# Implementierung

## Einrichtung

### Raspberry Pi

Zunächst muss der Raspberry Pi eingerichtet werden [https://www.w3schools.com/nodejs/nodejs\_raspberrypi.asp]. Bei der Wahl des Geräts haben wir uns für einen Raspberry Pi 3 Model B entschieden. Im Prinzip müsste das Projekt aber auch auf alle anderen Raspberry Pi Modellen portierbar sein.

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 man das Betriebssystem auf die 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 haben wir 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 haben wir 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 wurde. 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 haben wir 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

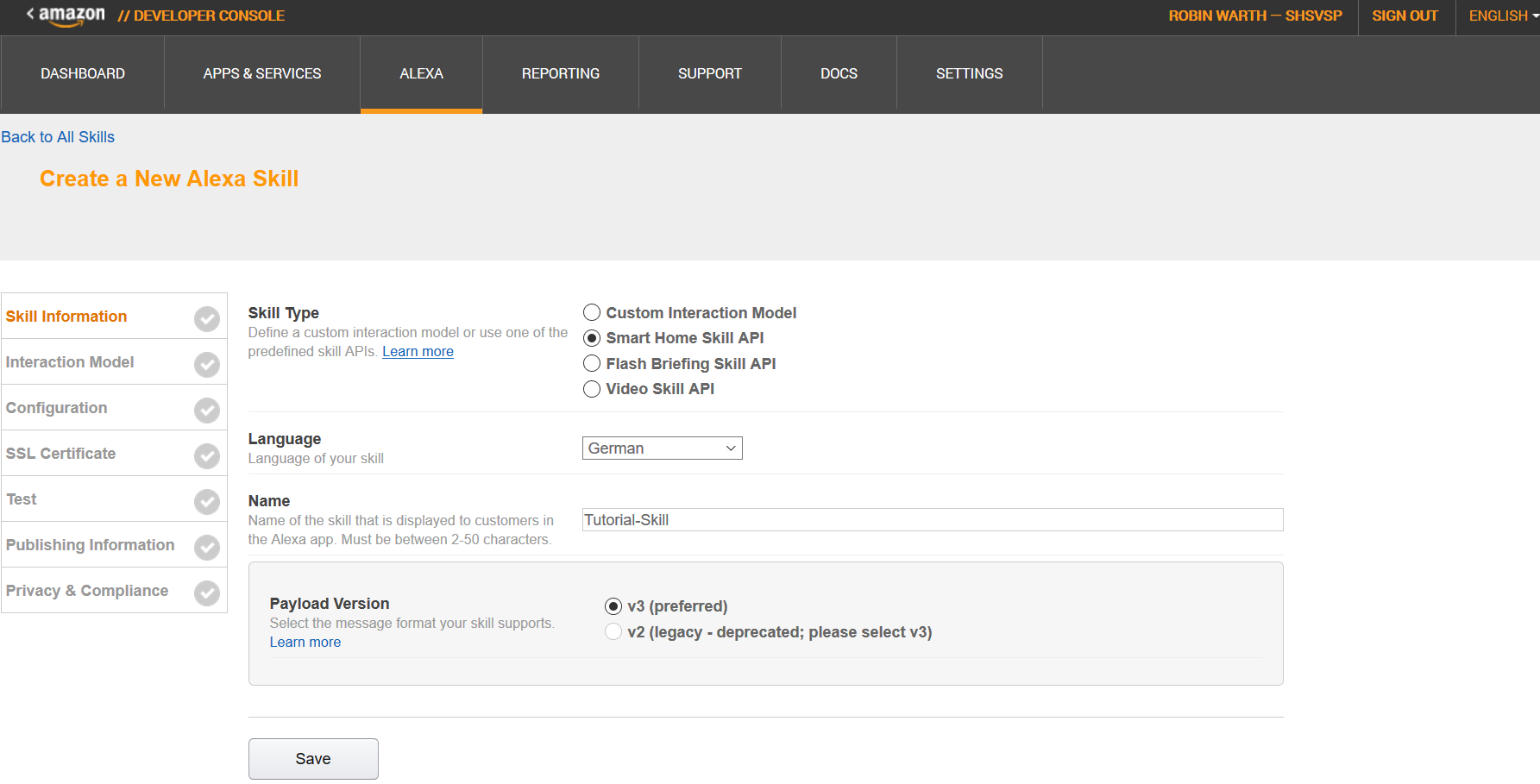
Für das Erstellen von Alexa Skills muss ein Amazon Developer Konto erstellt werden [Link].

### 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]

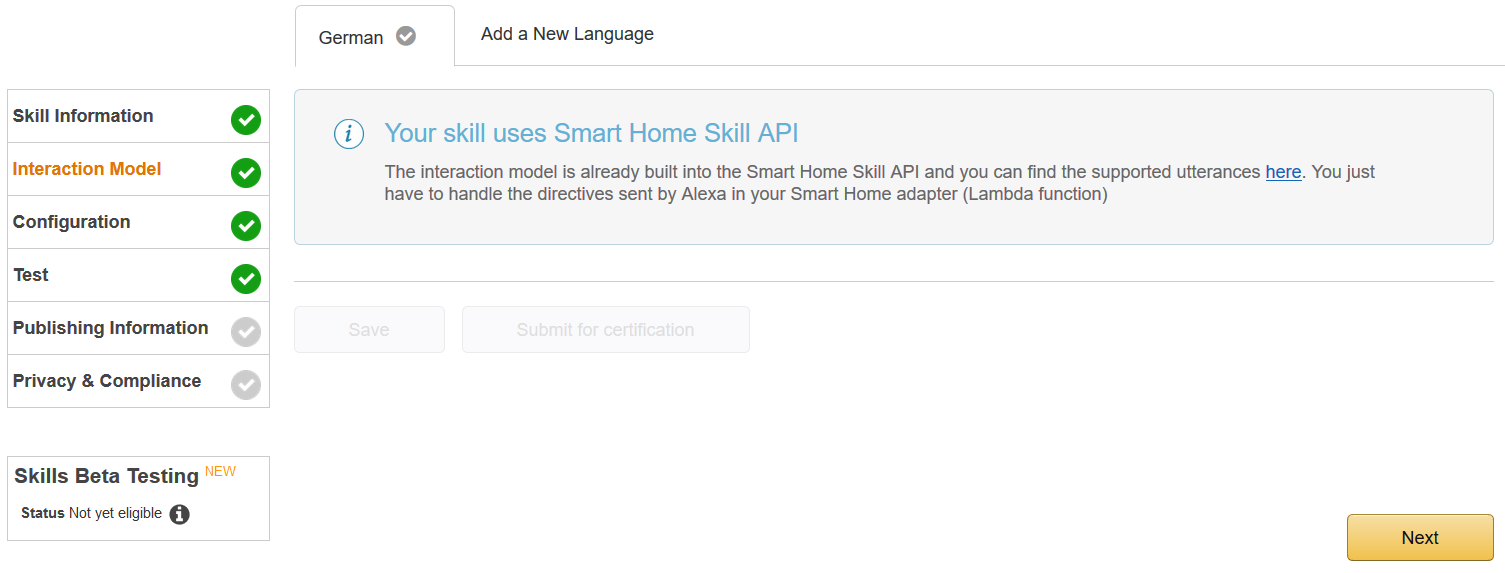


[bei Konzept Punkt einführen zum Abwägen zwischen Skill Typen]

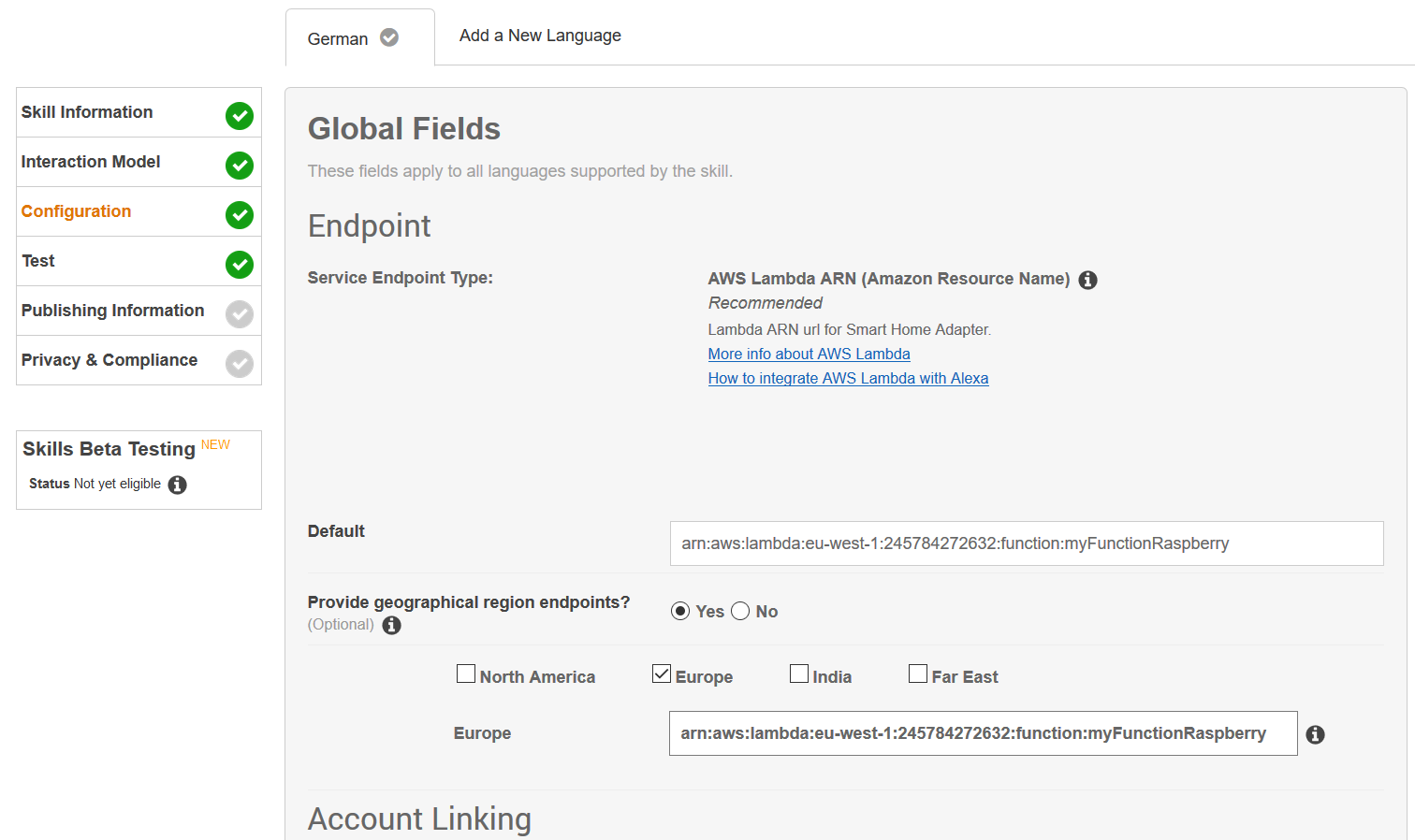
Zunächst einmal müssen im ersten Schritt unter „Skill Information“ grundlegende Informationen über den Skill festgelegt werden, welche die weiteren Schritte und deren Komplexität beeinflussen.

Wie schon in [Konzept -> Skilltypen] beschrieben haben wir uns hier für einen Skill entschieden, 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. 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.



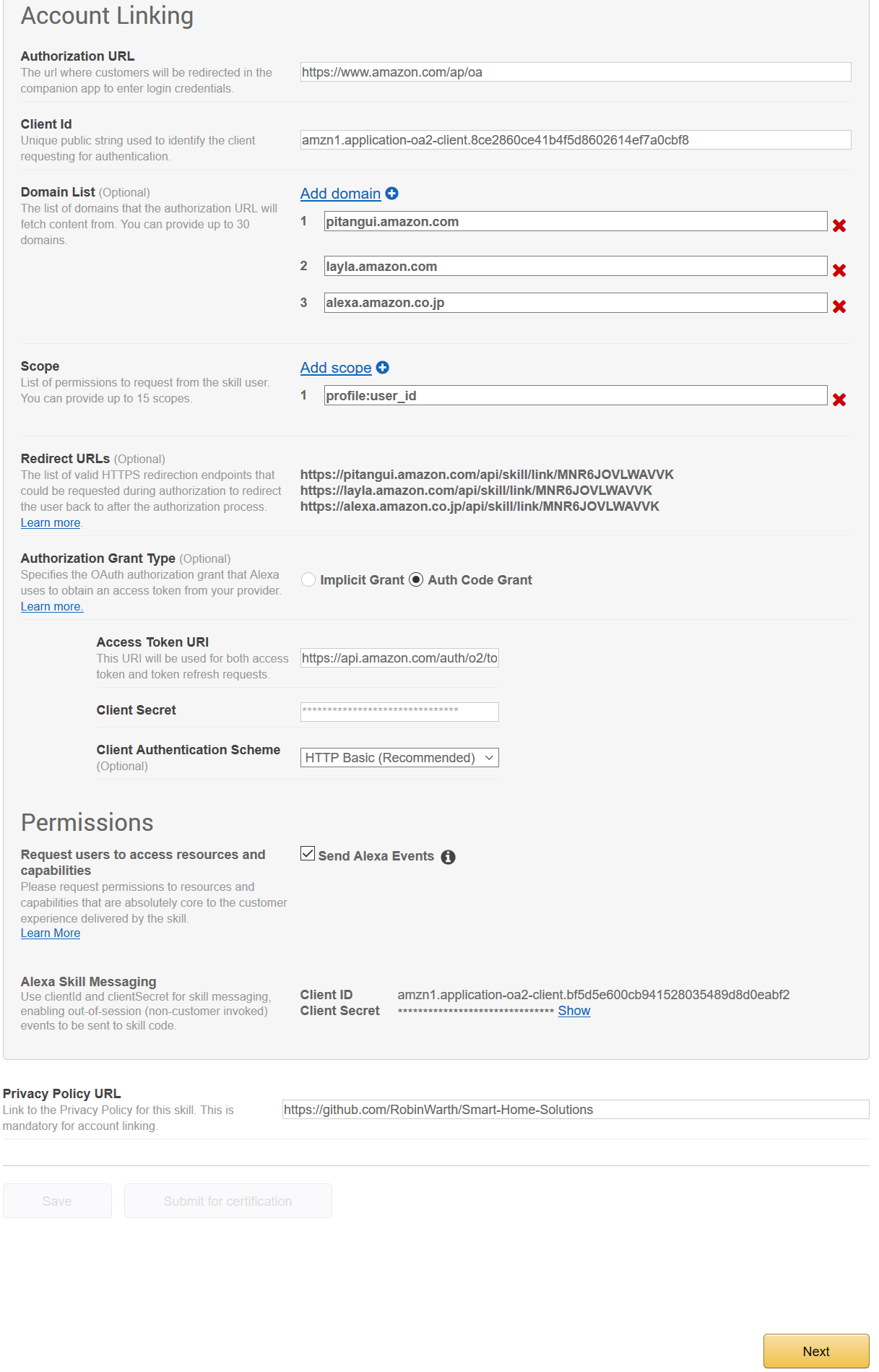
Da in diesem Skill die „Smart Home Skil API“ zum Einsatz kommt, muss in dem Schritt „Interaction Model“ nichts weiter getan werden. Das Interaktionsmodell ist bereits in die API eingebaut.



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.

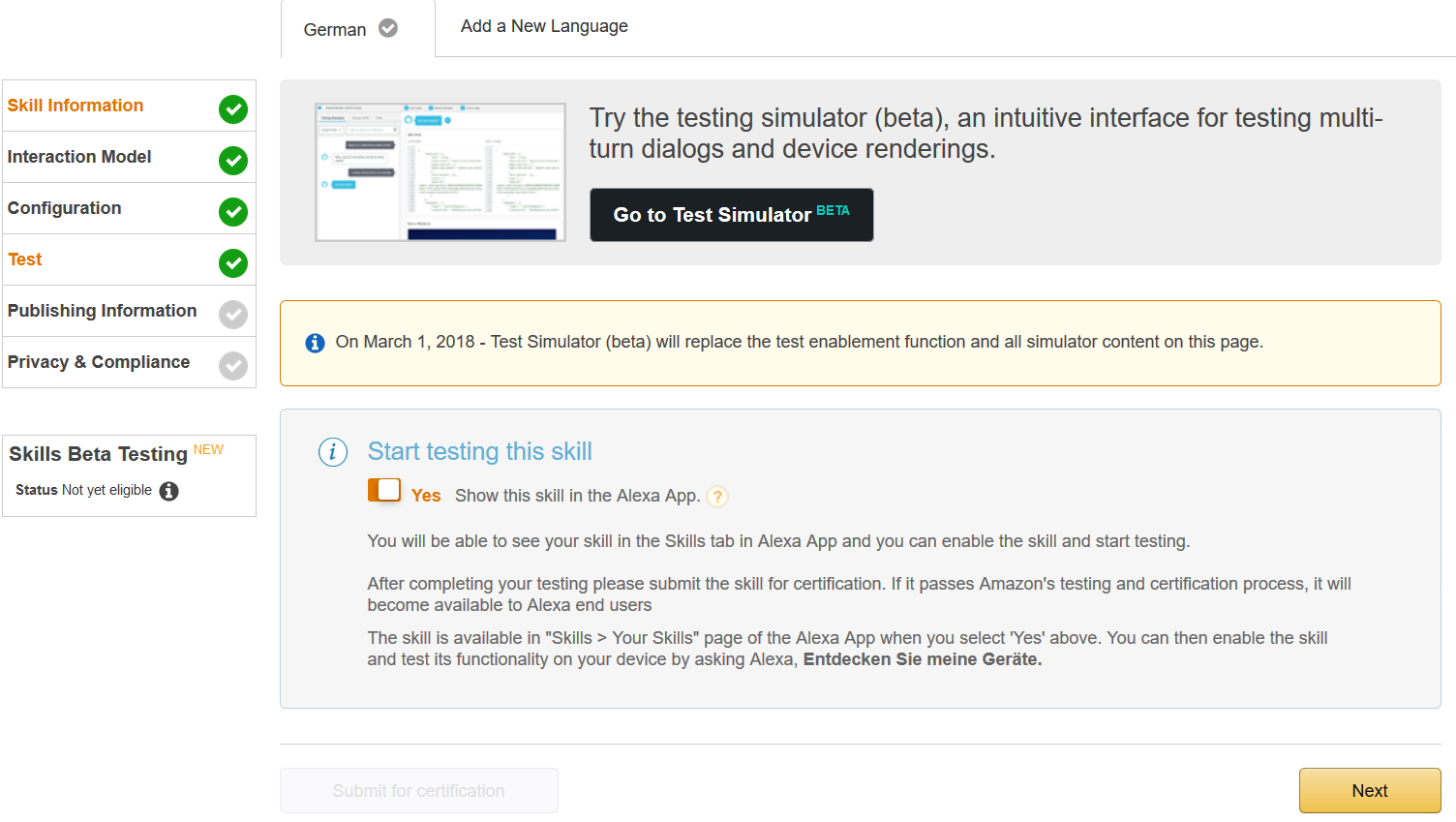
Bei Amazon Webservices wie dem Lambda Server wird jedem Server eine eindeutige Adresse zur Ressource und der darin enthaltenen Funktionen zugeteil. Diese Adresse nennt sich „Amazon Resource Name“ (kurz: ARN). Die ARN unseres Webservice lautet: „arn:aws:lambda:eu-west-1:245784272632:function:myFunctionRaspberry“. 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. 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.

…



Um den Skill mit der Alexa App auf dem Smartphone und einem Amazon Echo Gerät zu testen muss man nur den Schalter in Schritt „Test“ umlegen. Der Skill wird nun für die mit dem Konto verbundenen Geräte automatisch zum Testen eingerichtet. Alternativ kann man den Skill auch mittels, dem zur Zeit der Studienarbeit in der Beta-Phase befindlichen, Test Simulator ausführen. Der Test Simulator bietet jede Möglichkeit zum Testen wie mit einem Echo Gerät und sogar zusätzliche Funktionen. Im Test Simulator kann man direkt mit Alexa über Sprache oder geschriebenem Text interagieren. Hat man 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.