

# POLITECNICO DI TORINO

MASTER's Degree in INGEGNERIA  
INFORMATICA (COMPUTER ENGINEERING)



MASTER's Degree Thesis

Single-click recording and playback via  
Alexa for oral narratives

Supervisors

Prof. GIANPIERO CABODI

Prof. LUCIANO LAVAGNO

Candidate

LUDOVICO MULATERO

APRIL 2023

## Abstract

The project of this thesis is the research and development of an Alexa skill, a voice assistant, which queries the [tiraccontounastoria.org](http://tiraccontounastoria.org) server and then handles the different requests in the most appropriate way. The server contains stories told and experienced in first person, where anyone can listen to public stories and can tell new ones if they register.

The aim is to be able to carry out an initial search via Alexa for the stories on the website and finally listen, always via Alexa, to the audio file saved on the server. When Alexa receives a request from a user, the voice assistant processes the task and sends a specific request to the server, based on what the user wants, and then receives an answer to provide to the user who asked a specific question.

Server side, various requests are made to different URLs, depending on the type of requirement desired. Each kind of request will query the database on the basis of the information obtained and create a response. The server extracts, analyzes, filters and reorders the correct data based on the intent request from the skill and serializes these data to be sent to the back-end on Alexa that create a dedicated answer for the user.

Alexa side, one has to define the various intents: the set of example sentences that a user usually says. Some of these intents are accompanied by slots: extra information that is provided for a specific fact. After that, Alexa route the voice command to the most appropriate intent and the developer handle each intent so that Alexa give the appropriate response.

When starting the skill, the user can request a specific topic, check which public stories are present and which stories are told by a narrator. When Alexa plays the requested list, that list is divided into groups so as not to create a list that is too long. Cases of homonymy are handled, such as stories told by one person, and the stories reproduced are not repeated until all homonyms are listed. It is possible to listen to the audio by directly asking for the track title or number in the list. Once playing, it is possible to use the playback and positioning commands for a preset time by the developer.



# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Ti Racconto Una Storia . . . . .	1
1.2	Amazon and Alexa . . . . .	2
1.2.1	What is a skill? . . . . .	3
1.2.2	Automatic Speech Recognition and Natural Language Understanding . . . . .	4
1.2.3	Amazon Developer and AWS . . . . .	4
<b>2</b>	<b>The idea</b>	<b>6</b>
2.1	Record and playback audio using Alexa . . . . .	6
2.1.1	How to make it possible . . . . .	6
2.2	How they can communicate - ngrok . . . . .	7
<b>3</b>	<b>Implementation server-side</b>	<b>8</b>
3.1	HTTP request & JSON response . . . . .	10
3.2	Transfer data using ngrok . . . . .	11
<b>4</b>	<b>Implementation Alexa-side</b>	<b>13</b>
4.1	New dependencies required . . . . .	15
<b>5</b>	<b>Limitations, problems and (some) solutions</b>	<b>17</b>
5.1	Problem: Response delay . . . . .	17
5.2	Problem: Speech-To-Text (and some solutions) . . . . .	17
5.3	Problem: Audio file . . . . .	18
5.4	Limitation: Playback controller . . . . .	18
5.5	Limitation: Reload mp3 file . . . . .	19
<b>6</b>	<b>How to create a skill for Alexa using JavaScript</b>	<b>20</b>
6.1	First step . . . . .	20
6.2	Build . . . . .	22
6.2.1	Invocations . . . . .	22

6.2.2	Interaction Model - Intents . . . . .	22
6.2.3	Interaction Model - JSON Editor . . . . .	24
6.2.4	Interfaces . . . . .	31
6.2.5	Evaluate Model . . . . .	31
6.2.6	Tools . . . . .	31
6.3	Code . . . . .	31
6.3.1	Some hints . . . . .	32
6.3.2	CloudWatch Logs . . . . .	33
6.3.3	S3 Storage . . . . .	33
6.4	Test . . . . .	33
6.4.1	Development - Alexa Simulator . . . . .	33
6.4.2	Development - Manual JSON . . . . .	34
<b>7</b>	<b>Export-Import an existing project</b>	<b>35</b>
7.1	How to export an existing project . . . . .	35
7.2	How to import an existing code . . . . .	35
<b>8</b>	<b>Conclusion</b>	<b>37</b>
<b>A</b>	<b>Acknowledgments</b>	<b>39</b>



# Chapter 1

## Introduction

### 1.1 Ti Racconto Una Storia



A website named `tiraccontounastoria.org` is an idea where everyone can save a story using their voice and permits to make public their stories, so that anyone can hear them in the future.

In the website, some stories are already uploaded on a server and ready to be played. Here, it's possible to find true stories by true people, because it's the story teller that choose the arguments, what to say and for how many time speak about.

The front-end and the back-end are built with Django, a framework for web development in Python, and exploits the Django Rest Framework library to create RESTful APIs.

At the first time, a new user need to be signed in and only after that the user can record and save his own stories.

A story is recorded by the user as short or long voice messages and sent to the server of the website using the Telegram app. When the audio arrives, the same server extract and transcribe the keywords to have also the possibility to user to read the audio. This function require more or less time based on how long is the recorded story. Once it's ended, there is also the possibility to modify the text for a better matchmaking. This action is possible for the creator directly in his private area on the website.

When a new voice message is created, the server set it automatically as private, then it's possible to switch to public so everyone can listen the recorded story.

For example, people with the same story can provide different point of view of the same thing. It can be also useful to permits to grandparents to be remembered and for the new generations to listen to the voice of their parents.

## 1.2 Amazon and Alexa



**Alexa** is a virtual assistant technology largely based on a Polish speech synthesizer named Ivona, bought by Amazon in 2013. In 2014 the first device was released by Amazon and it's now called **Amazon Echo** or, simply, **Echo**. Not only Echos, but more device's manufactures can connect to Alexa. It is capable of voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, playing audio-books, and providing weather, traffic, sports, and other real-time information, such as news. Alexa can also control several smart devices using itself as a home automation system.



Alexa works using a cloud service divided in **Alexa Voice Service** and **Alexa Skill Kit**. The Alexa Voice Service needs only a WiFi connection and a microphone, it means that any device can be Alexa-enabled. The other side of Alexa is the Alexa Skill Kit. Once that Alexa is connected, the content that a user has access to is called **skill**.

### 1.2.1 What is a skill?

Skills are functionalities that the developer gives to Alexa to be able to do different things. When a user say a wake word, the device recognizes that is invoked and sends all the sentences to the **Alexa Service**. Using the **Automatic Speech Recognition**, it converts the audio to text, then Alexa sends the request to the back-end using **Natural Language Understanding** to know the meaning of the sentence, so the developer can create a dedicated answer to be returned for the user.

There are two sides to building an Alexa skill: **Voice Interaction Model** and **Programming Logic**. Originally, the programming logic was build on a external hosted-service or every other back-end, but now both are available on `developer.amazon.com`.

#### Voice Interaction Model

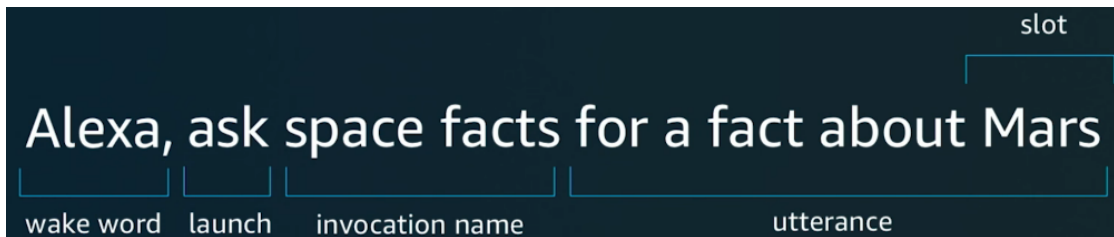
Basically, this is the main part of what is related with the voice. The skill and the back-end of the skill are JSON request and JSON response.

To start the interaction and open a skill, at first, it's required to activate the device using a **wake word** (typically **Alexa**). After that, it's possible to open the skill itself using words like **open**, **launch**, **start**, followed by the **invocation name**. The invocation name is the name that the developer decides and which the user will need to say to open the skill. This name, generally, is made by 2-3 words and can contain generic words. Furthermore, it's possible to open the skill and access directly a specific functionality (a command) using **utterance**.



The various utterances map to intents, because there are various way to say the same thing with the same meaning. For each intent created, a developer can create some example of common sentences and it's up to Alexa to route the user to the

right intent. But it's not enough, in a utterance a user can provide extra information for specific facts. That is called **slot**. Slot is the Alexa's voice equivalent of a variable used by developer to fulfill that specific intent. It's possible to divide slots in **built-in** and **custom**. Some built-in intents are mandatory to help the developer and are not required to set the most frequently used sentences and make the skill more user-friendly.



### Programming Logic

Here it's defined how to manage each intent that elaborates the request and provides a specific answer, in other words this is the back-end. A skill is an implementation of code mainly based in JavaScript or Python, it's possible to choose one of them at the creation, but, if a developer doesn't want to use a supported language, it's possible to use other languages not supported directly by Alexa and handle the JSON. Most skill run on cloud on the **Amazon's AWS Lambda** that is a Computing Service that runs code in response to events and automatically manages the computing resources required by that code.

### 1.2.2 Automatic Speech Recognition and Natural Language Understanding

Once Alexa is activated, the **Automatic Speech Recognition** will take care of turning the sound it receives into text. We speak of sound as voice is also a sound for Alexa. Once the text is obtained, the **Natural Language Understanding** will take care of understanding and routing the text to the correct intent. This whole process uses **AI** to transform sound into phonemes, then into words, then into sentences and finally into intents. The process will return a JSON with the text to be spoken by Alexa itself, this part is called **Text-To-Speech (TTS)**.

### 1.2.3 Amazon Developer and AWS

Previously, an external hosting service was required for the back-end part (such as **AWS** or other back-end services or servers available), whereas now Alexa-Hosted skills, a back-end hosted by Alexa, simplified and based on AWS, is made available.

Now, **Amazon Developer** is the only site where you can find the Voice Interaction Model and the Programming Logic in `amazon.developer.com` and everything will be done here. So, there is no separate back-end to develop the code.

Among the products offered by AWS, one of them is **S3** (Amazon Simple Storage Service), which allows a data/object storage service. Among the data that is stored within S3, we find, for example, the URL containing the audio that was previously stopped.

# Chapter 2

## The idea

### 2.1 Record and playback audio using Alexa



The idea is to bring together the stories recorded in a website and use an automated system that to get back the correct response, to see if they could become something more.

The aim of this project is to create a skill in Alexa that permits to the people to search in some way a story and playback the registered **mp3** file already saved on the server.

#### 2.1.1 How to make it possible

At this moment, related on what it is possible to build using Alexa, the skill can do different types of search and, also, receives the audio to be played. This is possible

adding code in the back-end part on the server that extracts and manipulates the correct data based on the intent request from the skill, then the server serializes and sends these to the back-end on Alexa to create a dedicated answer for the user.

When the skill is started, it's possible to ask for a specific argument, check what are the all public stories present and who are the story teller of the different stories presents.

Below it's shown how the code works in the back-end of server-side and of the Alexa-side.

## 2.2 How they can communicate - ngrok

To enable communication between the server and Alexa, the server must be online. However, since this runs locally and in the absence of a domain, a tunnel is what would allow the connection.

**ngrok** is an installable tool that does just that.

As said at <https://ngrok.com/docs/secure-tunnels/>:

**ngrok Secure Tunnels** work by using a locally installed ngrok agent to establish a connection to the ngrok service. Once the connection is established, you get a public endpoint that you or others can use to access your local service.

Ngrok is a necessary tool in the test phase for verifying the correct passage of data. By creating a tunnel, only those who have access to the tunnel are able to know what serialized data are being transferred. When the skill is ready to be made available to everyone, this tool will no longer be required.

This tool uses port 443 for communication between nodes on the network and using that port is precisely a requirement of Alexa to allow the transfer and then the listening of mp3 audio. Once installed on the device, using the command **ngrok http 8000** from the terminal, a direct communication between outside and inside can be established on port 8000.

## Chapter 3

# Implementation server-side

In order to be able to send data to Alexa, these data must be serialized in a JSON object, because this is the format in which Alexa wants them. The main steps before sending the data are:

- extracting the data from the database according to the request made on the Alexa side
- checking and filtering the data so that it is congruent with the request made
- creating the minimal object by adding only the attributes Alexa needs to be able to generate a response in the best way.

The server-based back-end is based on the Django REST framework. Developed in Python, the main files in the project that will be worked on are:

- **serializers.py** located in `../storyteller-master/st/serializers.py`;
- **views.py** located in `../storyteller-master/st/views.py`;
- **urls.py** located in `../storyteller-master/st/urls.py`.

The only `import` required in the **serializers.py** file is from `rest_framework` `import serializers`. This import permits to use a shortcut, use **ModelSerializer** to create a serialized class that creates an automatic set of fields and a simple default implementation of `create()` and `update()` methods.

In this way, using the various models already written, it's possible to return the serialized response in the **views.py** file calling this function.

As shown in the code below, `model` contains the model to serialize and `fields` contains all the data about the columns of the table present in the database that the developer wants to use.

```
1 class AnswerStorySerializer ( serializers.ModelSerializer ) :
2     class Meta:
3         model = Answer
4         fields = [ 'id', 'text', 'answerStatus', 'answerType',
                    'audio' ]
```

---

In the **views.py** file, there is the main part of the developed code. These functions send the JSON response to Alexa as a object that contain all the needed information. Here, the serialized data has been manipulated to create a response with only the essential data and sorted in a way that permits, sometimes, the best possible matchmaking. In this way, the data are accessible in `serializerTitle.data`.

```
1 def allTitleStory ( request ) :
2     storyTitle = Story.objects.all()
3     serializerTitle = StoryTitleSerializer ( storyTitle, many =
        True )
```

---

If an identification (ID) is passed as a parameter in the function for a specific request, it is possible to get the single object that contains only the required information. This ID is part of the URL that send the request.

```
1 def storyOfAPerson ( request, story ) :
2     song = Answer.objects.get( id = story )
```

---

In the **urls.py** file, there is the list of the all accessible links that a user can use to access a specific page of the website or see the JSON response for a specific request. The developer sets the different URLs.

```
1 url ( r'^person/', views.personList ),
```

---

Sometimes a parameter is added at the end to provide a specific set of data: `(?P<parameter>[a-zA-Z_0-9=]+)` where in the square parenthesis it is defined what kind of characters are permitted or not. It's required to set the import of the **views.py** file.

```
1 url ( r'^searchStoryAbout/?P<words>[A-Za-z_0-9=-]+' , views .
      searchStoryAbout ) ,
```

---

### 3.1 HTTP request & JSON response

At server-side, only few functions are created and used. To make a search on what is present in the database, it's possible to have a JSON object that contains all the persons that have told a story, which stories a single person has told, which stories are present in the database (only the public stories) and a general search about some keywords. There is another function that returns the mp3 file that contains the story chosen by the user to be listened.

Serialized data also contains fields that do not need to be sent, because those fields serve only as a control and filter. Consequently, it is possible to reduce the serialized data to be sent that contain only useful and necessary information.

To know the contents of a specific variable, it's required to parse the JSON data using `JSONRenderer().render(field["column"]).decode("utf-8")`, where the name `field` is the iterator's value and `column` is one of the fields added in the serializer.

The different functions are:

- **allTitleStory**: in this case, it's not enough to return back the serialized data because the stories must be filtered to know if these are public or private.
- **personList**: the idea is similar to the title story, moreover the list of persons is filtered on who already have a story saved. The object is also sorted by the number of stories for each author, to have a list that starts on who is more famous.
- **storyTitleOfAPerson**: to ask for a specific story teller, using the ID, the function returns back only his public stories.
- **searchStoryAbout**: here the function returns all the story tellers that match exactly with the first name or the last name with the words that are passed. Moreover, it makes a deep research on every title and inside each story to match with the required words. All the words are passed as a single parameter at the end of the URL. The words are concatenated by an underscore to be passed as a single parameter to the function.

First of all, split the parameter, there is a research in the text of the story to match every single word and count how many times these word are contained. For each story, divided for ID, there is the final sum of how many times



every word is found in the text and these ID are sorted by this final sum in descending order. To make more real the final result, only words longer than 3 characters are considered, otherwise another possible solution should be a self-created stop words list.

After that, it makes a research to obtain an exact match with the first name or last name to save the story teller. Subsequently, the algorithm checks and counts how many times the parameters are contained in the title and then the different title are sorted by sum in descending order.

There will then be an object containing the titles sorted in descending order, with the titles will be added based on the word search within the various stories sorted previously.

Only a single object can be returned, so there is an array that contains all the person and title found.

- **storyOfAPerson**: using the ID of the story that a user want to listen to, this function return directly the mp3 file to be played.

## 3.2 Transfer data using ngrok

Briefly, **ngrok** is a useful tool for creating a tunnel in the network for the secure transfer of information. It will create a link to be able to access a local server remotely.

### How to install ngrok on Linux

The installation of this tool is rather quick. Once the account has been created and logged into <https://dashboard.ngrok.com/login>, simply open the terminal on a Linux server and enter the command:

```
curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc | \
sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null && \
echo "deb https://ngrok-agent.s3.amazonaws.com buster main" | \
sudo tee /etc/apt/sources.list.d/ngrok.list && \
sudo apt update && sudo apt install ngrok
```

and press enter. Finally, you only need to link the previously created account with the command `ngrok config add-authtoken TOKEN`, replacing **TOKEN** with what is generated on

<https://dashboard.ngrok.com/get-started/your-authtoken>.

### How to use ngrok

Once you have done this, you can start ngrok with the command `ngrok http 8000` and use the URL provided. That link generated must be added to the list of links

allowing access to the server back-end in the variable `ST_ALLOWED_HOST` in the file `/storyteller-master/.env`. In the back-end of Alexa, the same URL must be passed as the first part of the parameter to the axios function concatenated with the specific path to execute the HTTP request.

### **Enable communications**

Once the tool is installed, a further package must be installed from the command line and then add some row in the file `/StoryTeller/settings.py`'.

The first step is to install **CORS** from the root directory. CORS (Cross-origin resource sharing) is a mechanism to request the reserved resources of a web page from another domain than the one from which the first resource was served.

The command line to use is: `python -m pip install django-cors-headers`.

Within the `settings.py` file, in `INSTALLED_APPS`, this must be indicated as even though our new app exists within the Django project, Django does not “know” about it until we explicitly add it.

A new line containing `corsheaders.middleware.CorsMiddleware` must also be added in `MIDDLEWARE`.

Finally, at the end of the file, add `CORS_ALLOW_ALL_ORIGINS=True`.

## Chapter 4

# Implementation Alexa-side

The back-end used to develop the skill is based on **Node.js**, but let's proceed by steps.

The creation of intent takes place in the **Build** part. Each intent has a specific function. There are intents that have no slots to worry about, they only receive basic commands (`searchStoryTellerIntent`, `searchTitleStoryIntent`, `OtherPersonOrStoryIntent`), other intents have to work with slots. Each slot must be identified by a type, the ones used in this project are:

- **AMAZON.Person** is a slot dedicated to receive a first and last name (`SaveStoryTellerIntent`);
- **AMAZON.Number** is dedicated to receive any number (`TellStoryIntent`). By opening the slot in question, in the **Validations** page it is possible to provide a range for which the said number may be valid, so as to avoid possible errors. In this case it is useful to consider only positive numbers;
- **AMAZON.SearchQuery** retrieves all that is said, it is not possible to have another slot in the same utterance (`SaveStoryTellerIntent`, `TellStoryIntent`, `SearchStoryAboutIntent`, `AllPersonOrStoryIntent`)

At the creation (see *How to create a skill*, Chapter 4), a pre-filled workspace is ready in the **Code** page. The first intent is dedicated to the first action the skill must take when it is invoked. It is usually a welcome or welcome back message, followed by what the skill itself can do.

In this skill, the new developed intents are:

- **SearchStoryAboutIntent**: Once the intent has received the slot containing what is to be searched for, the words in question are all concatenated with underscores. This new parameter is passed to a function that makes an HTTP request and receives a JSON object containing all the information for which

the search was made. An initial check is made to ensure that something is present in it, otherwise Alexa will say that the search produced no results. If the object is not empty, an initial list of people and stories present will be provided;

- **AllPersonOrStoryIntent:** With this intent, the user can define how many names or titles Alexa should list, otherwise it's possible to ask to hear them all;
- **OtherPersonOrStoryIntent:** A separate intent will allow the user to continue listening to the previously started list or have Alexa tell you that no more items are available;
- **SearchStoryTellerIntent:** Another possibility is to allow searching by story teller, i.e., to know who has reported something. An HTTP request will be sent and a JSON containing the first and last names will be received. Alexa will list them in groups, with the possibility of hearing the next ones exploiting the intent described above in **OtherPersonOrStoryIntent**;
- **SearchTitleStoryIntent:** As for the story tellers, the execution is the same for the stories;
- **SaveStoryTellerIntent:** It allows for the user to search specifically for a particular person and find out what stories they have told. An initial check immediately verifies whether the person requested is among those existing in the database. If present, their titles will be listed. The case of homonymy is also handled: an author is randomly chosen and excluded for a subsequent query for the same name. The operation will continue until the last homonym or another person is requested;
- **TellStoryIntent:** There are two ways of deciding which story to listen to: by saying the number in the list or by specifying the title. A first check is made on the basis of the number said, if this is zero or greater than the number of stories present, Alexa will generate a random number and execute it. Otherwise, based on how the user says the utterance, anything that is not representable as a number the speech recognition will interpret it as text, thus without symbols, or replies that the entered number is not valid. If the user says the title, several attempts will be made to find the best possible association with the request. The use of stop-words on both (the request and the available stories) added with the comparison on the word count manages to give the best possible matchmaking. In the case of homonymy on titles and imperfect comparisons, titles will be chosen randomly one at a time until they are exhausted. Instead, if the number is said, Alexa will automatically play

the story in that position.

For greater certainty that Alexa chooses the audio correctly, it will repeat the title and author;

- **StartOverIntentHandler**: It allows the skill to start playing audio again and will only be valid if there is already an mp3 playing;
- **NextIntentHandler**: It skip ahead a set time in listening;
- **PreviousIntentHandler**: On the contrary, it allows one to go back a set time in listening;
- **PauseAudioIntentHandler**: It pauses the audio;
- **PlayAudioIntentHandler**: It allows the audio to be resumed from where it had stopped, whether it was only momentarily stopped or if resumed at a later time. If the audio is already playing, the request for this intent is ignored.

Finally, there remain only the last intentions dedicated to errors, i.e., intentions that should not be invoked but will be in the event of a malfunction somewhere in the code.

## 4.1 New dependencies required

- **ask-sdk-s3-persistence-adapter**: S3 is an AWS service that stores more information and objects. Alexa-hosted skills uses S3 to store our key value pairs using the persistent adapter of S3. S3 is an available database ready for the users by Alexa-hosted skills;
- **ask-sdk-dynamodb-persistence-adapter**: At the creation of a new skill, the developer gets access to an Amazon DynamoDB table for persisting data.
- **i18next**: Used to have multiple languages answer at the same time. It's also possible to have more languages (e.g. English-UK and English US) that redirect to the same response, otherwise it is possible to add a new object for a specific language to overwrite the same sentence.
- **axios**: A Node library for the HTTP request. It returns JSON data. This library also has a timeout, which can be useful because the skill has 8 seconds to respond, so a response (which can be empty) is set anyway.

```
1     var config = {
2         timeout : 6500,
3         headers: { 'Accept': 'application/json' }
4     }
5     return axios.get( url,config )
6         .then( result => result.data )
7         .catch(( error )=>{
8             console.log( error );
9         });
10
```

---

Alexa-Hosted is based on JavaScript that use Node.js so, for each new dependency added in the `package.json`, an `npm install` is done automatically when the developer click on the **Deploy** button and this will take longer when running the program, precisely because it will have to perform the installation.

# Chapter 5

## Limitations, problems and (some) solutions

### 5.1 Problem: Response delay

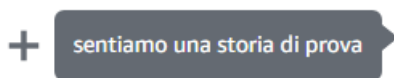
**Problem:**

As introduced in the axios explanation, Alexa has **8 seconds** to provide a response. Sometimes the answer comes immediately, other times it will take a little longer to get it.

**Solution:**

The **progressive response** allows itself to travel in parallel with the real response and only if the response does not come immediately does it intervene to avoid leaving silences and not knowing whether the command has been acquired or not.

### 5.2 Problem: How Alexa convert the sentences



**Figure 5.1:** What the user can say

```
2023-02-14T17:24:29.289+01:00      2023-02-14T16:24:29.289Z 15b2047b-215b-4554-aa2c-fa266a067fb1 INFO 1 storia di prova
2023-02-14T16:24:29.289Z      15b2047b-215b-4554-aa2c-fa266a067fb1  INFO  1 storia di prova
```

**Figure 5.2:** What Alexa returns

**Problem:**

When a slot is filled, Alexa will fill that variable using the slot type it was declared with. With the slot AMAZON.SearchQuery, everything is translated into text form, except for numbers which are transcribed into numeric form, i.e. digits. In particular, in the Italian language everything that resembles the word 'uno' is proposed as '1', i.e. un, uno, una, un' will be recognised as numerical digit '1'. The problem arises when one wants to listen to a certain title containing one of these words but the algorithm requires a 1-1 correspondence in order to play the story.

**Possible solutions:**

- The use of stop-words would exclude less relevant words within the string, but this would also lead to a greater number of possible associations with other titles. It could be reduced by counting all words in the passed string and in the comparative string.
- (version valid for Italian) The use of an algorithm capable of replacing the number '1' with all its article variants and performing the comparison. Fast but with low probability of success.
- (valid version for Italian) The use of simple permutations with the intention of trying all the different possibilities. Much more effective, but unsuitable for large amounts of data.

### 5.3 Problem: Audio file

For the playback of an audio file, Alexa imposes constraints. These include an HTTPS end-point accessible on port 443, the server must have a valid and trusted SSL certificate. The file format must be AAC/MP4, MP3, PLS, M3U/M3U8 or HLS, and with bit-rates between 16kbps and 384kbps.

**Problem:**

The files saved on the server are in a WAV format because this was required by the synthesizer for transcription into text.

**Solution:**

Save a second copy of the audio file in a format required by Alexa.

### 5.4 Limitation: Playback controller

**Problem:**

Once the mp3 file is loaded and executed, Alexa will only allow more commands



related to audio and the possibility of exiting the skill. This means that Alexa will not allow a second search while a file is playing.

The only possible solution is to stop the skill, re-execute it and finally be able to perform a second search.

More specific commands, such as going forwards or backwards in song playback, are only possible by a set amount in the code. It is not the user who defines this precisely because it is no longer part of the directives.

As specified in <https://developer.amazon.com/en-US/docs/alexa/custom-skills/playback-controller-interface-reference.html>:

**Note:** When responding to any PlaybackController request, you can only respond with AudioPlayer directives. The response cannot include any of the standard properties such as outputSpeech, card, or reprompt. Sending a response with these unsupported properties causes an error

## 5.5 Limitation: Reload mp3 file

A limitation of listening to audio is that it is reloaded every time you move through the track. A new request will be made to the server to resume from the requested point. Having to reload the audio file each time a request is made to move to the track, requires that there be a stable and reliable connection, otherwise the time spent waiting for the jump could be longer than listening to the audio in the same period of time. Linked to this, the impossibility of not being able to let the user decide how far to move forward or backward in the track is underperforming. For stories that are too short, a jump of a few minutes is a very big time. Conversely, for such long stories, a jump of about a few seconds is little.

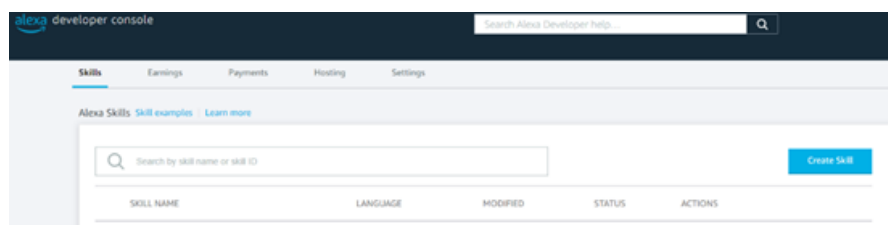
## Chapter 6

# How to create a skill for Alexa using JavaScript

### 6.1 First step

By logging in at <https://developer.amazon.com/alexa/console/ask>, you can create a new account or use an existing Amazon account. Initially the page will be empty, as new skills are created they will be added to this page. There is a limit of 75 skills set by Amazon that can be added, to add new skills you must delete previous ones.

At first, click on the blue **create a skill** button.



Decide on a name for the skill and the main language you want to use. Click on **next** in the top right hand corner.

The screenshot shows the first step of the Alexa skill creation process. At the top, there are four numbered steps: 1. Name, Locale (highlighted in blue), 2. Experience, Model, Hosting service, 3. Templates, and 4. Review. Below the steps, the title is "1. Name your Skill". A sub-header reads: "Enter a name for your skill. This is not the same as the skill invocation name, which you'll set up after you complete the skill set up process." There is a text input field labeled "Skill name" containing the text "Demo skill". Below the field, it says "10/50 characters". A small note below the field states: "Brand names are only allowed if you provide proof of rights in the testing instructions or if you use the brand name in a referential manner that doesn't imply ownership (examples of terms that can be added to a brand name for referential usage: unofficial, unauthorized, fan, fandom, for, about)." Below this, the next step is "2. Choose a primary locale". A sub-header reads: "A locale refers to a language and the country in which it's spoken. Build your skill in your primary locale. You can add more locales after you create the skill." There is a dropdown menu showing "Italian" with a downward arrow.

For this skill, the choice will be **other**. Leave the **custom** model for greater flexibility. Choose the programming language from JavaScript and Python where Alexa-Hosted will create a back-end for the developer directly in the back-end for developer. Instead, **provision your own** the developer define his back-end. To reduce latency, the nearest host region should be chosen and click on **next** button.

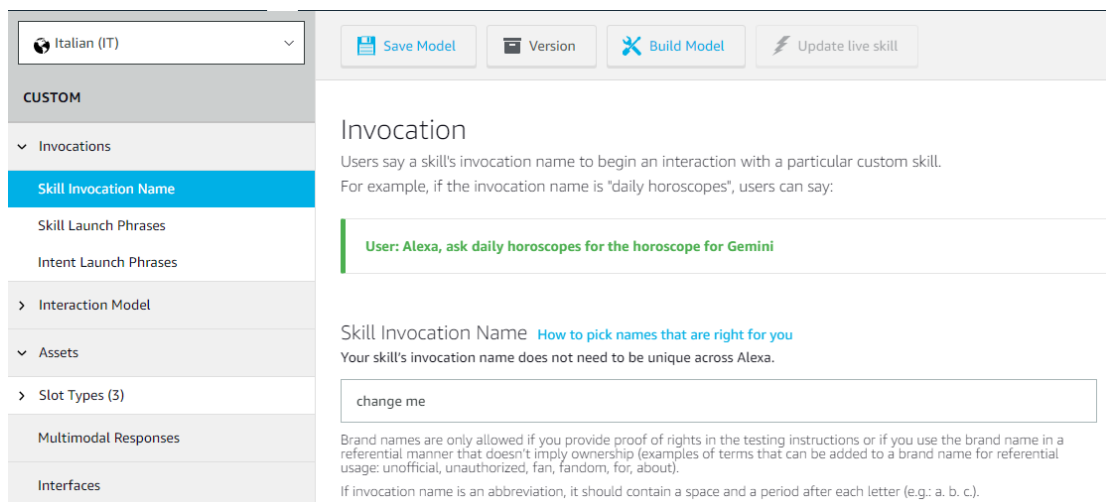
The screenshot shows the "Hosting region" section. The title is "Hosting region". Below the title, it says: "To reduce perceived latency, choose the hosting region closest to the majority of your users. [Learn more](#) [↗](#)". There is a dropdown menu showing "EU (Ireland)" with an upward arrow. Below the dropdown, there is a list of regions: "US East (N. Virginia)", "EU (Ireland)", and "US West (Oregon)". The "EU (Ireland)" option is highlighted in blue.

In the **templates** section, there is no need to edit, then click **next** again and finally **Create Skill**. It will take a couple of minutes to create.

## 6.2 Build

### 6.2.1 Invocations

At the beginning it provides the name of the skill, but it's possible to change to anything the developer want. The invocation name that will be chosen will be the one used to execute the skill. Finally, click on **Save Model**.



### 6.2.2 Interaction Model - Intents

Here we have the **intents**, that are the function that the skill has. Some are predefined and already set. It's possible open them and add other hints to make more comfortable what the function can do. Here the developer can add the intents that he want.

In the **Intent** section within **Interaction Model** it's possible to create the intents needed. By clicking on **+ Add Intent**, the developer can choose from one of the already existing intents or create a new one with a specific name and finally click on **create custom intent**. In the **sample utterances**, it's also possible to indicate which sample sentences will direct Alexa to that specific intent. For each phrase one or more slots can be added, it's useful if these are needed in the **code** part. The syntax for creating a slot is: **{slot}**. It will be automatically created and then the developer must indicate the type of slot from those present.

Intents

[+ Add Intent](#)

NAME	UTTERANCES	SLOTS	TYPE	ACTIONS
<a href="#">AMAZON.CancelIntent</a>	-	-	Required	<a href="#">Edit</a>
<a href="#">AMAZON.HelpIntent</a>	5	-	Required	<a href="#">Edit</a>
<a href="#">AMAZON.PauseIntent</a>	-	-	Required	
<a href="#">AMAZON.ResumeIntent</a>	-	-	Required	
<a href="#">AMAZON.StopIntent</a>	2	-	Required	<a href="#">Edit</a>
<a href="#">HelloWorldIntent</a>	6	-	Custom	<a href="#">Edit</a>   <a href="#">Delete</a>
<a href="#">AMAZON.NavigateHomeIntent</a>	-	-	Required	<a href="#">Edit</a>
<a href="#">SearchStoryTellerIntent</a>	5	-	Custom	<a href="#">Edit</a>   <a href="#">Delete</a>
<a href="#">SaveStoryTellerIntent</a>	5	1	Custom	<a href="#">Edit</a>   <a href="#">Delete</a>

Sample Utterances (12) ⓘ

What might a user say to invoke this intent?

voglio sentire [titolo]

sentiamo [titolo]

scelgo la numero [numero]

scelgo [titolo]

fammi ascoltare [titolo]

Dialog Delegation Strategy ⓘ [Learn more](#)

fallback to skill setting ▼

Intent Slots (2) ⓘ

ORDER	NAME	SLOT TYPE	ACTIONS
1	<span>numero</span>	AMAZON.NUMBER	<a href="#">Edit Dialog</a>   <a href="#">Delete</a>
2	<span>titolo</span>	AMAZON.SearchQuery	<a href="#">Edit Dialog</a>   <a href="#">Delete</a>

## Slot

**Dialogs:** By clicking on the slot, it is possible to define whether it is mandatory and to demand that it be filled.

**Validations:** Depending on the type of slot created, only certain values can be accepted. In the case of a slot of type number, it is possible to make the user only receive numbers greater than zero (useful when he wants to listen to a song in a certain position).

## Intent validation

Only when the entire Dialog Delegation is completed, Alexa resumes the intent with the required slot and asks for a confirm. Only after a confirm by the user, the intent has been activated and the JSON is created.

### 6.2.3 Interaction Model - JSON Editor

In the **JSON Editor** it's possible to find what is previously said, so the developer can easily modify only the JSON and everything will be automatically updated once the build is done.

The code below shows the creation of the intents for this project.

```
1 {
2   "interactionModel": {
3     "languageModel": {
4       "invocationName": "storia di prova",
5       "intents": [
6         {
7           "name": "AMAZON.CancelIntent",
8           "samples": []
9         },
10        {
11          "name": "AMAZON.HelpIntent",
12          "samples": [
13            "cosa puoi fare",
14            "aiutami",
15            "cosa posso fare",
16            "help",
17            "aiuto"
18          ]
19        },
20        {
21          "name": "AMAZON.PauseIntent",
22          "samples": []
23        },
24        {
25          "name": "AMAZON.ResumeIntent",
26          "samples": []
27        },
28        {
29          "name": "AMAZON.StopIntent",
30          "samples": [
31            "basta",
32            "stop"
```

```
33     ]
34   },
35   {
36     "name": "HelloWorldIntent",
37     "slots": [],
38     "samples": [
39       "chi ha sempre ragione",
40       "ciao",
41       "come stai",
42       "dì ciao",
43       "salutami",
44       "salutarmi"
45     ]
46   },
47   {
48     "name": "AMAZON.NavigateHomeIntent",
49     "samples": []
50   },
51   {
52     "name": "SearchStoryTellerIntent",
53     "slots": [],
54     "samples": [
55       "chi ha raccontato cosa",
56       "quali persone ci sono",
57       "quale persona ha raccontato qualcosa",
58       "chi c'è ",
59       "chi ha raccontato una storia"
60     ]
61   },
62   {
63     "name": "SaveStoryTellerIntent",
64     "slots": [
65       {
66         "name": "persona",
67         "type": "AMAZON.Person"
68       }
69     ],
70     "samples": [
71       "che cosa ha raccontato {persona}",
72       "voglio sapere i titoli delle storie di {persona}",
73       "voglio sapere le storie di {persona}",
74       "quali storie ha raccontato {persona}",
75       "scegli tu quella di {persona}"
76     ]
77   },
78   {
79     "name": "TellStoryIntent",
80     "slots": [
81       {
```

```
82         "name": "numero",
83         "type": "AMAZON.NUMBER"
84     },
85     {
86         "name": "titolo",
87         "type": "AMAZON.SearchQuery"
88     }
89 ],
90 "samples": [
91     "voglio sentire {titolo}",
92     "sentiamo {titolo}",
93     "scelgo la numero {numero}",
94     "scelgo {titolo}",
95     "fammi ascoltare {titolo}",
96     "quella dal titolo {titolo}",
97     "fammi ascoltare la numero {numero}",
98     "fammi ascoltare la {numero}",
99     "voglio sentire la storia che si intitola
100 {titolo}",
101     "vai con la {numero}",
102     "sentiamo la {numero}",
103     "racconta la numero {numero}"
104 ]
105 },
106 {
107     "name": "SearchTitleStoryIntent",
108     "slots": [],
109     "samples": [
110         "quali storie mi proponi",
111         "Proponimi qualche storia",
112         "Proponi qualche storia",
113         "Elencami le storie disponibili",
114         "Quali storie ci sono",
115         "I titoli delle persone",
116         "Quali sono i titoli disponibili",
117         "Quali titoli ci sono",
118         "Quali storie sono disponibili"
119     ]
120 },
121 {
122     "name": "SearchStoryAboutIntent",
123     "slots": [
124         {
125             "name": "argomento",
126             "type": "AMAZON.SearchQuery",
127             "samples": [
128                 "{argomento}",
129                 "su {argomento}"
130             ]
131         }
132     ]
133 }
```



```
130     }
131   ],
132   "samples": [
133     "raccontami una storia di {argomento}",
134     "cosa mi sai dire su {argomento}",
135     "trovami qualcosa su {argomento}",
136     "cercami qualcosa su {argomento}",
137     "proponimi qualcosa su {argomento}",
138     "raccontami qualcosa su {argomento}",
139     "cerca qualcosa su {argomento}",
140     "cosa proponi su {argomento}",
141     "voglio sentire qualcosa su {argomento}",
142     "qualcosa su {argomento}",
143     "dove si parla di {argomento}",
144     "fammi ascoltare qualcosa su {argomento}",
145     "quali storie ci sono sulla {argomento}"
146   ]
147 },
148 {
149   "name": "AllPersonOrStoryIntent",
150   "slots": [
151     {
152       "name": "argomento",
153       "type": "AMAZON.SearchQuery"
154     }
155   ],
156   "samples": [
157     "fammi sentire tutte le {argomento}",
158     "elenca tutti i {argomento}",
159     "sentiamo la prima {argomento}",
160     "elenca la prima {argomento}",
161     "elenca tutte le {argomento}",
162     "sentiamo le prime {argomento}",
163     "elenca le prime {argomento}",
164     "sentiamo tutte le {argomento}"
165   ]
166 },
167 {
168   "name": "OtherPersonOrStoryIntent",
169   "slots": [],
170   "samples": [
171     "mostra i prossimi risultati",
172     "mostrami i risultati successivi",
173     "mostrami altri risultati",
174     "Dimmi le successive",
175     "Quelle dopo",
176     "Dopo cosa c'è",
177     "Le prossime",
178     "Dimmene altre"
```

```
179         ]
180     },
181     {
182         "name": "AMAZON.StartOverIntent",
183         "samples": []
184     },
185     {
186         "name": "AMAZON.NextIntent",
187         "samples": []
188     },
189     {
190         "name": "AMAZON.PreviousIntent",
191         "samples": []
192     }
193 ],
194     "types": []
195 },
196 "dialog": {
197     "intents": [
198         {
199             "name": "AllPersonOrStoryIntent",
200             "confirmationRequired": false,
201             "prompts": {},
202             "slots": [
203                 {
204                     "name": "argomento",
205                     "type": "AMAZON.SearchQuery",
206                     "confirmationRequired": false,
207                     "elicitationRequired": false,
208                     "prompts": {}
209                 }
210             ]
211         },
212         {
213             "name": "SearchStoryAboutIntent",
214             "confirmationRequired": false,
215             "prompts": {},
216             "slots": [
217                 {
218                     "name": "argomentoo",
219                     "type": "AMAZON.SearchQuery",
220                     "confirmationRequired": false,
221                     "elicitationRequired": true,
222                     "prompts": {
223                         "elicitation":
224 "Elicit.Slot.1483798665032.217800179993"
225                     }
226                 }
227             ]
228         }
229     ]
230 }
```

```
227     },
228     {
229         "name": "TellStoryIntent",
230         "confirmationRequired": false,
231         "prompts": {},
232         "slots": [
233             {
234                 "name": "numero",
235                 "type": "AMAZON.NUMBER",
236                 "confirmationRequired": false,
237                 "elicitationRequired": false,
238                 "prompts": {},
239                 "validations": [
240                     {
241                         "type": "isGreaterThanOrEqualTo",
242                         "prompt":
243 "Slot.Validation.1395478475284.1573965218901.1548188764617",
244                         "value": "0"
245                     }
246                 ],
247             },
248             {
249                 "name": "titolo",
250                 "type": "AMAZON.SearchQuery",
251                 "confirmationRequired": false,
252                 "elicitationRequired": false,
253                 "prompts": {}
254             }
255         ]
256     },
257     "delegationStrategy": "ALWAYS"
258 },
259 "prompts": [
260     {
261         "id": "Elicit.Slot.250658637082.644736003049",
262         "variations": [
263             {
264                 "type": "PlainText",
265                 "value": "Devo sapere se tutte o il numero di
266 storie o persone da farti conoscere"
267             }
268         ]
269     },
270     {
271         "id": "Elicit.Slot.1483798665032.217800179993",
272         "variations": [
273             {
274                 "type": "PlainText",
```

```
274         "value": "Devo conoscere almeno l'argomento"
275     }
276 ]
277 },
278 {
279     "id": "Elicit.Slot.1049738387051.291288904675",
280     "variations": [
281         {
282             "type": "PlainText",
283             "value": "Ritenta"
284         }
285     ]
286 },
287 {
288     "id": "Elicit.Slot.932451633348.1161941290713",
289     "variations": [
290         {
291             "type": "PlainText",
292             "value": "devo conoscere almeno l'argomento"
293         }
294     ]
295 },
296 {
297     "id":
298     "Slot.Validation.1395478475284.1573965218901.1548188764617",
299     "variations": [
300         {
301             "type": "PlainText",
302             "value": "Inserisci un numero maggiore di zero"
303         }
304     ]
305 },
306 {
307     "id":
308     "Slot.Validation.1172069686837.1461406282166.589840667553",
309     "variations": [
310         {
311             "type": "PlainText",
312             "value": "Inserisci un valore corretto"
313         }
314     ]
315 }
316 }
```

---

## 6.2.4 Interfaces

This is a fundamental step in listening an audio. The developer must enable **Audio Player** to give Alexa the ability to play external audio.

## 6.2.5 Evaluate Model

After clicking on **Build Model**, an initial test can be carried out to check whether the main commands work. At the top right, **Evaluate Model** offers the possibility of interaction so as to know whether the text is understood and whether the slots are identified correctly. It allows interaction without the use of the back-end.

## 6.2.6 Tools

Permissions: there is the list of the permission to have. There is no save button to build that part, all is automatic updated. Developing the code, [`alexa::profile:given_name:read`] is the address from the request's Alexa API. To load the data, the function will use the **UPSclient** (User Profile Service) that is a client that aggregate all this type of request. If that permission is not enabled, the code will works anyway but without these information. Final user-side, in the Alexa app a popup will appear to ask to the user if he want to enable these permission or not.

## 6.3 Code

The Alexa object is created at the beginning using the **ask-sdk-core**. At the end of the program is Alexa object is instantiated and the handler are registered in `.addRequestHandler()` and all terminated using `.lambda()`. The button for compiling the code is in the top right-hand corner and is named **Deploy**.

```
1 const Alexa = require('ask-sdk-core');
2
3 const IntentHandler = {
4   canHandle(handlerInput) {
5     return Alexa.getRequestType(handlerInput.requestEnvelope) ===
6       'IntentRequest'
7       && Alexa.getIntentName(handlerInput.requestEnvelope) ===
8       'Intent';
9   },
10  handle(handlerInput) {
```

```
9     const speakOutput = "Hi";
10     return handlerInput.responseBuilder
11         .speak(speakOutput)
12         .reprompt('add a reprompt if you want to keep the session open
13     for the user to respond')
14         .getResponse();
15     }
16 };
17 exports.handler = Alexa.SkillBuilders.custom()
18     .addRequestHandlers(
19         IntentHandler)
20     .lambda();
```

---

How a handler work:

- First function: **canHandle()** know if a handler is the correct one to use the request. It returned only true or false.
- Second function: if the first is true, determines what needs to be done, once the request is coming in.

### 6.3.1 Some hints

- **Inteceptors**: they run with each request or response. The **RequestInteceptors** run just after the request arrives at Alexa and before the handlers handle the response, while the **ResponseInterceptors** run just before the response is sent to Alexa.
- **Session Attributes**: it's a short-term memory. Those attributes that are persistent only during the session, when the skill is stopped that attributes are deleted. The session attributes are called using **handlerInput.attributesManager.getSessionAttributes()** that returns an object that can be already filled or empty object, if not defined.
- **Persistent Attributes**: long-term memory. Attributes that can be used for more than one session are retrieved using an async function called **attributesManager.getPersistentAttributes()** and set in the sessionAttributes using **attributesManager.setPersistentAttributes**. To save the session attributes, instead, the first step is write them in the persistent attributes using **attributesManager.setPersistentAttributes(sessionAttributes)** and using an async function save them using **attributesManager.savePersistentAttributes()**. The lambda function add a new line **.withPersistentAdapter(persistentAdapter)** to say that it's used the persistent adapter.

- **Progressive response:** `CallDirectiveService` is a request to something external, it may require time. Starting from the user when asks something to the skill until the skill will response, may pass until 8 seconds because after that the skill stop to work. Waiting 8 seconds for an answer, so the Progressive Response permits us to insert a message to say while the skill elaborate the truth answer. The progressive response is queued and, only if the message in the queue arrives before the real answer, Alexa will start with the progressive response and then with the answer. Otherwise only the answer will said by Alexa and the progressive response will be ignored.

### 6.3.2 CloudWatch Logs

Sometimes it is useful to be able to know the value of variables, to understand how they are changed by the code and to fix new errors. **CloudWatch** is a tool that collects console.logs and keeps track of them, just like on a black screen. It also has a history of previous printouts. It is not very intuitive to use, but the history displayed is in chronological order and is constantly updated during testing, so the developer can keep track of how the data is changed. Each time a deployment is made, this creates a new log stream.

### 6.3.3 S3 Storage

Having spoken of **persistent attributes**, there is also a database in which this data is stored. There is no way to change or view the saved data, the only possible action is to delete the content. When the dedicated page is opened, it will be empty and devoid of content. In order to be able to delete the present data, it is necessary to step back into the directory by pressing on the name following **Amazon S3 > Bucket >** so that the object in question can be selected and deleted.

## 6.4 Test

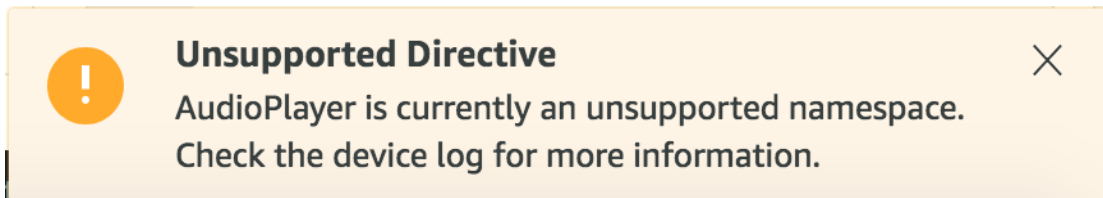
Once the deployment has been completed, the developer can test the operation in the **Test** section.

At the first access to the test page, the whole page will be greyed out. The developer will have to switch from **Off** to **Development** in the top left-hand corner and then you can start interacting.

### 6.4.1 Development - Alexa Simulator

With the provided simulator it is possible to have a first interaction with Alexa, being able to interact either via keyboard or vocally if equipped with a microphone.

The only limitation that this simulator has is that actual audio playback is not possible, so for this type of test it is necessary to opt for an Echo dot or by downloading the application on the smartphone. Since the developer owns the skill, it is automatically installed and ready to use on the device. Thanks to the simulator, it is possible to see the JSON object being sent and received by Alexa.



## 6.4.2 Development - Manual JSON

The same JSON object sent can be copied and modified within this section. In this way, it is possible to modify part of the request made and see how Alexa's response behaves.

The object that is created will only be sent when the required slots are correctly filled, if present in a given intent. There will therefore be no empty or partial request.



## Chapter 7

# Export-Import an existing project

### 7.1 How to export an existing project

To export a skill, after a first login phase and selected the skill to export, simply go to the Code section, click on **Download Skill** then **Continue**. A zip file will be downloaded containing a folder with a JSON file containing all the various intents, a folder with the back-end and a `skill.json` file.

### 7.2 How to import an existing project

To import a skill the developer needs to choose which skill to modify (or create a new one as explained above), in the **Code** section click on **Import Code** and add the desired zip file. A window will open and the affected files can be added. Finally press on **Next** and **Import**. Some requirements on the zip file concern the total size which must not exceed 50MB, UTF-8 encoded, no more than 100 files together can be uploaded and must be in JavaScript or Python language in the lambda folder.

## Export-Import an existing project

---

Select .zip archive with lambda folder:

alexa\_skill.zip

Select files: You can import up to 100 files totaling 6MB.

Name	Size	Action
<input checked="" type="checkbox"/> /	50.3 KB	
<input checked="" type="checkbox"/> lambda/		
<input checked="" type="checkbox"/> constants.js	357 Bytes	New
<input checked="" type="checkbox"/> index.js	36 KB	Overwrite
<input checked="" type="checkbox"/> interceptors.js	3.1 KB	New
<input checked="" type="checkbox"/> local-debugger.js	4.9 KB	Overwrite
<input checked="" type="checkbox"/> localisation.js	2.8 KB	New
<input checked="" type="checkbox"/> logic.js	1.9 KB	New
<input checked="" type="checkbox"/> package.json	435 Bytes	Overwrite
<input checked="" type="checkbox"/> util.js	879 Bytes	Overwrite
<input type="checkbox"/> skill.json		

Cancel

Back

Next

# Chapter 8

## Conclusion

In this project, the aim is to combine a website containing oral stories with the potential offered by the voice assistant Alexa. [tiraccontounastoria.org](http://tiraccontounastoria.org) offers an audio recording service via a telegram bot with saving on the server and listening to the previously recorded stories directly from the URL.

Alexa allows interaction with the user without the help of a PC to make it easier to search for potential stories to listen to.

The Python code development on the back-end of the server receives the requests and manages the data to be processed to be sent in JSON format to Alexa.

In the Alexa part, there is a first part where intents have to be created, i.e., sentences that the user potentially says depending on the type of request made. The second part concerns the handling of these intents, i.e. what Alexa will say according to the invoked intent, and is developable in JavaScript or Python. It is possible to play an mp3 file by searching for information on the people present or the stories available, or to search for a specific topic of one or more words. Furthermore, using the basic and suitably modified functionality, it is possible to manage the playback and pause of the audio file.

There are some limits that are imposed directly by Alexa and others that are somehow solvable. It is possible that, in the future, these will be somewhat improved in the AI or other mechanisms will be introduced/updated (such as new types of slots, a single integration between all the various intents or whatever) for a much more user-friendly use than it already is.

The work done is only the beginning. Listening to an audio message is now possible, interaction with Alexa is the basis of everything, but what is possible from the site is much more. What this part of the project involves is listening to stories that are public and audible to everyone. But some of these stories can go back

to being private, others always have been. A login phase also on the Alexa app will be necessary so that the private part of that user can also communicate with the hidden stories. A better interaction for searching stories when they are shared. The title of the story is always the same, but not who is telling it. A possible future implementation could also be in the graphical user interface. Improving the way the users can view, choose, interact on devices that have a display showing the results. For example, providing the user's display with a cover of the image used for that story, flanked by the person who told it. This is something that should always be reviewed over time, as the improvements being made on these devices are constantly evolving.

# Appendix A

## Acknowledgments

Mi è doveroso dedicare questo spazio della mia tesi a tutte le persone che mi hanno supportato nel mio percorso di crescita universitaria e professionale.

Innanzitutto, un grande ringraziamento al mio relatore Cabodi Gianpiero e al mio correlatore Lavagno Luciano, pronti a guidarmi in ogni fase della realizzazione della tesi. Un grazie anche al Dott. Pasini Paolo, con il quale ho accresciuto le mie conoscenze e le mie competenze.

Un doveroso ringraziamento anche ai miei genitori, senza loro sicuramente non sarei arrivato così lontano. Un ringraziamento a mia sorella Lucrezia, perchè anche quando era più lontana era sempre vicina nei miei giorni più bui. Non voglio dimenticare entrambe le mie due fantastiche nonne, che si occupavano di sfamarmi e che senza loro non avrei avuto la forza di portare a casa questi risultati.

Un ringraziamento anche a Max, compagno universitario che si rivelato molto di più. Un vero amico nei momenti di bisogno e un buon alleato nello studio, tra le mille risate e gli infiniti pianti. Non voglio dimenticare anche i miei amici della triennale e magistrale: Ale, Beppe, Cri, Fra, Luca e Albi.

Poi gli amici di sempre, quelli che non hai bisogno di sentirli sempre ma ci sono nel momento del bisogno. Davide, Riccardo, Brusco, Dani e pochi altri. Un infinito grazie va anche a loro che mi hanno sopportato (ma anche supportato) dal primo giorno in cui ho stretto loro la mano.

Non voglio dimenticare tutto il vicinato che ha sempre e costantemente creduto in me, incoraggiandomi sempre più nell'arrivare al mio traguardo e a farmi pesare meno i giorni passati in casa.

Infine volevo ringraziare anche coloro che non hanno creduto in me, per dimostrare loro che, a volte, la perseveranza e insistere su ciò che si ritiene corretto e giusto per se stessi porta al risultato ottenuto.