# Implementierung

## Einrichtung

### Raspberry Pi

Zunächst muss der Raspberry Pi eingerichtet werden [https://www.w3schools.com/nodejs/nodejs\_raspberrypi.asp]. Die Entscheidung für die Geräteauswahl fiel auf den Raspberry Pi 3 Model B. Das Projekt ist aber auch auf alle anderen Raspberry Pi Modellen portierbar.

Für eine einfache Bedienung des Raspberry Pi kommt in diesem Projekt das „Raspbian Stretch with desktop“ [https://www.raspberrypi.org/downloads/raspbian/] zum Einsatz. Solange die Ansteuerung der GPIO Ports gewährleistet ist, kann auch ein beliebiges anderes Betriebssystem auf dem Raspberry Pi eingerichtet werden.

Um ein Betriebssystem für den Raspberry Pi zu installieren, muss das Betriebssystem auf der MicroSD-Card des Raspberry Pi installiert („geflasht“) werden. Am einfachsten geht dies mit dem kostenlosen Programm Etcher [https://etcher.io/]. Etcher ist für Windows, Linux und MacOS erhältlich. In Etcher muss man lediglich das Medium (die MicroSD-Card) sowie ein Image zur Installation auswählen und die Installation starten. Sobald Etcher das Betriebssystem auf dem Raspberry Pi installiert hat, kann die MicroSD-Card in den Raspberry Pi gesteckt werden und von dort direkt gebootet werden.

Der Raspberry Pi kann entweder über direkt angeschlossene Peripheriegeräte bedient werden oder per Remote. Für die Bedienung über eine SSH empfiehlt sich Putty [https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html]. In unserem Projekt wurde außerdem RealVNC [https://www.realvnc.com/de/] verwendet, welches auf dem Raspbian Stretch bereits vorinstalliert ist. RealVNC hat gegenüber SSH den Vorteil, dass der gesamte Desktop über Remote bedient werden kann.

### Python

Installierte/benötigte Python Version für elropi.py?

### Node.js

Für das Projekt wird Node.js in Version 9.x verwendet. Um das aktuellste Node.js v9.x zu installieren führt man folgende Befehle aus [https://nodejs.org/en/download/package-manager/]:

curl -sL https://deb.nodesource.com/setup\_9.x | sudo -E bash -

sudo apt-get install -y nodejs

### Cloud9

[Extra? Ausführlicher?]

Für die Entwicklung auf dem Raspberry Pi wurde die Web-DIE Cloud9 [https://github.com/c9/, https://dev.to/jtlunsford/setup-javascript-ide-on-pi-3 ] verwendet, welche auf dem Raspberry Pi als Server installiert ist. Die Web-IDE ermöglicht besonders einfaches Entwickeln und Testen über jeden Webbrowser auf Computern im selben Netzwerk wie der Raspberry Pi.

### Git

Zur Sicherung und Versionierung wurde ein GitHub Repository eingerichtet.

git clone <https://github.com/RobinWarth/Smart-Home-Solutions.git>

git remote add origin <https://github.com/RobinWarth/Smart-Home-Solutions.git>

### Amazon Developer Konto

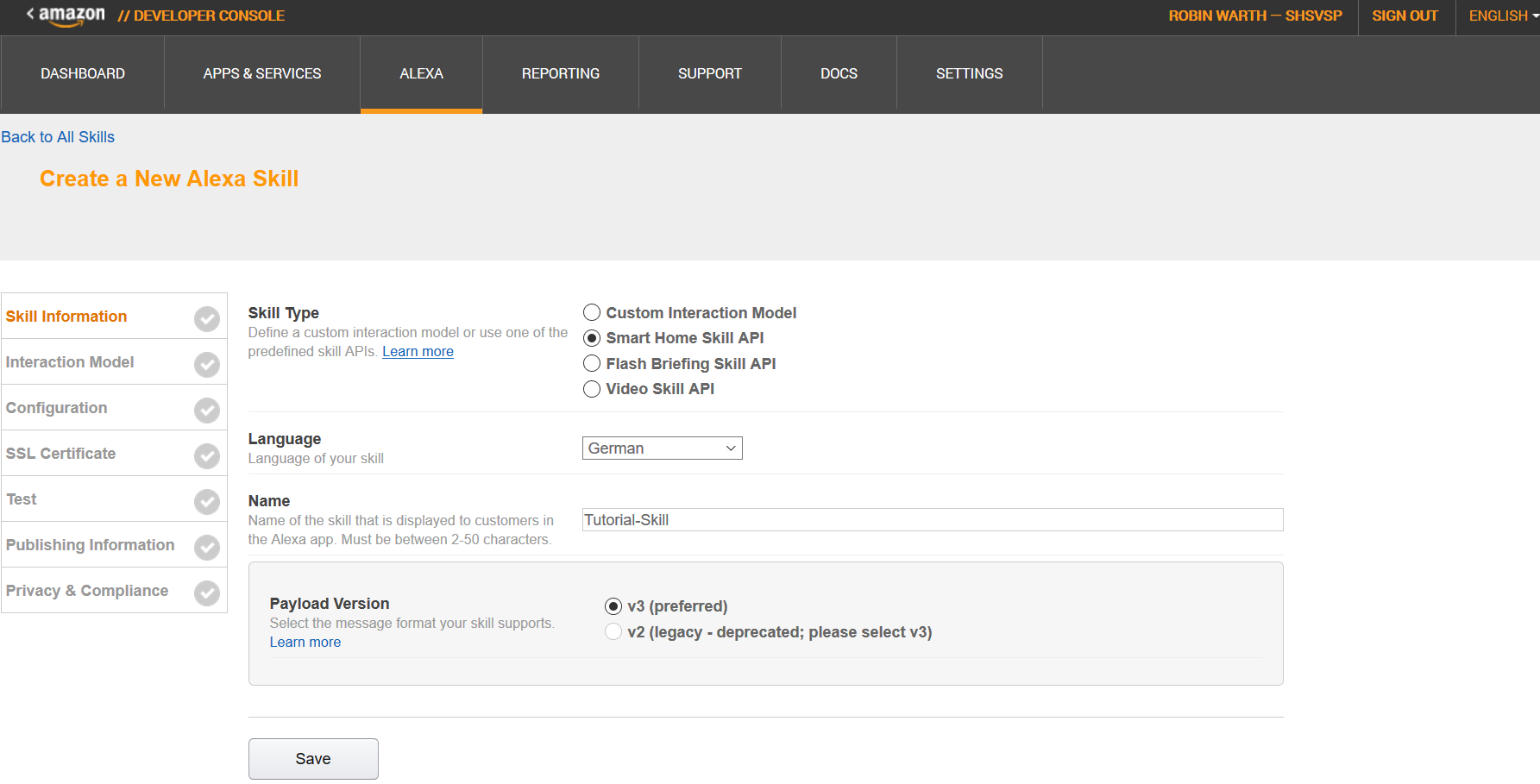
Für das Erstellen von Alexa Skills muss ein Amazon Developer Konto erstellt werden [Link].

### Amazon Webservices (AWS) Lambda

Um einen Alexa Skill mit der Smart Home Skill API zu erstellen, wird ein AWS Lambda Server vorausgesetzt. Hierfür muss ein Konto für AWS erstellt und ein AWS Lambda Server gemietet werden.

## Alexa Skill

Bilder - Screenshots: [https://developer.amazon.com/edw/home.html#/skill/amzn1.ask.skill.920e5e6a-89da-4f0e-9728-69870ca80c88/de\_DE/info]

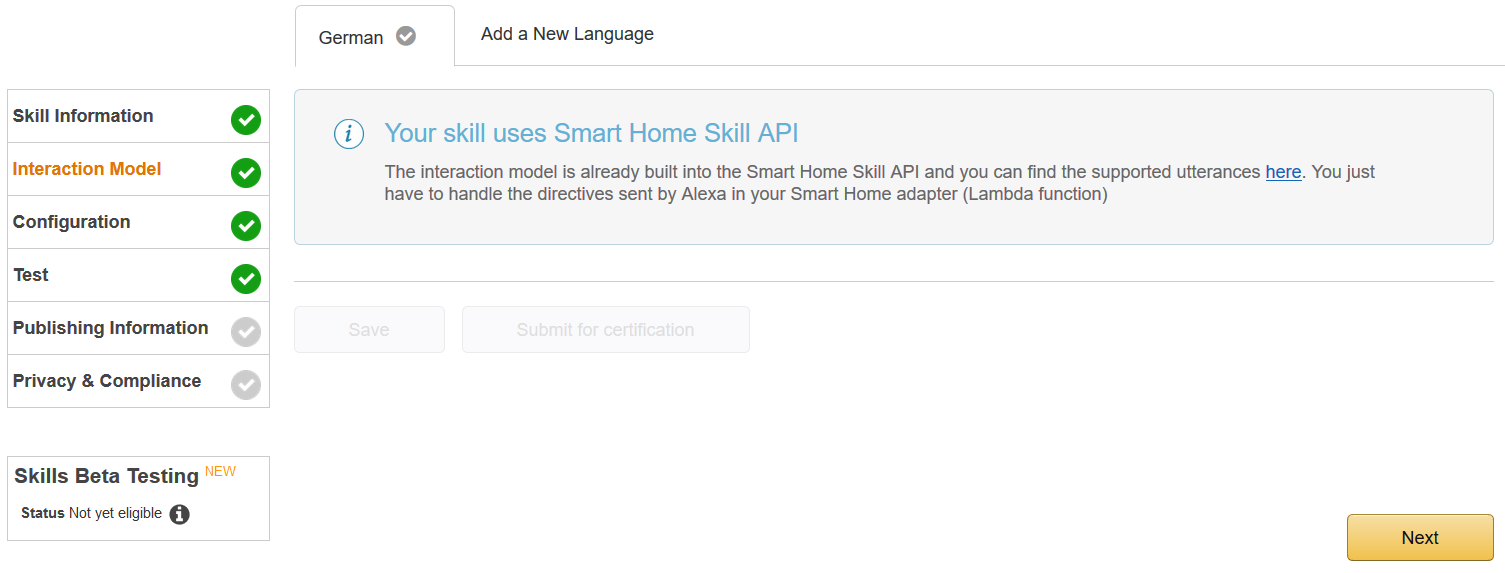


[bei Konzept Punkt einführen zum Abwägen zwischen Skill Typen]

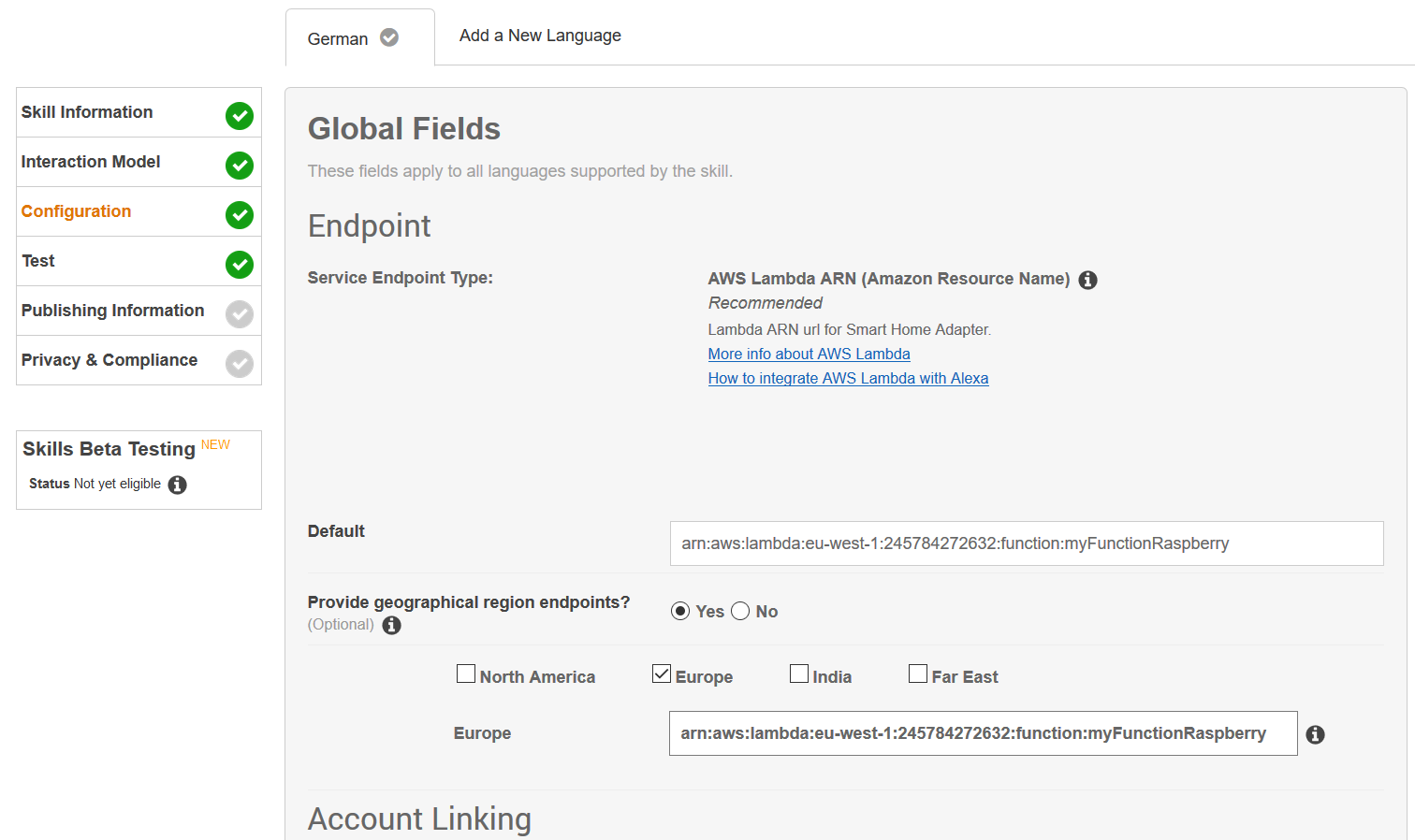
Zunächst einmal müssen im ersten Schritt unter „Skill Information“ grundlegende Informationen über den Skill festgelegt werden, welche die weiteren Schritte und deren Komplexität beeinflussen.

Wie schon in [Konzept -> Skilltypen] beschrieben, fiel die Entscheidung auf einen Skill, welcher die „Smart Home Skill API“ unterstützt. Außerdem müssen in diesem Schritt noch die unterstützten Sprachen des Skills sowie ein Skill Name festgelegt werden.

Die Payload Version bestimmt in welcher Form und Stiel Nachrichten im JSON-Format zwischen Skill, Alexa Smart Home API und Server ausgetauscht werden sollen. Dies wird im weiteren Verlauf noch einmal erneut aufgegriffen, wenn der Aufbau und Stil der JSON Nachrichten weiter erläutert wird. [Verweis auf das Kapitel] Auch wenn hier noch zusätzlich die Auswahl einer Payload Version aufgeführt ist, so lässt sich doch nur die Payload Version „v3“ auswählen. Payload Version „v2“ ist zum Zeitpunkt dieser Studienarbeit schon so veraltet, dass man sie nicht einmal auswählen kann. Sie ist nur für ältere schon erstellte Skills noch aufgeführt.



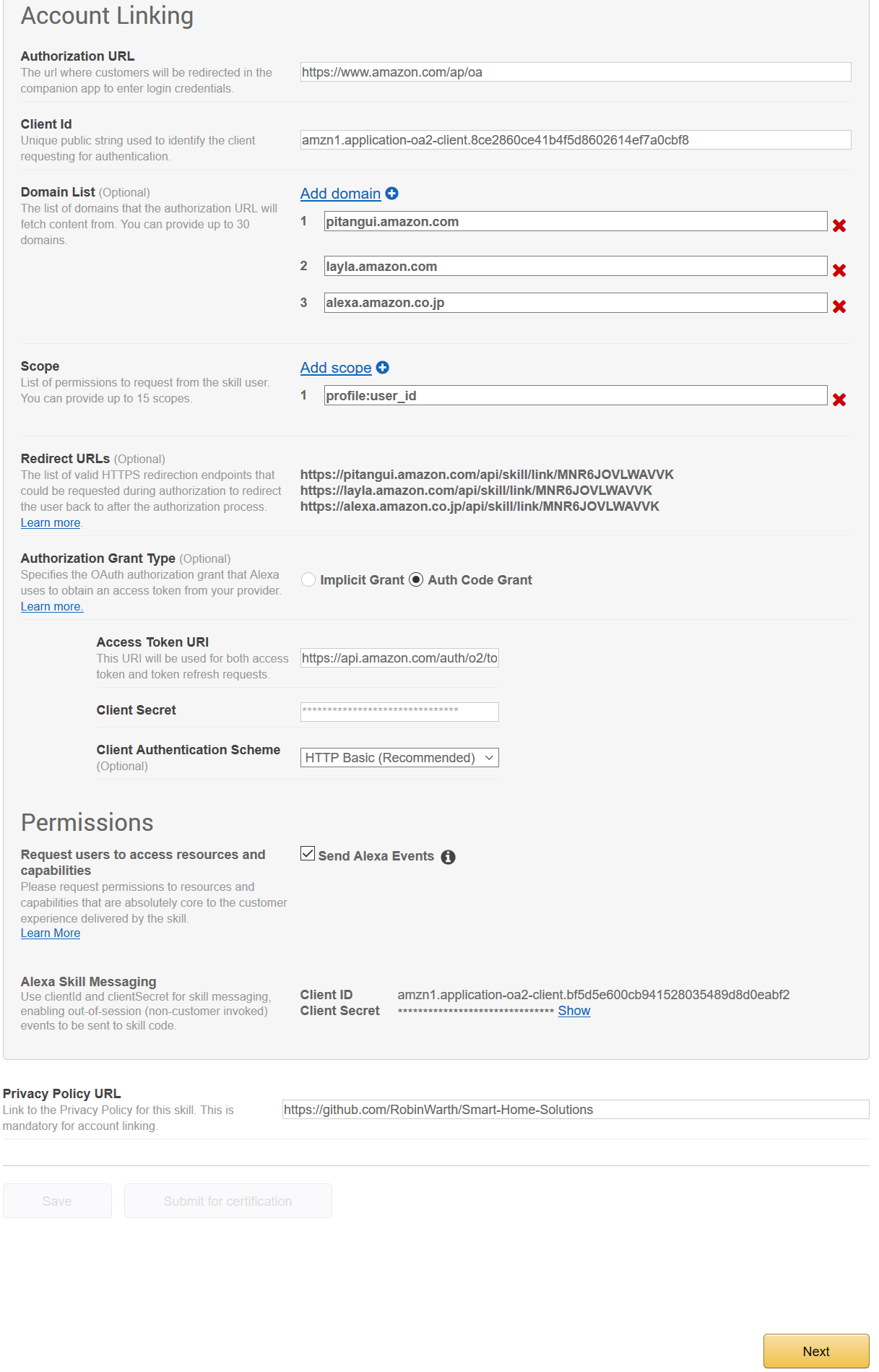
Da in diesem Skill die „Smart Home Skil API“ zum Einsatz kommt, muss in dem Schritt „Interaction Model“ nichts weiter getan werden. Das Interaktionsmodell ist bereits in die API eingebaut.



Im Schritt „Configuration“ legt man zunächst den Service Endpunkt fest, welcher die von Alexa gesendeten Direktiven geeignet verarbeiten kann. Da hier die „Smart Home Skill API“ zum Einsatz kommt, wird man von Amazon dazu verpflichtet einen „Amazon Web Services“ (kurz „AWS“) „Lambda“ Server zu nutzen. Zu dem Aufbau und der Einrichtung des „AWS Lambda“ Server wird in Abschnitt [bla bla keine Ahnung, AWS Lambda] noch gezielt eingegangen.

Hat man aber statt der „Smart Home Skill API“ das „Custom Interaction Model“ als Skill Typ gewählt, hätte man anstatt dem AWS Lambda Server auch jeden beliebigen anderen zertifizierten HTTPS Webserver angeben können.

Bei Amazon Webservices wie dem Lambda Server wird jedem Server eine eindeutige Adresse zur Ressource und der darin enthaltenen Funktionen zugeteil. Diese Adresse nennt sich „Amazon Resource Name“ (kurz: ARN). Die ARN unseres Webservice lautet: „arn:aws:lambda:eu-west-1:245784272632:function:myFunctionRaspberry“. In der ARN sind Informationen über den Server Typ (hier: „Lambda“), über den Server Standort (hier: „eu-west“, Irland) und die Funktion (hier: „myFunctionRaspberry“) enthalten, welche den Endpunkt repräsentiert und die Direktiven von Alexa entgegennimmt.

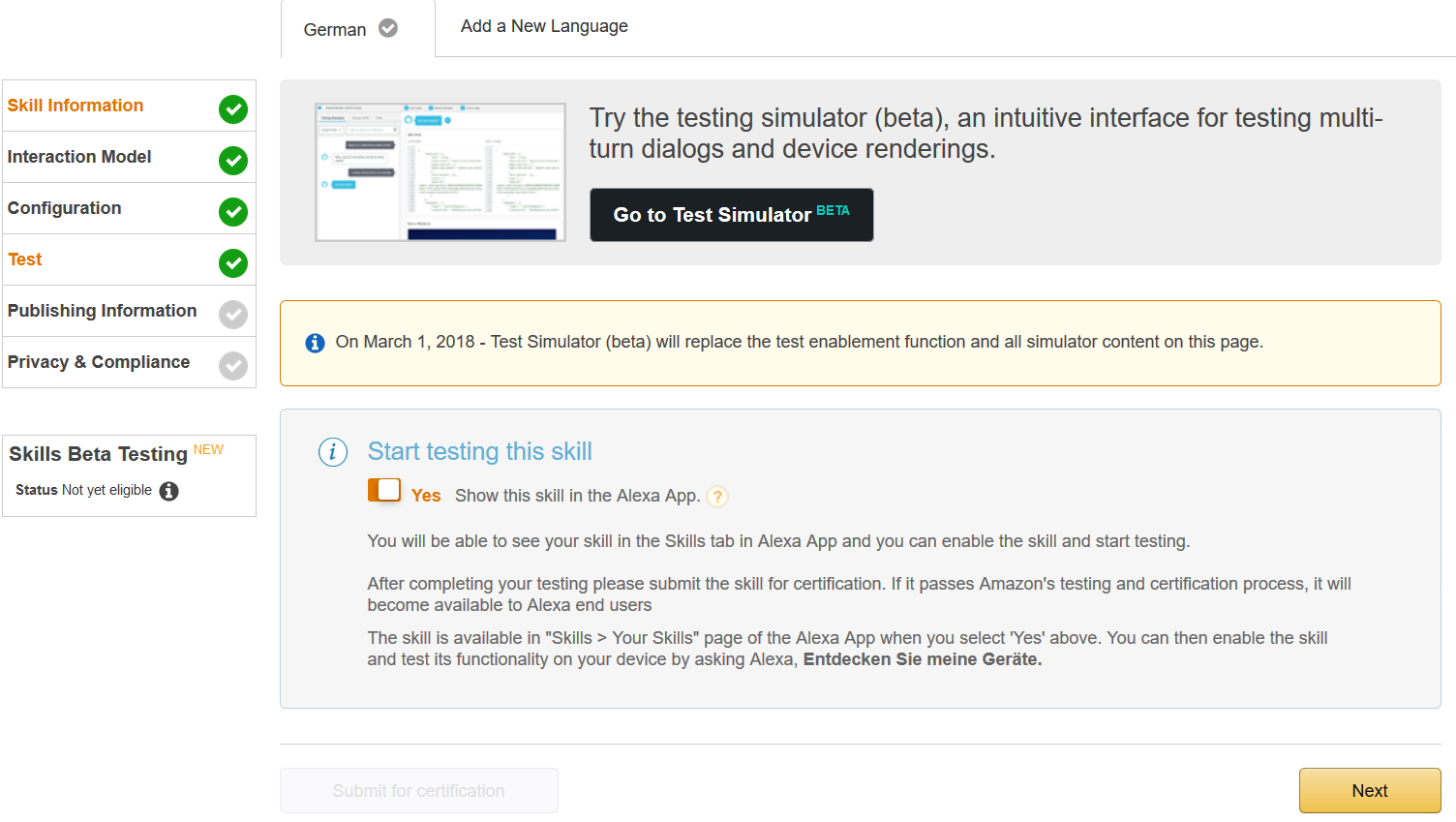


Das Account Linking ist für Skills der „Smart Home Skill API“ zwingend notwendig, aber für Skills mit „Custom Interaction Model“ ist dies optional. Das Account Linking erfolgt mit Hilfe des OAuth2 Authentifizierungsstandard [https://oauth.net/2/, mehr dazu?].

Das Account Linking ist notwendig, um jeden Benutzer des Skills eindeutig identifizieren zu können. Im Allgemeinen wird hier eine Verknüpfung zwischen Alexa-Amazon-Account und dem Account beim Hersteller der Smart Home Geräte – wie zum Beispiel Lampen - geschaffen.

In diesem Project erfolgt das Account Linking mit einem Amazon Account. Die Einstellungen hier sind alle auf „Login with Amazon“ (kurz: „LWA“) getrimmt [https://developer.amazon.com/de/docs/login-with-amazon/web-docs.html, genauer beschreiben?]. Der Aufwand um einen guten, sicheren und verlässlichen OAuth2 Server selbst zu erstellen wäre für diese Studienarbeit zu hoch gewesen. Zudem muss der OAuth2 Server auch von Amazon akzeptiert sein und daher wären zusätzliche gültige Zertifikate notwendig. Die Möglichkeiten, welche durch Account Linking geschaffen werden, würden ohnehin nicht ausgereizt werden in dieser Studienarbeit und dem Einsatz Gebiet von dem hier erstellten Alexa Skill [https://developer.amazon.com/de/docs/smarthome/authenticate-an-alexa-user-account-linking.html].

…



Um den Skill mit der Alexa App auf dem Smartphone und einem Amazon Echo Gerät zu testen, muss nur der Schalter in Schritt „Test“ umgelegt werden. Der Skill wird nun für die mit dem Konto verbundenen Geräte automatisch zum Testen eingerichtet. Alternativ kann der Skill auch mittels, dem zur Zeit der Studienarbeit in der Beta-Phase befindlichen, Test-Simulator ausgeführt werden. Der Test-Simulator bietet dieselbe Möglichkeit zum Testen wie ein Echo-Gerät und sogar zusätzliche Funktionen. Im Test-Simulator kann direkt mit Alexa über Sprache oder geschriebenem Text interagiert werden. Wurde zu Beginn das „Custom Interaction Model“ statt der „Smart Home Skill API“ gewählt, lassen sich im Test Simulator auch die JSON Direktiven anzeigen, welche von Alexa oder dem Skill-Endpoint gesendet werden.

## Lambda Server

### Konfiguration

Allgemeines? => zu Grundlagen

Bevor der Lambda Server eingerichtet wird, muss ein Alexa Skill bereits erstellt sein. Bei der Einrichtung des AWS Lambda Server für den Alexa Skill wird die skill-ID benötigt. Ebenfalls ist ein AWS Account zwingend notwendig.

Nach der Anmeldung bei AWS wird zu Lambda navigiert. Für den Lambda Server ist es wichtig die richtige Region auszuwählen. Mögliche Regionen für Alexa Skills sind Asia Pacific (Tokyo), EU (Ireland), US East (N. Virginia) und US West (Oregon) [https://developer.amazon.com/de/docs/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html].

Als nächstes erstellt man eine Lambda Function um in dieser die, von Alexa gesendeten, Direktiven zu entgegen zu nehmen und zu verarbeiten.

Bei der Erstellung einer „Lambda Function“ kann man aus verschiedenen Blaupausen wählen oder eine leere Funktion, ohne Codebeispiel, wählen. Für den Einsatz der Lambda Function in dieser Studienarbeit hat zur Zeit der Bearbeitung leider keine passende Blueprint existiert, diese waren veraltet und haben eine alte „Payload-Version“ der Alexa Direktiven unterstützt. Dank Codebeispielen in der Online Dokumentation von Amazon Alexa kann die Lambda Funktion dennoch zügig einsatzbereit gemacht werden. Der Name der hier erstellten Lambda Funktion lautet „myFunctionRaspberry“.

Danach muss eine Rolle gewählt werden. Da zu diesem Zeitpunkt noch keine eigene definierte Rolle existiert. Für dieses Project wurde eine Rolle Namens „myRoleRaspberry“ erstellt mithilfe des Templates „Basic Edge Lambda permissions“.

Durch das betätigen des „Create Function“ Buttons wird die Funktion von AWS eingerichtet. Um auf Ereignisse zu reagieren, fügt man „Trigger“ zur Funktion hinzu. Standardmäßig wurde ein Trigger „Amazon CloudWatch Logs“ bereits der Funktion hinzugefügt. Dieser ist für das Logging zuständig. Jede Log Ausgabe wie zum Beispiel „console.log('test');“ landet somit automatisch in den „Amazon CloudWatch Logs“ mit Datum und eindeutigen Identifikatoren. Um die Lamda Funktion für die „Alexa Smart Home Skill API“ bereit zu machen, wird der „Alexa Smart Home“ Trigger ausgewählt. Dieser Trigger muss nun nur noch mithilfe der skill-ID bereit gemacht werden.

### Entwicklung

#### Export Handler

[insert exports.handler = … here]

Mit exports.handler wird die Funktion definiert welche ausgeführt wird, sobald eine Direktive an die Lambda Funktion gesendet wird. In dem Parameter „request“ stecken die Informationen, welche mit der Direktive gesendet werden. Der Parameter „context“ wird von AWS Lambda dazu verwendet Laufzeitinformationen zur ausgeführten Lamda-Funktion bereitzustellen.

Hier im export-handler werden die verschiedenen Typen an Direktiven zunächst einmal anhand von dem „header“ unterschieden und entsprechend weitere Aktionen durchgeführt.

#### Direktiven

Die Direktiven welche von Alexa gesendet werden sind Nachrichten im JSON-Format. Die Direktiven beschreiben Aktionen, welche ausgeführt werden sollen, sowie Informationen zu dem Client welcher die Aktion ausführen möchte.

Beispiele für Direktiven sind:

* Report State

[insert example JSON]

Über die „ReportState“ Direktive wird der Status der Smart Home Geräte abgefragt. Dies geschieht insbesondere dann, wenn der Benutzer über sein Smartphone in der Alexa App seine Geräte ansieht. In diesem Fall werden in kleinen zeitlichen Abständen kontinuierlich ReportState Direktiven gesendet.

* Discover

[insert example JSON]

Über die „Discovery“ Direktive frägt Alexa nach verfügbaren Smart Home Geräten. Diese Direktive wird gesendet, wenn der Benutzer aktiv in der Alexa App nach Geräten suchen lässt.

* Power Controller

[insert example JSON]

Über „PowerController“ wird versucht aktiv einen Gerätestatus zu verändern. Zum Beispiel wenn der Benutzer sein Licht einschalten oder die Heizung aufdrehen möchte.