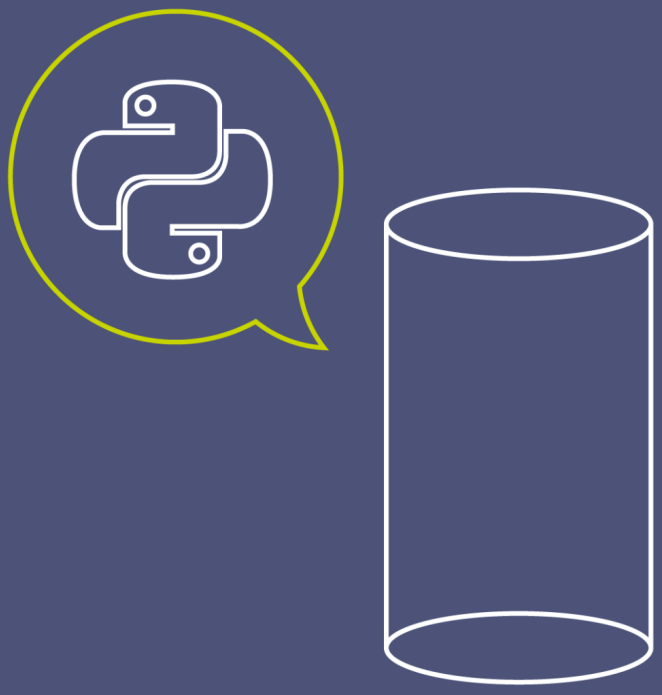
THIS GUIDE ASSUMES USING **DJANGO** AND **PYTHON**

IT IS EASY TO CREATE ALEXA SKILLS

...IF YOU KNOW HOW TO DO IT!



by Raffaele Colace

20tab

Introduction

If you are reading this e-book, then you most probably are a Python[1] developer using Django[2]

And I bet you also are a fan of new technologies, just as I am.

This e-book stems from the curiosity and the desire to implement Alexa skills in a simple way while being able to test and debug in the fastest way possible.

Speaking about Alexa, it has already been a while since Amazon started bombarding us with its Echo devices. Without further ado, let's go see what this voice assistant is about and how to use it. We will later create our skills and enrich it with **custom features**.

The present e-book focuses specifically on the technical side and aims to make life easier for those developers who choose to go down my same path.

Shall we start?

What is **Alexa?**

It is a virtual assistant distributed by Amazon which, through the various Amazon Echo devices, allows you to control several objects, services and contents just by using your voice.

As far as domotics[3]
However, what is truly interesting is being able to implement the skills which are most useful to us.

What is a skill?

A **skill** is an "ability" integrated into Echo devices allowing Alexa to carry out **specific operations**. To make it simple: a skill is to Alexa as an app is to smartphones.

For instance, if you want to use Alexa to know which are the latest articles on our blog, we can implement a skill dealing with this specific behaviour.

We will use **Django** and **Python** to accomplish this.

Django is one of the best open source web frameworks written in Python. In my company we always use it for our projects since it is fast and easy to use; that is why we will try to make the most out of it in this particular case too.

Prerequisites

Let's try and dive a little deeper into the technical details by finding out how to use Django for our purpose. Before we start, though, we need to make some preliminary remarks.

- In order to implement a skill, you first need to have an Amazon developer account[4]
- It is clearly important **to know Python**; on the other hand, knowing Django so well is not essential.
- Lastly, you should know how to use ASK CLI[5]

This e-book aims to describe an approach addressed to those developers who love **coding via the terminal**: that is why I am going to use Amazon consoles as little as possible. Of course, there is nothing to prevent from using the instruments I am going to present with the aid of those tools.

However, before we start talking about code, I would like you to understand which was the main reason that led me to choose this approach.

Debugging and testing quickly on the local machine

I implemented my first skill using Amazon tools and guides and created it, by the book, via the different consoles[6] **available.**

When I ran into the first errors, I found it particularly hard to work with Alexa and AWS Lambda consoles to see my logs, prints, etc. I even considered using AWS-SAM-CLI.[7]

Either way, it was way too cumbersome for me, since I am, let's say it, kind of a lazy guy.

The issue I wanted to solve, thus, consisted in being able to see on my terminal the logs my skill would produce as a result of the user interaction.

Anyway, my first experience led me to the following steps:

1. Implementing a functionality
2. Deploying
3. Testing via the console

All of the above required a few minutes' wait: deployment was **slow** and tests implied **multiple clicks** on the browser. Not to mention debugging, which forced me to change page and go on AWS Lambda console.

I would say that frustration and long waits are good enough reasons for trying to **find a more satisfying approach.** Let's see, then, how to implement a skill from scratch and let's try to figure out how to improve the whole process.

How to create a skill from scratch

Let's suppose you want to stay up-to-date on the latest news on our blog. We will then have to implement a skill which will read you the latest published article.

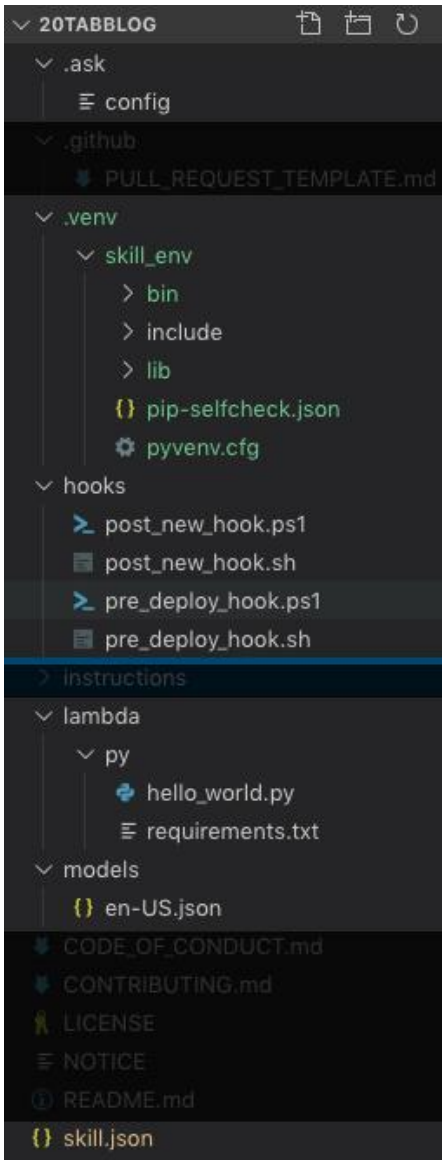
First of all, you need to **install ASK-CLI**. After having installed it and configured it with your account, you only have **to create a new skill** with the terminal command:

```
$ ask new
```

You will then have to choose the language to be used and a template from which to start. We will opt for **Python3**, obviously, and a simple class-based Hello World.

The project structure, at this point, will look like the following image:

Let's analyse the files and directories we are facing:



- **.ask**: directory containing ASK CLI configuration file.
- **.venv**: **virtualenv** used by the **lambda** function.
- **hooks**: contains the "post_new_hook" and "pre_deploy_hook" script hooks. Everytime you fire a "ask new" or "ask deploy" command, the execution of one of these hooks starts.
- **lambda**: contains all the source code executed by the lambda function and will be the file where we are going to write our Python code.
- **models**: are defined in this directory in **JSON** format. This is nothing but the file in which the "Intents" are defined for every language.
- **skill.json**: contains the skill manifest[8]

For all the details concerning the skill files, please see the official documents. The code we have seen in the example, on the other hand, can be found at the following link: https://github.com/rafleze/20tab-blog-skill

Our goal consists in succeeding at **debugging and testing** while we are developing our skill, **straight on our computer**, without necessarily using all of Amazon's consoles.

How to configure a Django project that can handle requests from Alexa

Now, let's see how to directly connect our skill to our **Django** project.

First of all, you need to make sure you have Django installed on your machine, or, better yet, in a virtualenv[9].

```
$ virtualenv --python=python3 20tabblogenv
```

```
$ pip install django
```

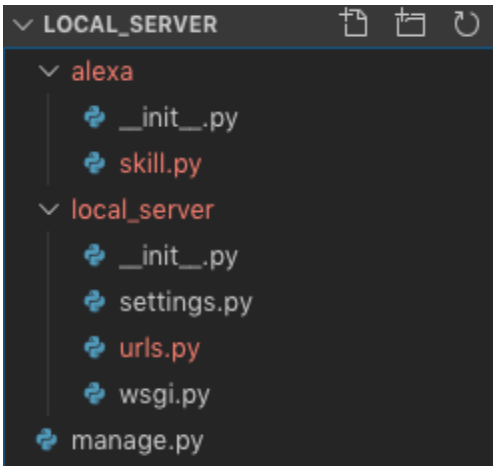
Right after that, we can create the project:

```
$ django-admin startproject local_server
```

In order to use this project as a custom webservice[10], we will have to install in our virtualenv a library which will surely make our life easier: django-ask-sdk[11].

```
$ pip install django-ask-sdk
```

At this stage, we will create a Django application, containing the lambda function, in which we will implement all the necessary handlers. Our project will have a structure similar to the following one:



skill.py will contain all the handlers which are necessary for our purpose.

However, the most important thing is remembering to expose in the **urls.py** file an endpoint which is able to accept requests from our skill.

```
from django.urls import path
from alexa.skill import skill
from django_ask_sdk.skill_adapter import SkillAdapter

my_skill_view = SkillAdapter.as_view(
    skill=skill)

urlpatterns = [
    path('', my_skill_view, name=' index')
]
```

We can finally start our local server. The problem is Alexa requires a public secure endpoint with **https**. That is why **ngrok** comes to our aid when we execute the command

```
$ ngrok http 8000
```

```
Session Status                 online
Account                        Rafleze (Plan: Free)
Version                        2.3.34
Region                         United States (us)
Web Interface                  http://127.0.0.1:4040
Forwarding                     http://7e6afdd5.ngrok.io ->
http://localhost:8000
Forwarding                     "https://7e6afdd5.ngrok.io" -
> http://localhost:8000

Connections                    ttl  opn  rt1  rt5  p50  p90

                               0    0    0.00 0.00 0.00 0.00
```

At this stage, we only need to change the configuration of the Alexa skill endpoint in the skill.json file.

```
"apis": {
  "custom": {
    "endpoint": {
      "uri": "https://7e6afdd5.ngrok.io",
      "sslCertificateType": "Wildcard"
    }
  }
}
```

Let's deploy the skill and try to launch it. What we notice here is that we will have a request on our local server, therefore we will be able to do all the necessary **debugging** directly **on our machine**.

```
System check identified no issues (0 silenced).
September 11, 2019 - 11:14:45
Django version 2.2.5, using settings
'local_server.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[11/Sep/2019 11:20:05] "POST / HTTP/1.1" 200 391
```

The Django project source code can be downloaded here:
https://github.com/rafleze/20tab-blog-django

The steps are few but varied: taking them again and again, I realised that many could be **automatised** and that we could try and structure files and directories so as to have a reusable **boilerplate**.

How to perform local tests in very few steps

Each and every stage is good for figuring out what happens during Alexa skills developing process using Django as a local web service. But after having understood all of them it is important **to speed this procedure up**.

I did this by creating a boilerplate which allows me to implement a skill logic in two simple steps without wasting any time with multiple different configurations. For this purpose I used cookiecutter[1][2] which made my life so much easier.

Here https://github.com/20tab/django-alexa-template you can find the Boilerplate and a mini guide on how to use it.

One of the most interesting aspects about this procedure was being able to use **Django Test Suite** to test lambda function handlers in a simple and quick way.

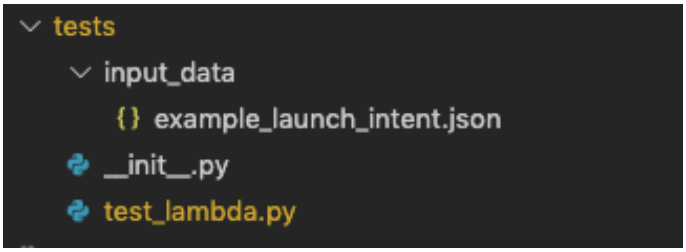
How to structure a skill test with **Django Alexa Template**

The working method we use in our company implies writing tests for all the projects we manage.

We like to use the **TDD** (test-driven development) model when we code, while **BDD** (behaviour-driven development) carries us through the whole development process.

In this respect, please find here a link to a very interesting video on how we structure some of the tests: https://www.20tab.com/blog/dalle-user-stories-ai-test-automatici-di-django/

Of course, this case could not have been different. The boilerplate contains a directory with a test example. Here is how the structure looks like:



In order to write and structure a test, we need to know what happens every time the skill is invoked.

When the user asks Alexa to launch a skill, the service makes a standard http POST request, sending a JSON payload (for details on the payload please see the documentazione ufficiale[13]).

It is worth knowing that every payload contains information about the calling skill, the session, the context and the user. It also provides us with data on the type of request, as you can notice from the following code portion:

```
"request": {
  "type": "LaunchRequest",
  "requestId": "",
  "timestamp": "2019-08-31T09:03:15Z",
  "locale": "it-IT",
  "shouldLinkResultBeReturned": false
}
```

All of this is intercepted by the lambda function, which deals with launching the correct **handler**.

Now that we have understood what happens, we can structure our first test. First of all, we need to have a payload to send to our server. We can see an example in the “input_data” directory.

Our test then has to send a post request to our service and check whether the result is as expected, like below.

```
from django.test import Client, TestCase
from django.urls import reverse
from unittest.mock import patch
import json

class TestSkill(TestCase):

  def setUp(self):
    self.client = Client()

  def get_response(self, filename):
    with open(f'tests/input_data/{filename}', 'r') as f:
      payload = json.load(f)
      res = self.client.post(
        reverse("alexa_skill"), data=json.dumps(payload), content_type= "json")
      return res

  def test_launch_intent(self):
    res = self.get_response("example_launch_intent.json")
    self.assertIn(
      "Ciao",
      res.json()['response'][ 'outputSpeech'][ 'ssml']
    )
    self.assertEqual(res.status_code, 200)
```

1. The defined TestCase has a **setUp** function in which we are going to establish the client in order to make https requests.
2. We use the **get_response** function to obtain this result, passing the file containing the information about the request as payload.
3. Finally, we implement the test so as to check the good content of the response.

Needless to say, this test is intentionally simple and trivial for demonstration purposes.

Conclusions

The beginning of this adventure was rather frustrating since implementing skills with the ordinary tools required many clicks.

Django Alexa Template[14] , on the contrary, allows us to create, publish and test our skills in very few steps, thus saving a lot of time.

Now the adventure becomes fun, interesting and, most of all, useful.

Start implementing skills quickly and right away by using Django Alexa Template. You can find it here: https://github.com/20tab/django-alexa-template

Links and useful resources:

1. Python - [https://www.python.org/]
2. Django - [https://www.djangoproject.com/]
3. Domotica - [https://it.wikipedia.org/wiki/Domotica]
4. Account developer - [https://developer.amazon.com/]
5. ASK CLI - [https://developer.amazon.com/es/docs/smapi/quick-start-alexa-skills-kit-command-line-interface.html]
6. About the developer console - [https://developer.amazon.com/it/docs/devconsole/about-the-developer-console.html]
7. AWS-SAM-CLI - [https://github.com/awslabs/aws-sam-cli]
8. Skill manifest - [https://developer.amazon.com/es/docs/smapi/skill-manifest.html]
9. Virtualenv - [https://virtualenv.pypa.io/en/latest/]
10. Custom webservice - [https://developer.amazon.com/it/docs/custom-skills/host-a-custom-skill-as-a-web-service.html]
11. django-ask-sdk - [https://pypi.org/project/django-ask-sdk/]
12. cookiecutter - [https://github.com/cookiecutter/cookiecutter]
13. Request types reference - [https://developer.amazon.com/it/docs/custom-skills/request-types-reference.html]
14. Django Alexa Template - [https://github.com/20tab/django-alexa-template]
15. Alexa skills kit tutorials - [https://developer.amazon.com/it/alexa-skills-kit/tutorials]
16. Alexa skills kit - sdk for python - [https://developer.amazon.com/it/docs/alexa-skills-kit-sdk-for-python/develop-your-first-skill.html]

The author

Raffaele Colace, passionate about Agile and Lean methodologies, obsessed by programming best practices and able to code in different languages such as Swift, C#, Java.

He graduates in Computer Engineering at Rome's "La Sapienza". Right after that he becomes one of the co-founders at 20tab srl – the company in which he has gained most of his technological experience.

He has managed multiple software development projects – from startups to multinational corporations – both from the programming and the project management side, always accomplishing them with excellent results.

He is on a constant quest for new things to learn and to experiment on, with the aim of increasing his know-how.

Linkedin: /in/raffaele-colace/