus:

{

"version": "1.0",

"session": {

"new": true,

"sessionId": "jmsdfghjkgffghjk8mnbvcvbhnjmm",

"application": {

"applicationId": "gthkh"

},

"attributes": {

"string": {}

},

"user": {

"userId": "123",

"accessToken": "av312"

}

},

"context": {

"System": {

"application": {

"applicationId": "abcdef"

},

"user": {

"userId": "123",

"accessToken": "av312"

}

}

},

"request": {

"type": "LaunchRequest",

"requestId": "string",

"timestamp": "2017-04-05T20:14:30+01:00",

"locale": "de-DE"

}

}

Code 9 Request LaunchRequest nach [Amazon 2017]

Der Alexinator-Skill wird mit folgendem JSON Objekt Antworten, die Struktur wird von dem Framework Alexa-App (s. 2.4) generiert.

{

  "version": "1.0",

  "response": {

    "directives": [],

    "shouldEndSession": false,

    "outputSpeech": {

      "type": "SSML",

      "ssml": "<speak>Willkommen bei Alexinator, dem Spiel das deine Gedanken lesen kann</speak>"

    }

  },

  "sessionAttributes": {

    "string": {},

    "status": "launch"

  }

}

Code 10 Postman Antwort Launsh Request

In dieser Antwort werden auch die Session Attribute mitgesendet welche in späteren Requests wieder zu finden sind. Da diese nicht von einem Request in den nächsten Request Kopiert werden sollen werden diese in Variablen gespeichert welche Postman bereitstellt. Dies geschieht im Reiter Tests dort werden drei Dinge getan. Als Erstes wird aus dem Response (Code 2) die Werte aus den Sektion sessionAttributes gelesen, als zweiter Schritt werden diese Werte in die Enviroment-Variablen (s. Tabelle 1) gespeichert. Als letztes werden Tests definiert welche überprüft ob diese Variablen gesetzt sind. Daraus folgt der Code:

var jsonData = JSON.parse(responseBody);

// Lesen der Variablen

var id= jsonData.sessionAttributes.akinatorSession;

var sig= jsonData.sessionAttributes.akinatorSignature;

var qu= jsonData.sessionAttributes.akinatorQuestion;

var st= jsonData.sessionAttributes.akinatorStep;

// Speichern der Variablen

postman.setEnvironmentVariable("id", id );

postman.setEnvironmentVariable("sig", sig );

postman.setEnvironmentVariable("qu", qu );

postman.setEnvironmentVariable("st", st );

// Testen der Variablen

tests["got ID"] = id !== null;

tests["got sig"] = sig !== null;

tests["got question"] = qu !== null;

Code 11 Postman Test Launsh Request

Nach ausführen des HTTP-Requests sind die Variablen gesetzt und können in Späteren Requests genutzt werden.

Als nächstes wird ein Beispiel für einen Intent (s. 4.2) Aufruf gezeigt. Dazu werden die Variablen welche im Vorherigen Request gesetzt wurden genutzt und aktualisiert.

Ein Intent Aufruf von AVS an den Server enthält die Session Attribute, welche vom vorherigen Request (s. Code 2) gesetzt wurden die Session Daten welche von Amazon gesetzt werden und in der Sektion „request“ wird angegeben welcher Intent (4.2) mit welchen Slots (4.2.1) gesendet wird.

{

  "version": "1.0",

  "session": {

    "new": true,

    "sessionId": "jmsdfghjkgffghjk8mnbvcvbhnjmm",

    "application": {

      "applicationId": "gthkh"

    },

    "attributes": {

      "string": {},

     "akinatorSession": "{{id}}",

     "akinatorSignature": "{{sig}}",

     "akinatorStep": "{{st}}",

     "akinatorQuestion": "{{qu}}"

     "status": "{{stat}}",

     "akinatorName": "{{nameGuess}}",

     "akinatorPicURL": "{{picURL}}"

    },

    "user": {

      "userId": "123",

      "accessToken": "av312"

    }

  },

  "context": {

    "System": {

      "application": {

        "applicationId": "abcdef"

      },

      "user": {

        "userId": "123",

        "accessToken": "av312"

      }

    }

  },

  "request": {

  "type": "IntentRequest",

  "requestId": "string",

  "timestamp": "2017-04-05T20:14:30+01:00",

  "locale": "de-DE",

  "intent": {

    "name": "GameRound",

    "slots": {

"answer": {

        "name": "answer",

        "value": "ja"

}

    }

  }

}

}

Code Postman Intent-Request

Zum Aktualisieren der Step Number und der aktuellen Frage wird über Reiter Tests der HTTP-Anfrage gemacht.

var jsonData = JSON.parse(responseBody);

var stat= jsonData.sessionAttributes.status;

var nameGuess = jsonData.sessionAttributes.akinatorName;

var picURL = jsonData.sessionAttributes.akinatorPicURL;

postman.setEnvironmentVariable("qu", qu );

postman.setEnvironmentVariable("st", st );

postman.setEnvironmentVariable("stat", stat );

postman.setEnvironmentVariable("nameGuess", nameGuess );

postman.setEnvironmentVariable("picURL", picURL );

tests["got question"] = qu !== null;

tests["got step"] = st !== null;

tests["got status"] = stat !== null;

Code Postman aktualisieren der Variablen

Nachdem der Intent-Request gesendet wurde, wird im Alexa-Skill der Passende Intent aufgerufen dafür ist das Framework Alexa-App (2.4) verantwortlich.

Nach erstellen der Verschiedenen HTTP-Requests in Postman, wenn der Server läuft getestet werden. Dazu muss nur noch „Senden“ Button gedrückt werden. In diesem Projekt muss die Reihenfolge beachtet werden, zuerst muss ein Launch-Request gesendet werden, anschließend ein StartGame-Request und als letztes der GameRound-Request. Der GameRound-Request kann immer wieder aufgerufen werden, bei Bedarf mit einer anderen Antwort im Antwort-Slot, bis das Spiel beendet ist.

Unter diesem Punkt wurde das Tool Postman vorgestellt und erklärt, es bietet den Vorteil Lokal auf dem eignen Rechner Entwickeln und Testen zu können. Diese Unabhängigkeit ist wichtig da nicht für jede Änderung, welche auch Fehler erzeugt könnte, das Projekt auf den Server geladen werden muss. Auch ist man für die Entwicklung nicht Abhängig von einer Internet Verbindung und kann alles lokal Testen.

# Zusammenfassung und Ausblick

* Was wurde erreicht
* Was wurde nicht erreicht

# Literaturverzeichnis

[Amazon 2017] Amazon Developer <https://developer.amazon.com/alexa>  
(Zugriff 17.5.2017).

[Postman 2017] Postman https://www.getpostman.com/docs/  
(Zugriff 17.5.2017).

[Nodejs 2017] Nodejs https://nodejs.org/en/  
(Zugriff 19.5.2017).

[Alexa-App 2017] Framework Alexa-App https://www.npmjs.com/package/alexa-app  
(Zugriff 19.5.2017).

[Yeoman 2017] Template Programm für Alexa-Skill https://www.npmjs.com/package/alexa-app  
(Zugriff 19.5.2017).

[MDN 2017] Promises https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global\_Objects/Promise  
(Zugriff 19.5.2017).

[Jseijas 2017] Template https://github.com/jseijas/generator-alexa#readme  
(Zugriff 12.6.2017).

[android 2017] Template http://www.androidauthority.com/bixby-vs-google-assistant-vs-siri-763201/  
(Zugriff 14.6.2017).

https://github.com/jseijas/generator-alexa#readme

[Duden 2014] Bibliographisches Institut Dudenverlag <http://www.duden.de>  
(Zugriff 17.9.2014).

[IDS2011] Dokumente zu den Inhalten der Rechtschreibreform, Überarbeitetes Regel Werk 2011 <http://www.ids-mannheim.de/reform/regeln2006.pdf> (Zugriff 18.3.2015).

[Katalog 2014] Katalog der Deutschen Nationalbibliothek <https://portal.dnb.de/opac.htm> (Zugriff 13.9.2014).

[Schmidt+2008] Kories, R.; Schmidt-Walter, H.: Taschenbuch der Elektrotechnik, 8. Aufl., Verlag Harri Deutsch, Frankfurt, 2008.

# Glossar

Alexa-Skill Eine Anwendung welche auf von Amazon zertifizierten Geräten ausgeführt wird