**Using APL to display on an Alexa Echo Screen using python - June ‘20**

I wanted to display an output on an Echo screen (e.g. spot or Show, or even Fire TV) using python. There are lots of NodeJS example and a few complicated python examples, but nothing (that I could find) that walk you through, or gave a full example of a response showing where and how to add the APL (Alexa Presentation Language) code.

Hopefully this will help you.

I should say at this stage there are two ways you can display on the screen. This confused me for a long time. You can use APL which is very flexible (as we will do) or you can use the Display directive and templates which may serve your purpose. These are explained at: <https://developer.amazon.com/blogs/alexa/post/7c9b6bea-0d82-4482-96ba-d1935c2617b9/how-to-quickly-update-your-existing-multimodal-alexa-skills-with-the-alexa-presentation-language>)

If APL is new to you, take a look at the blog above, or the webinar at:

<https://build.amazonalexadev.com/get-started-with-the-apl-on-demand-registration-ww.html>

There’s another excellent Cami Williams video but I can’t find it right now

There are three ways to create an Alexa Skill

1. Using Alexa Hosted skill – this is most probably the easiest way. You don’t need an AWS account – you can just get started.
2. Using AWS Lambda and the Developer Console (the ‘traditional’ way)
3. Using an IDE, and ASK CLI (Alexa Skills Kit and Command Line Interpreter)

I will try to produce a version for all three, but now we’ll use the third one, which is the most involved. I found this NodeJS example by Stuart Pockington (SP) which explains it quite well and we will follow this only using python. It’s also a bit out of date.

<https://medium.com/@punkpocko/alexa-presentation-language-293f8cf25a09>

Step 1 – Setup ASK CLI

Follow the instructions in the SP blog, or see <https://developer.amazon.com/en-GB/docs/alexa/smapi/quick-start-alexa-skills-kit-command-line-interface.html>

Step 2 – Choose a CLI

I will be using the Visual Studio code - follow the instructions in the SP blog, or follow <https://code.visualstudio.com/download>

Step 3 – Let’s start here – start Visual Studio Code

Create a folder for your skill

**mkdir AlexaAPL**

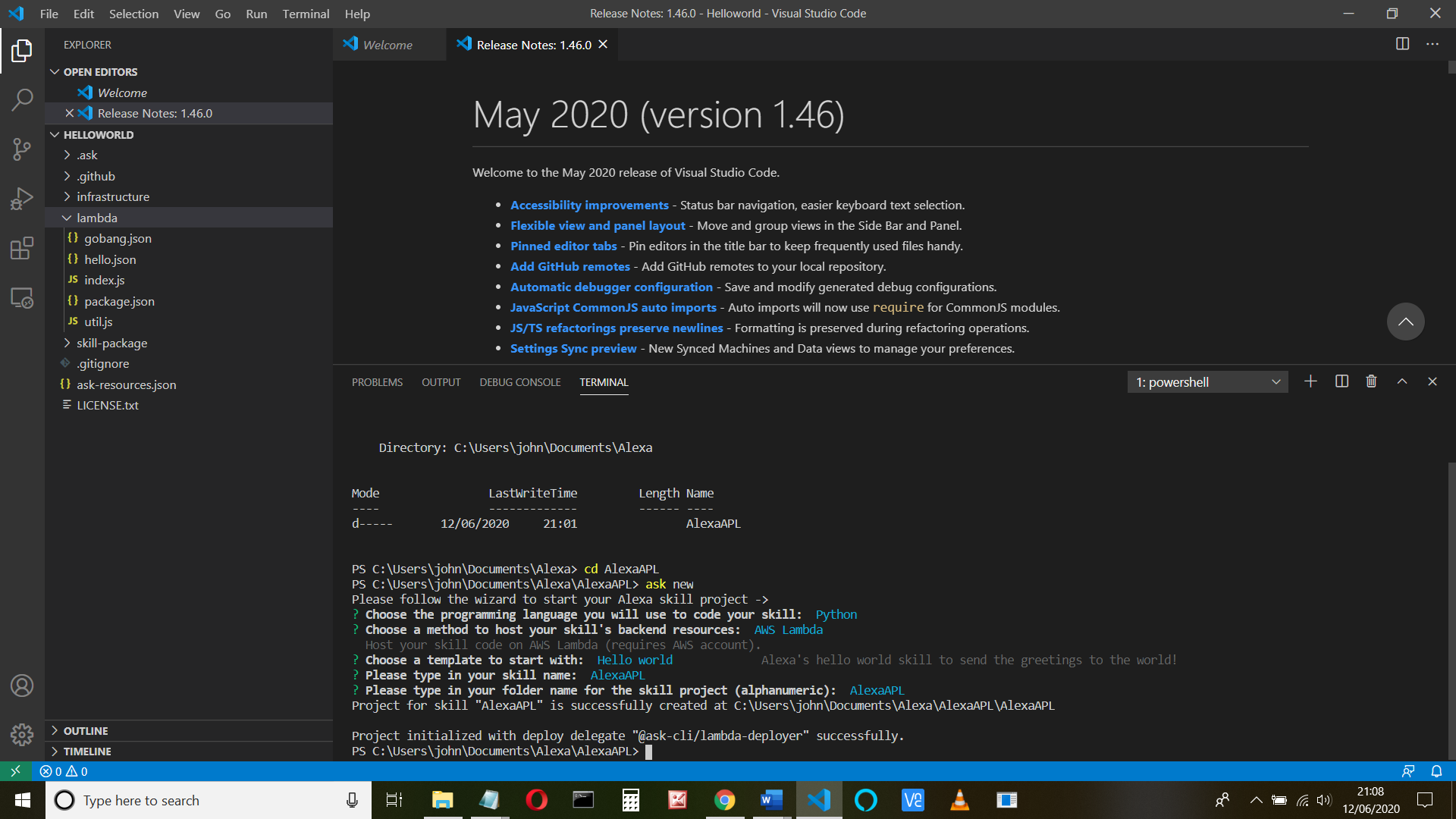
and

**cd AlexaAPL**

to create your skill type

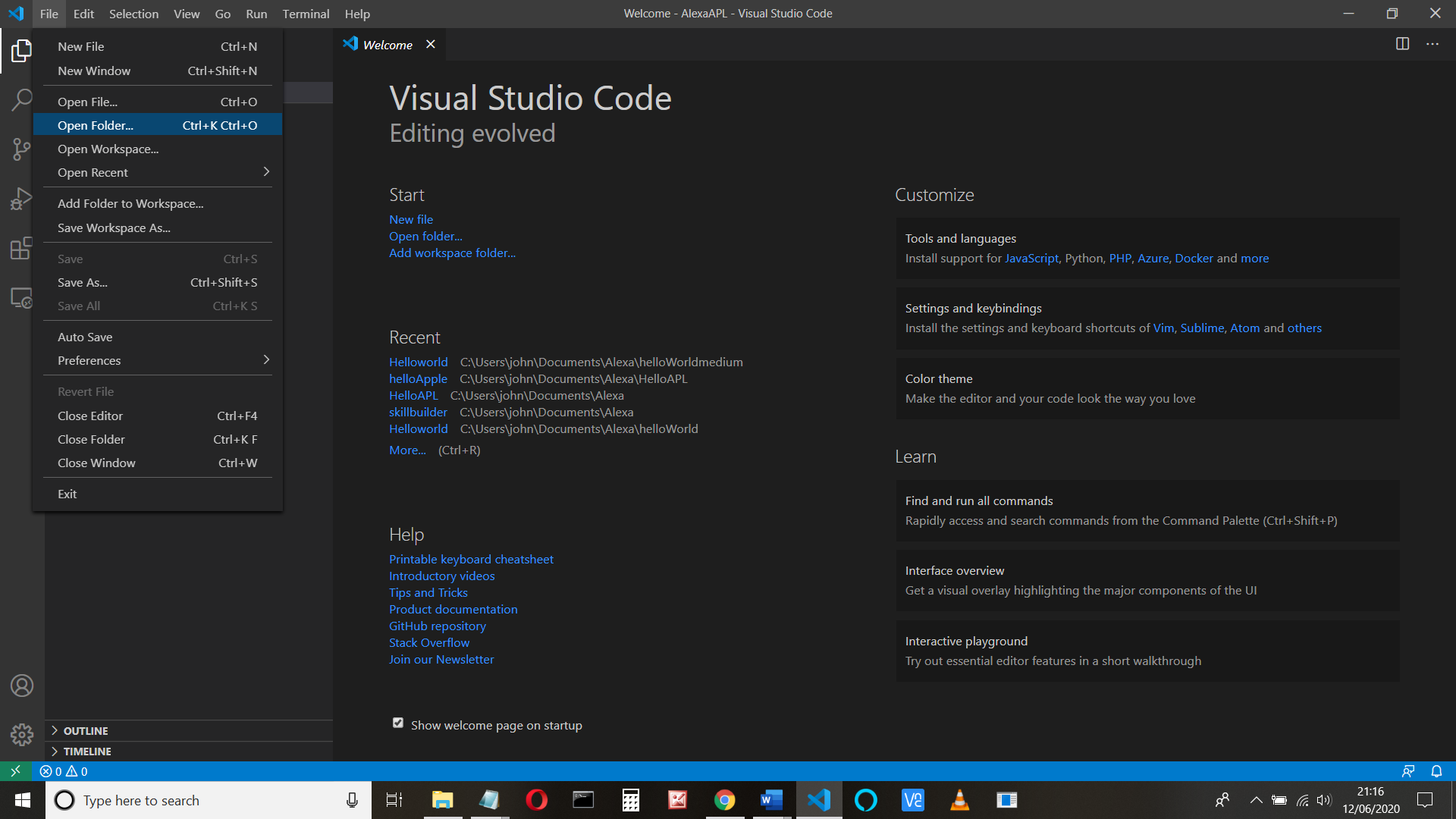
**ask new**

Answer the questions: choose python, AWSLambda (this requires and account but it’s free), Hello world skill and the rest as follows:



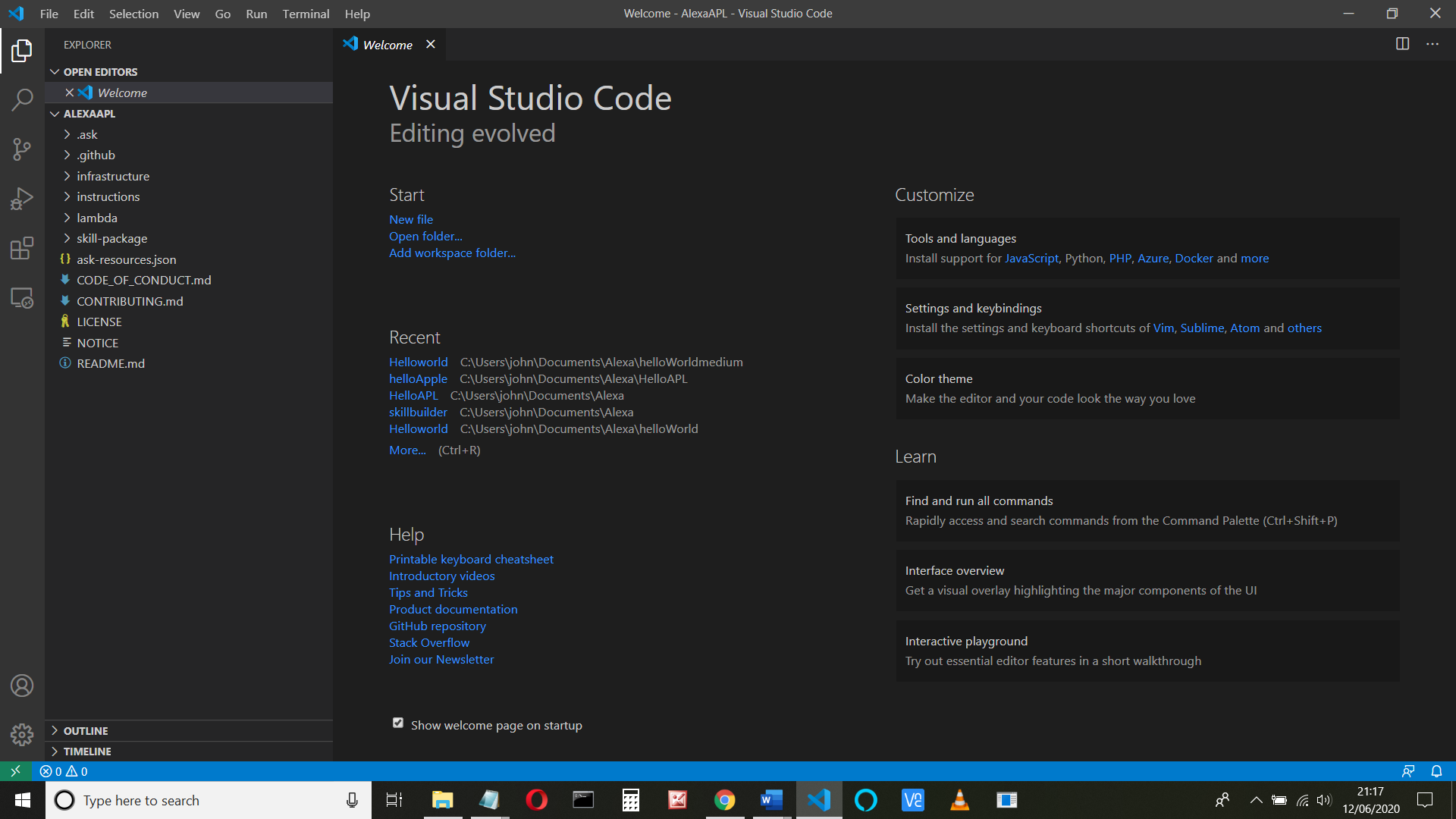
(This actually creates an AlexaAPL folder in the AlexaAPL folder)

Now find the folder in the VS code IDE



Choose File >Open Folder > AlexaAPL > AlexaAPL

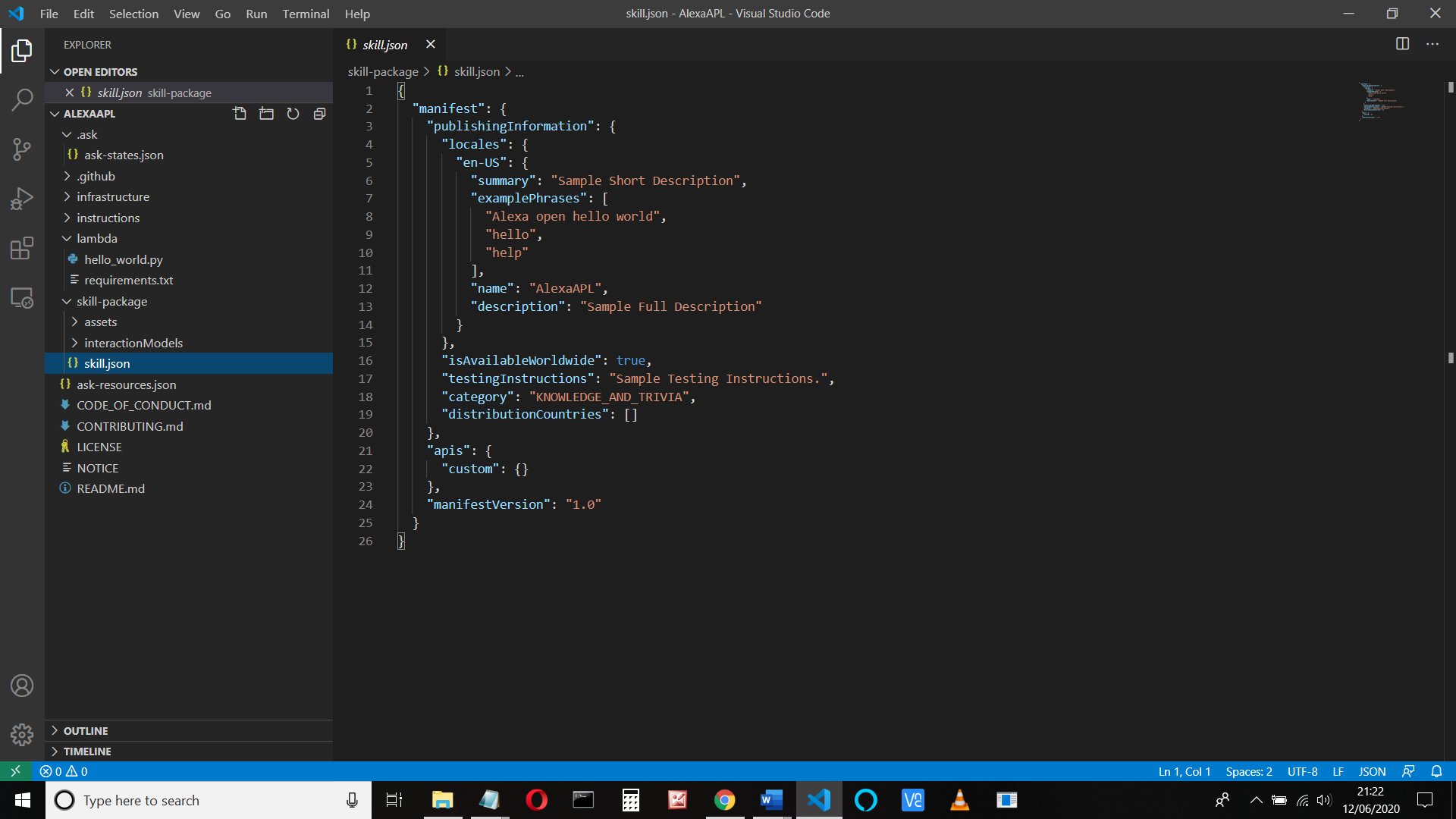
You’ll see a lot of files have been created for you:



These are explained in Step 4 of Stuart Pocklington’s blog, but the ones we want are **skill.json** in the .ask > lambda > skill-package folder.

**The skill.json file**

This holds items such as the name and example phrases:



We need to edit this to add our APL display

This can also be done in the interfaces in the Alexa developer console.

Change the code to add the following in the custom{} code

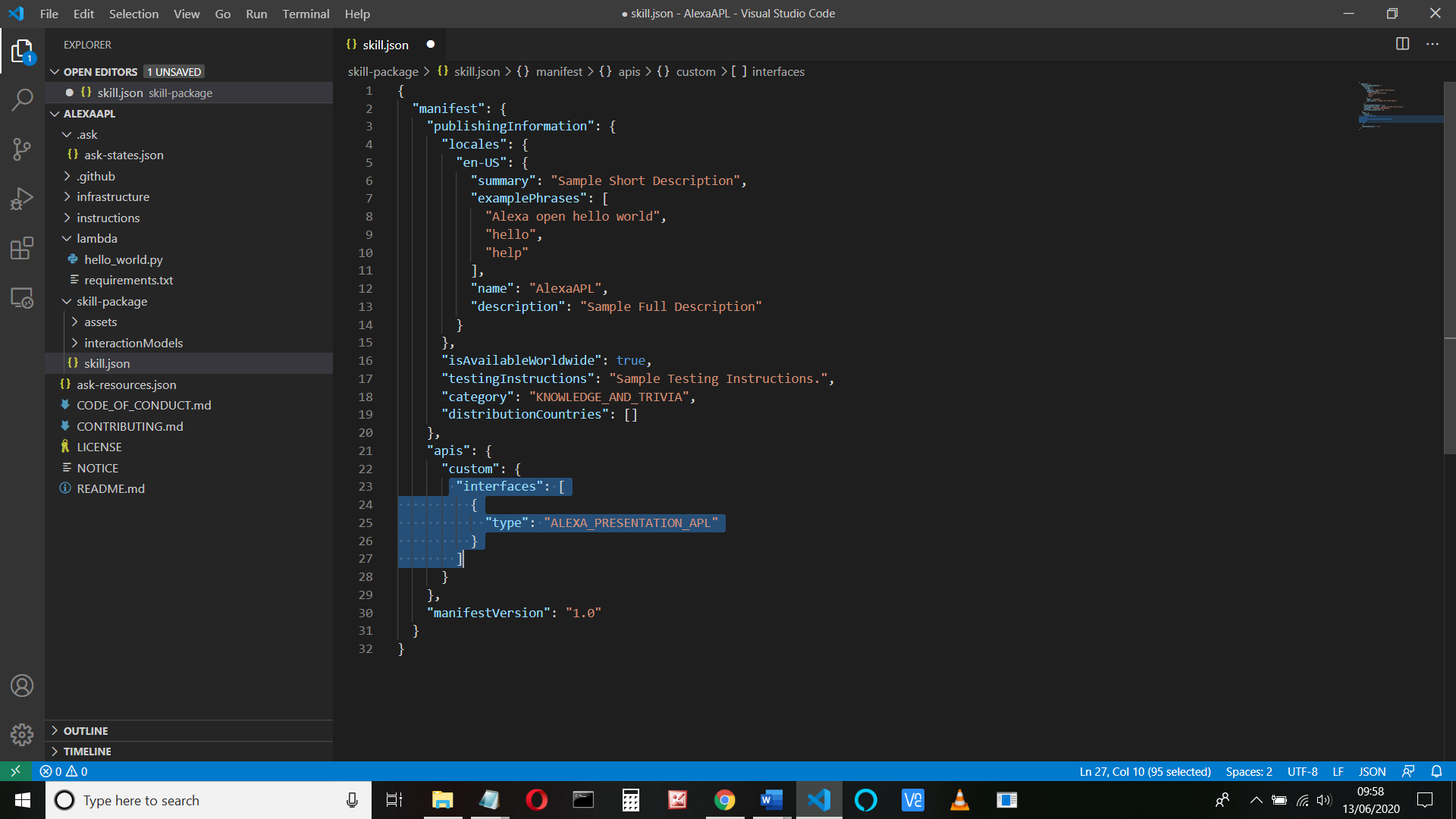
**"interfaces": [**

**{**

**"type": "ALEXA\_PRESENTATION\_APL"**

**}**

**]**



You can also change the “category” from KNOWLEDGE\_AND\_TRIVIA if you want (e.g. to Games, Trivia & Accessories, Education & Reference, Novelty & Humor etc.).

**Save the file.**

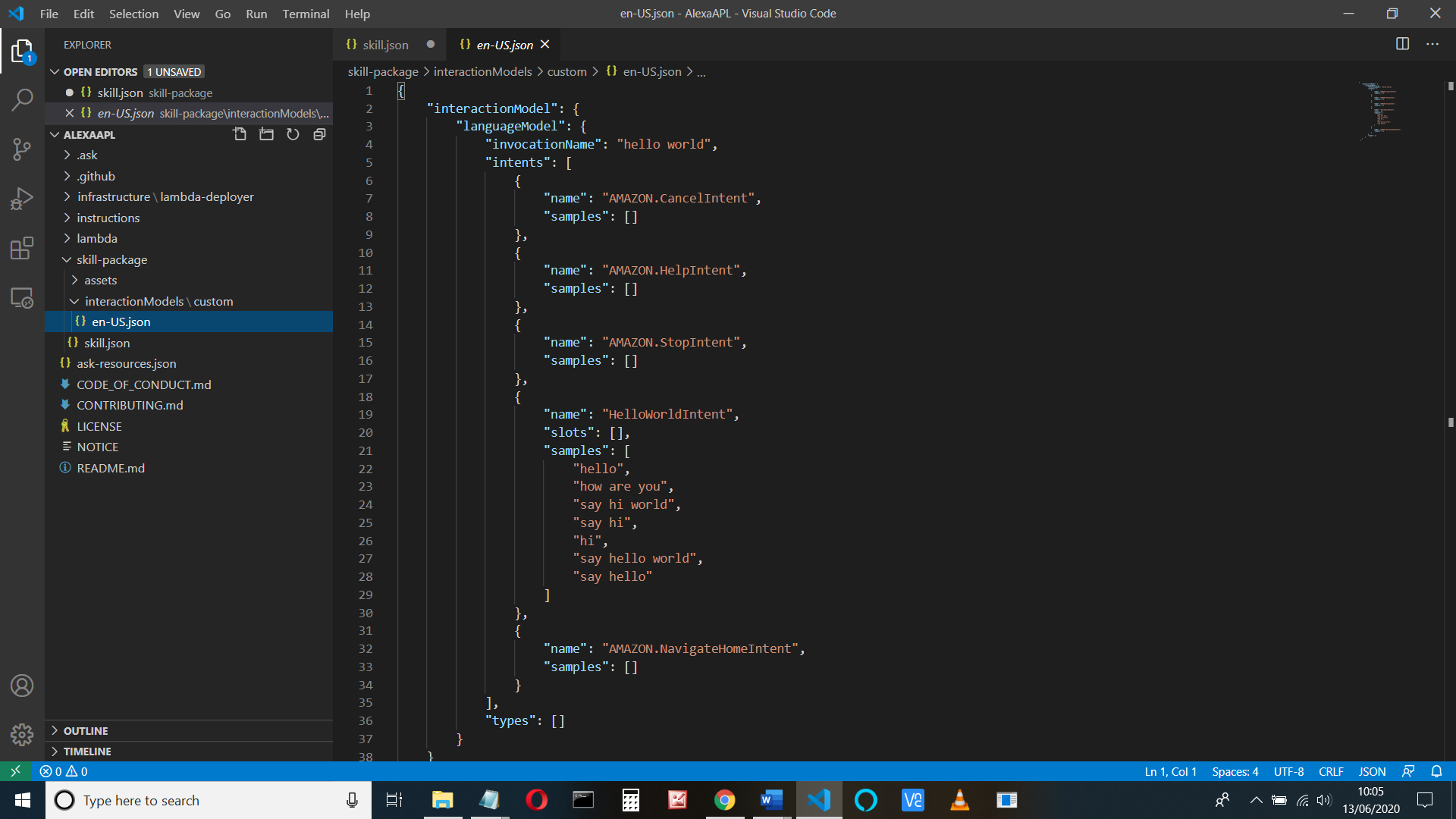
**The model.**

Now move on to skill model held in the **en-US.json** file.

This is found in the skill-package > interactionModels > custom folder

This file contains items such as the invocation word that launches the skill, the commands that the user may say, called **intents** (e.g. ‘hello’ or ‘hi’) and the default intents that Alexa needs (e.g. Stop. Help, Cancel etc.). If you don’t know about these, take a look now. You can see the intents the user can say to call the ‘HelloWorld code, under the HelloWorldIntent

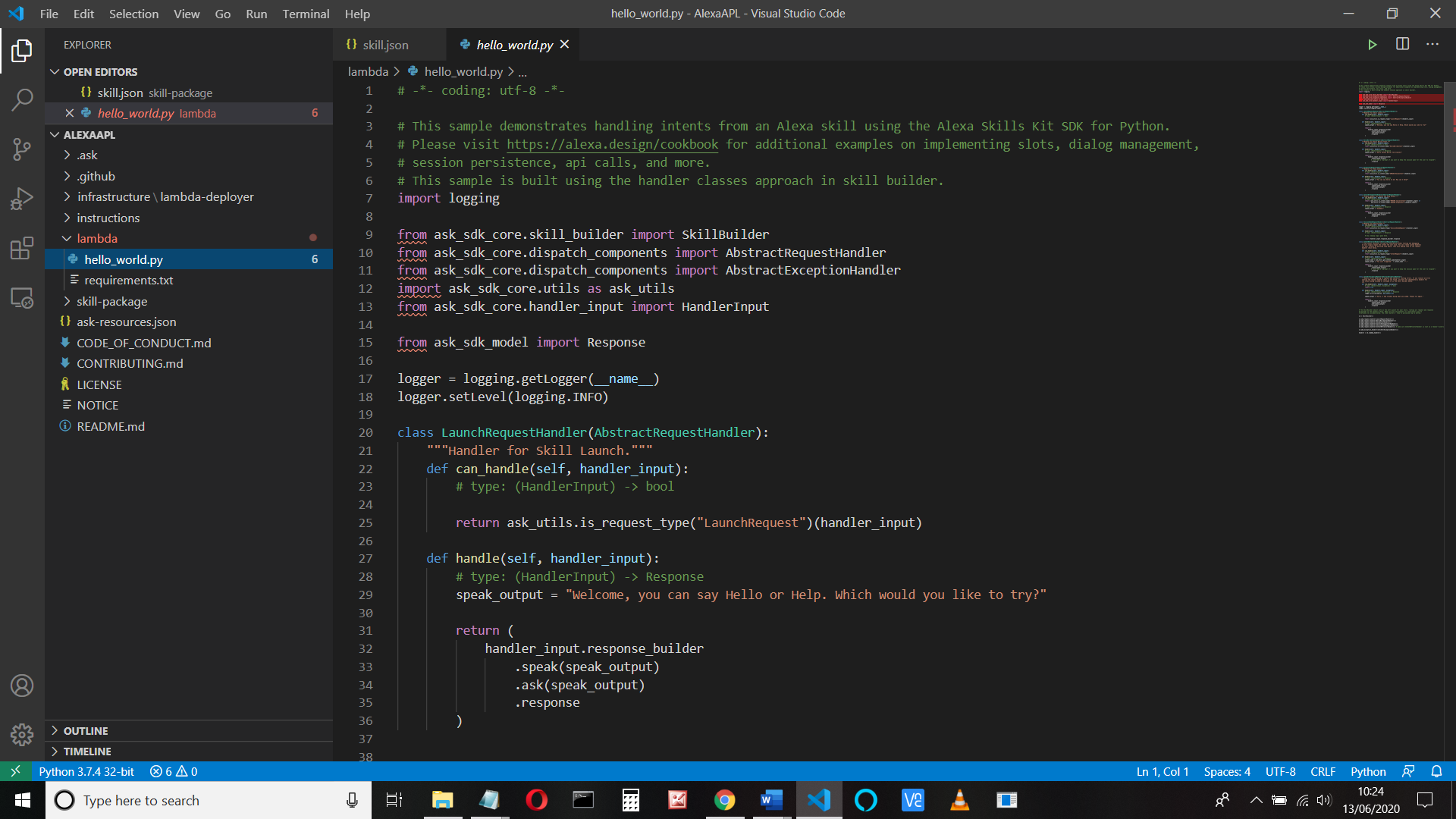
The invocation name is set to what you decided when you created the skill originally ( **"invocationName": "hello world",**)



If you want to add more intents, or change what can be said, this is where you do it. For instance, you could add ‘good morning’ to the HelloWorld Intent samples. If you changed anything, save the file.

**The python code.**

Let’s move on to look at the python code that’s been generated for us. This is the hello\_world.py code found in the **lambda** folder:



Lets’ try to understand a bit about this code first of all.

The entry point of the code is at the end: **handler = sb.lambda\_handler()**

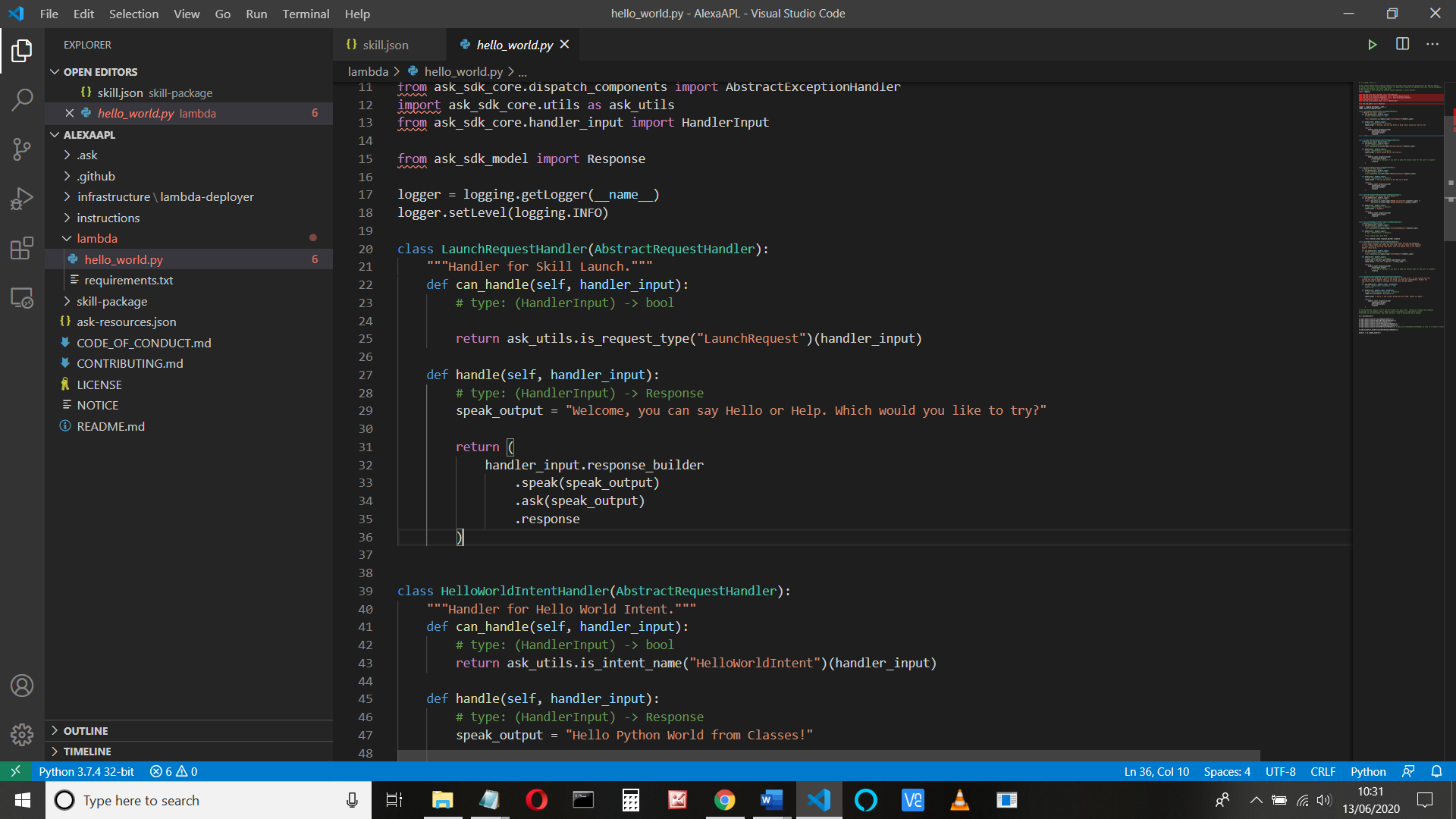
This is the SkillBuilder and you can see from the lines above where all the intents are added. If you want to add you own intent, this is where you do it (as well as adding the actual code further above of course)

Let’s start with the launch request. This code runs when the skill is launched and here you can set up any variables for your code.

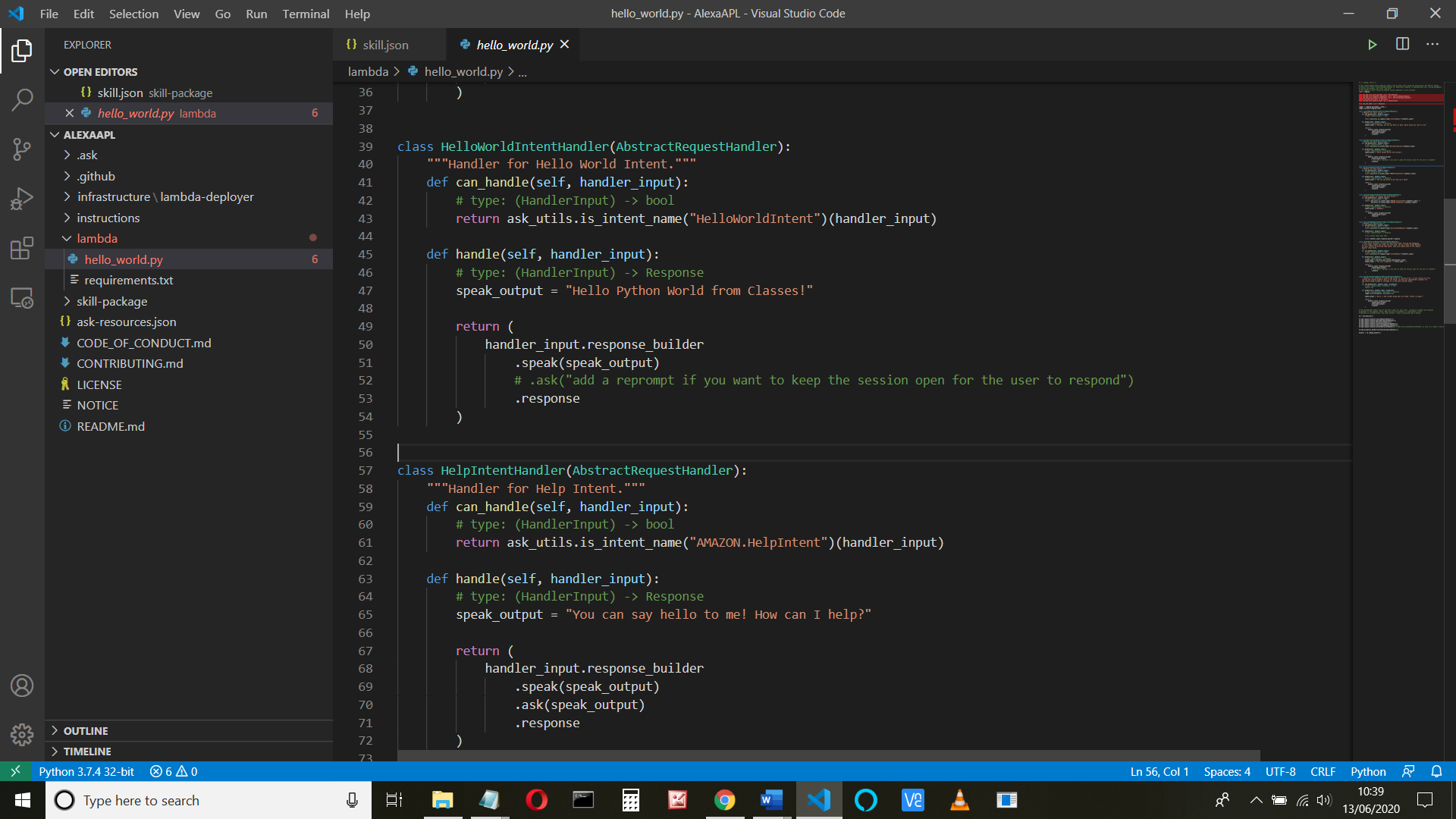
Note the **sb.add\_request\_handler(LaunchRequestHandler())**

Now find the LaunchRequestHandler (at the top of the code)

This code uses the response\_builder code to build a json response (a text file) which is sent back to the Alexa device. The device will say "Welcome, you can say Hello or Help. Which would you like to try?"



When the user says “Hello” (or “hi, or anything else matching the sample phrases in the en-US.json HelloWorid intent list we saw above) the HelloWorldIntent code is executed:



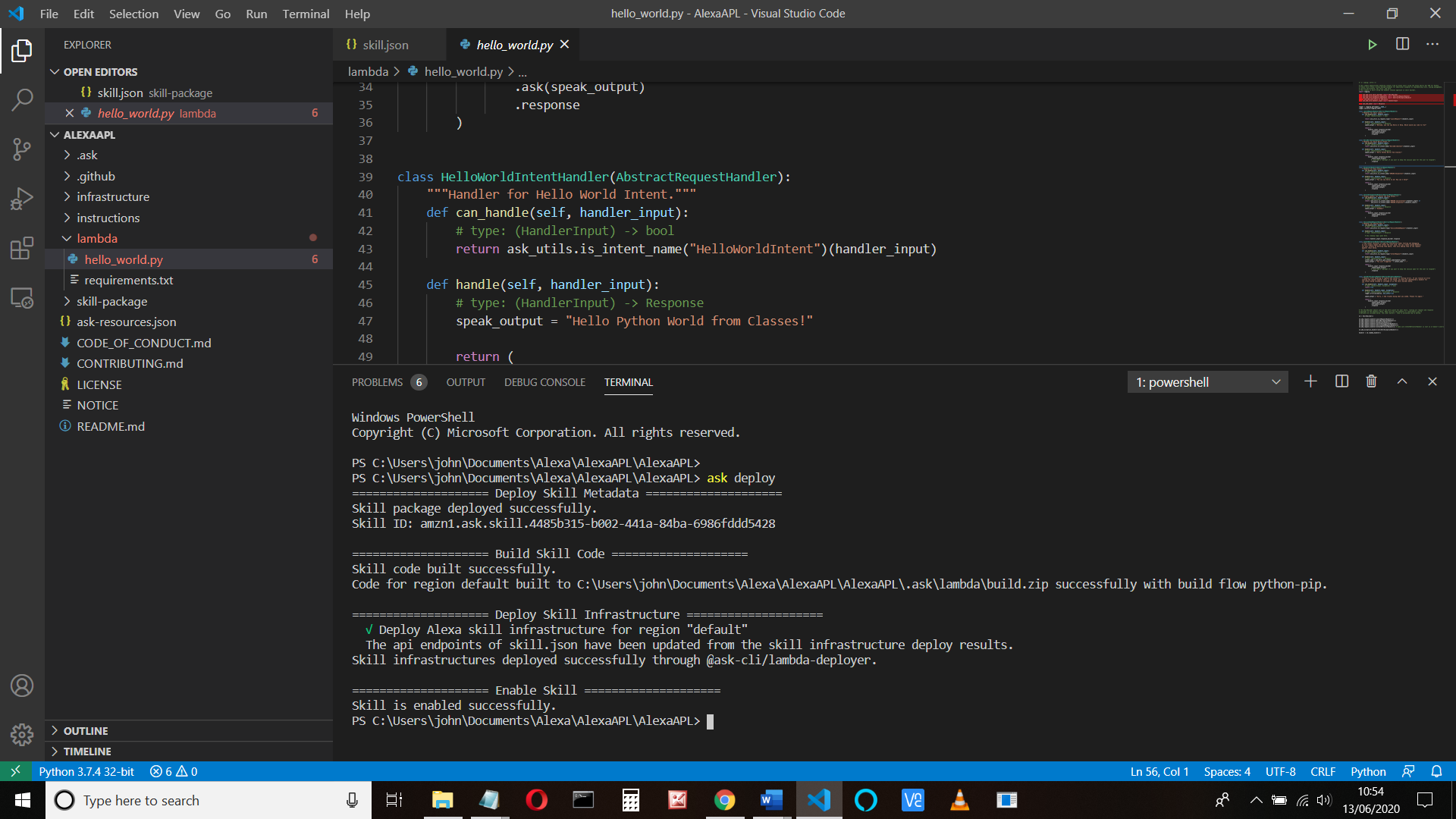
You should be able to work out what your device will say to you when you say hello.

Notice that the **.speak()** code is what Alexa will say, if you want to keep the session open add some code in the **.ask()** code. This is what Alexa will say if there is no response from the user.

**This is where we will add our APL code.**

Before we modify anything, let’s check that our skill works.

Open a terminal in VS code and type **ask deploy**. Don’t worry if you get a flashing screen, that means it’s working.

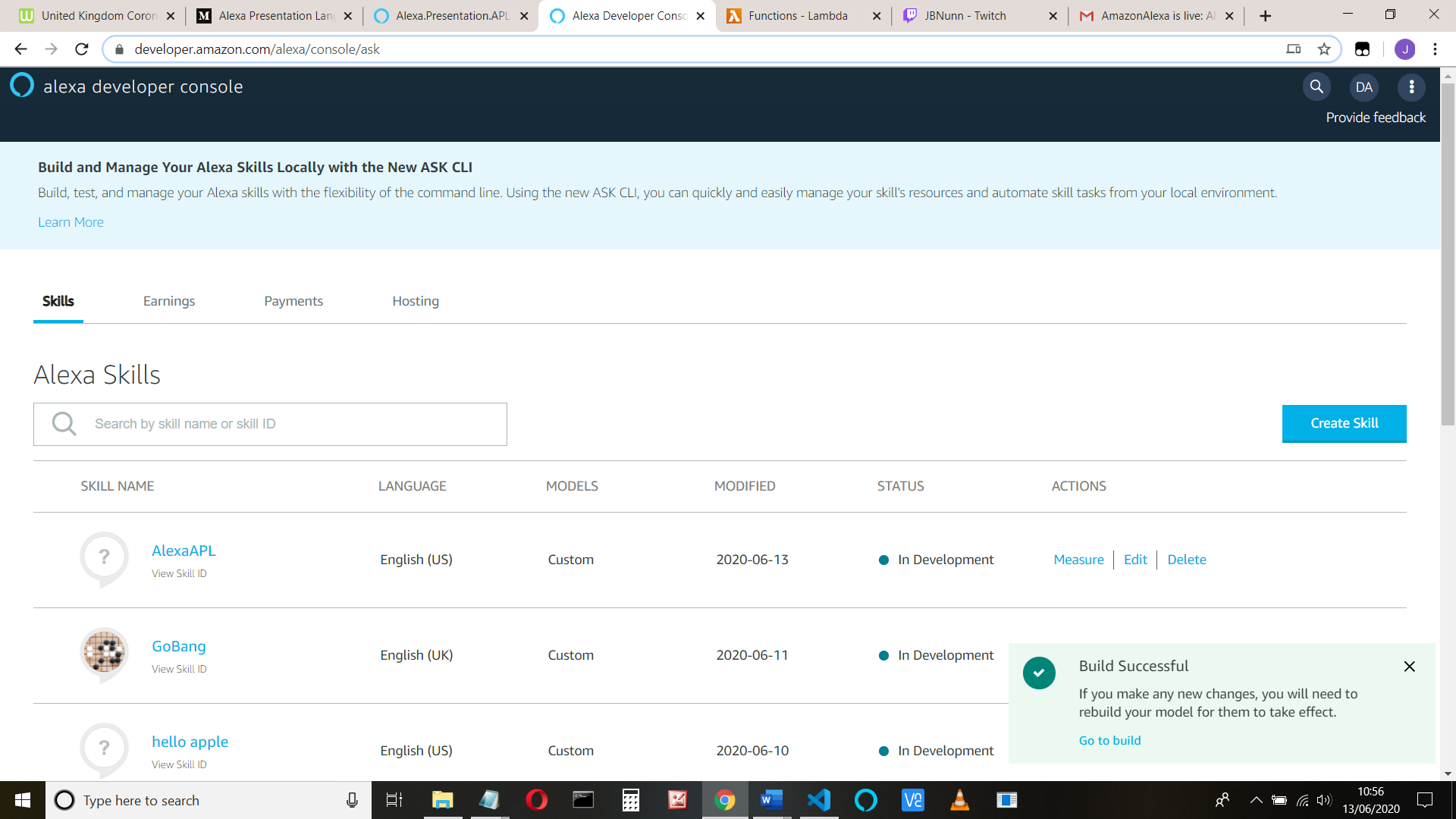


If you get any errors, make sure you are deploying from the correct folder.

Let’s test our program.

Open the developer console at: developer.amazon.com/alexa/console/ask

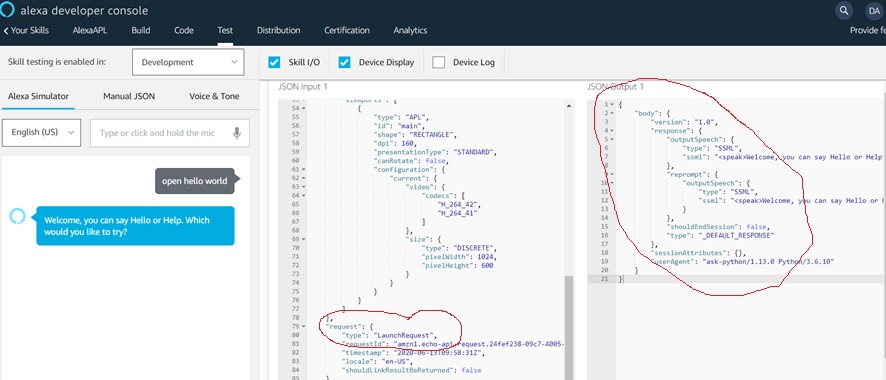
Your skill should be there:



Open the skill, click the Test tab, and enter ‘**open hello world**’ (or whatever you used as your invocation). If you’ve forgotten go to Build > invocation.

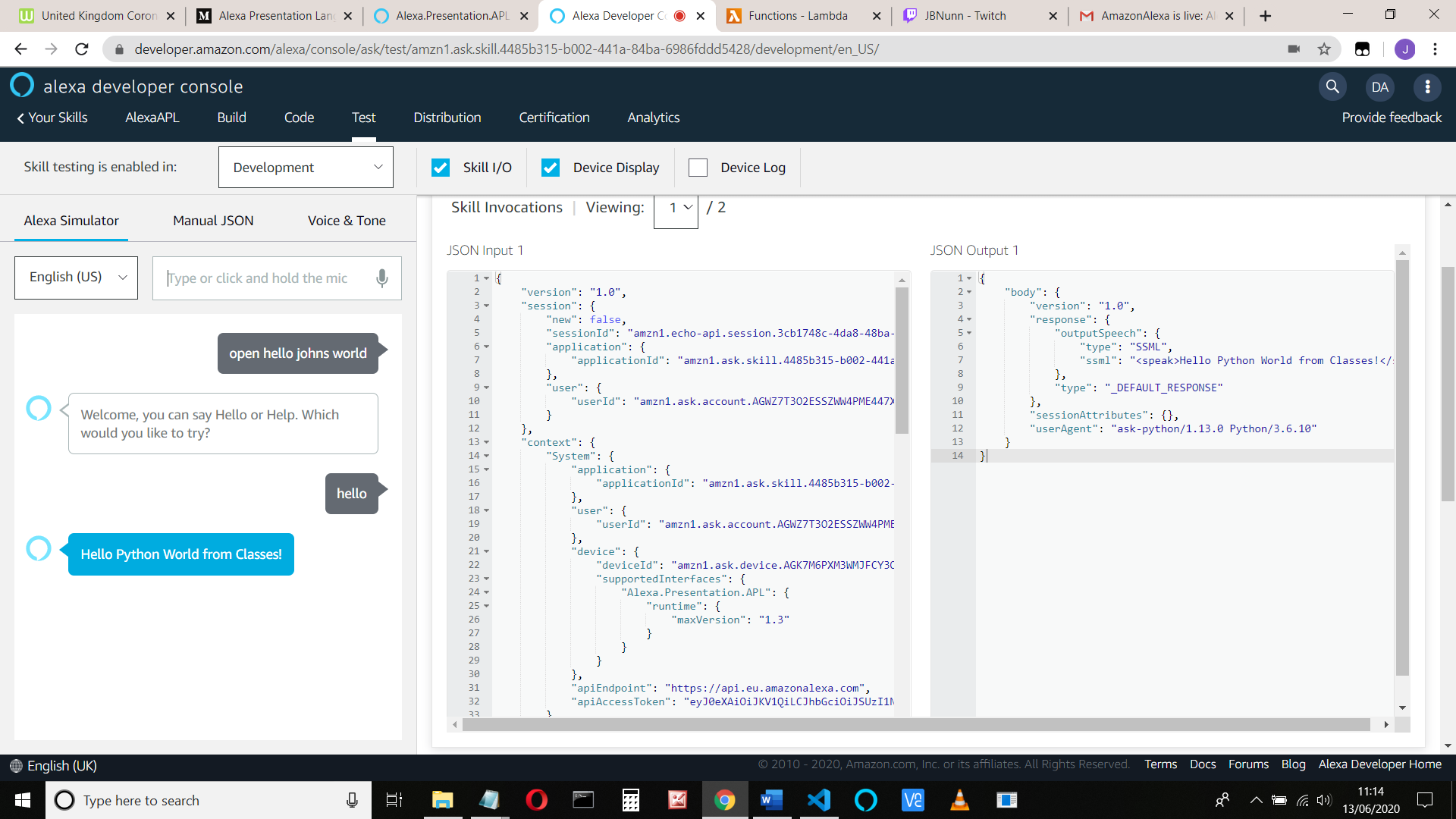
Be careful if you have more than one skill that’s invoked with the same phrase!

You should see:

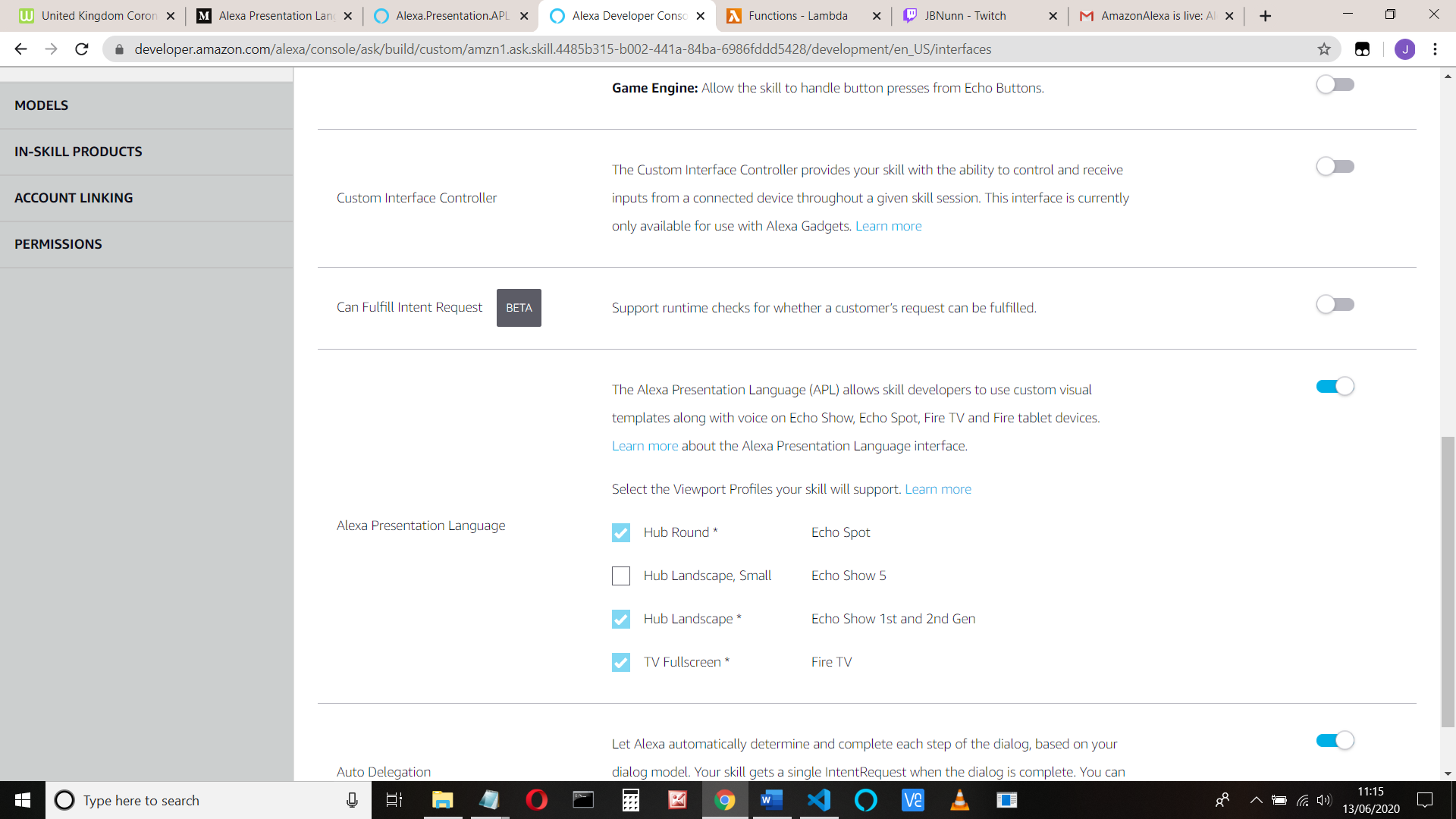


We can see the LaunchRequest and the response from our code. If you now type ‘Hello’ you can see the Hello intent invoked and the response.

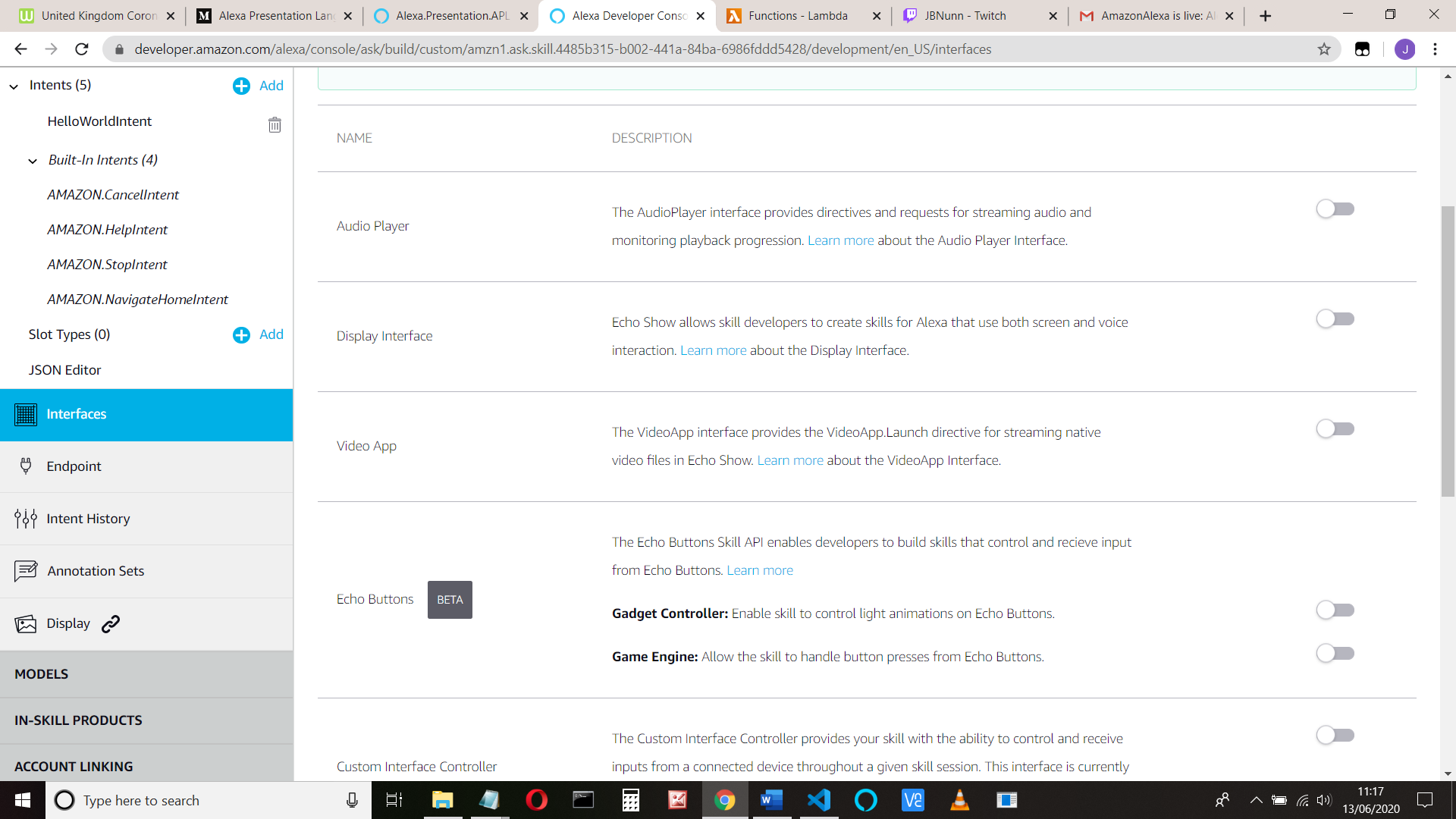
Why don’t you try the skill on your device?



Whilst you are still in the developer console go to Build > Interfaces and note that the Alexa Presentation Language is selected.

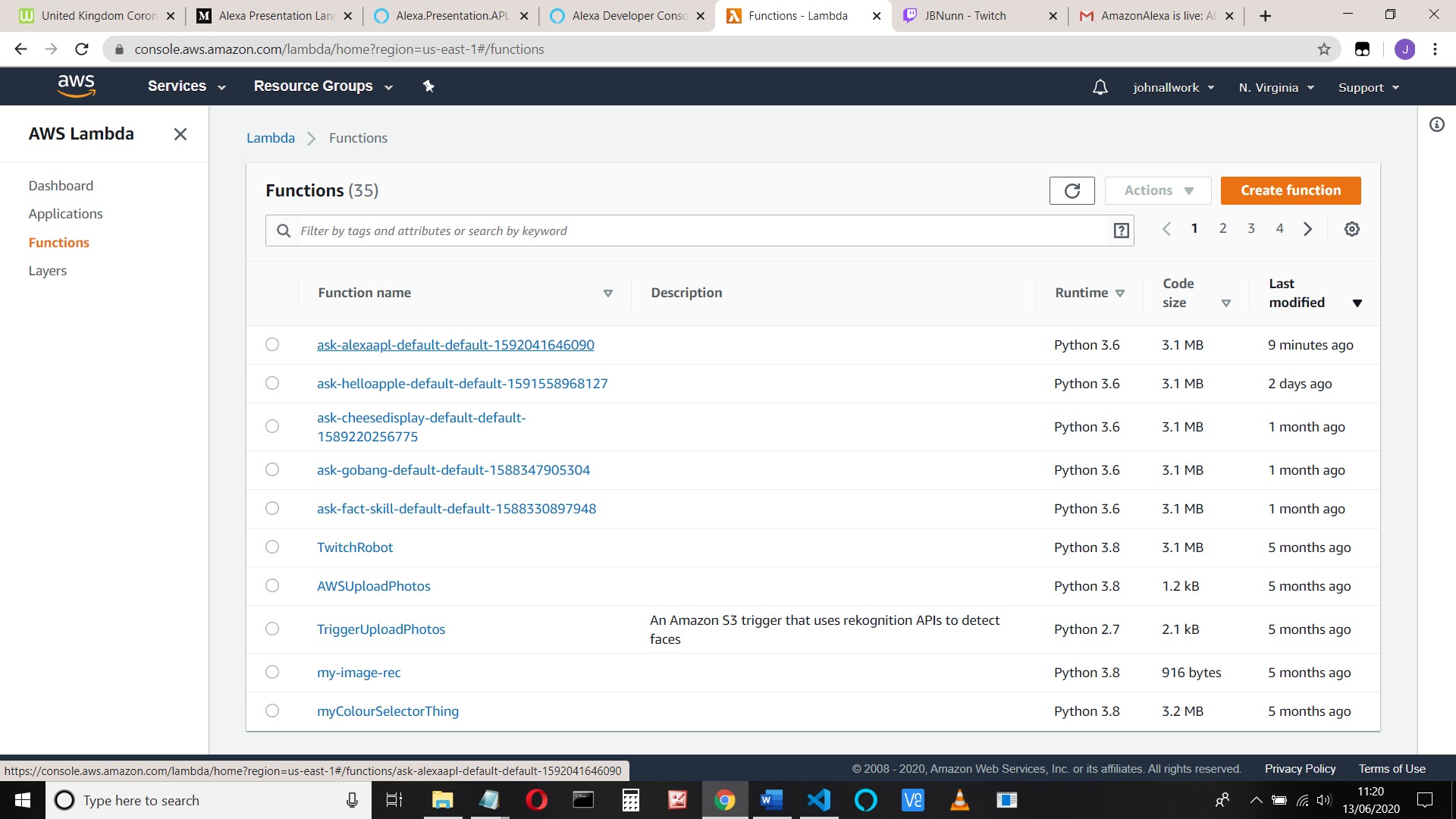


Also note that the Display Interface is NOT selected. Leave it like that. That too caused me a lot of problems.



It’s worth selecting Hub Landscape, Small. Then click **Save Interfaces**.

You can also open the aws lambda console and see that your code has been added:

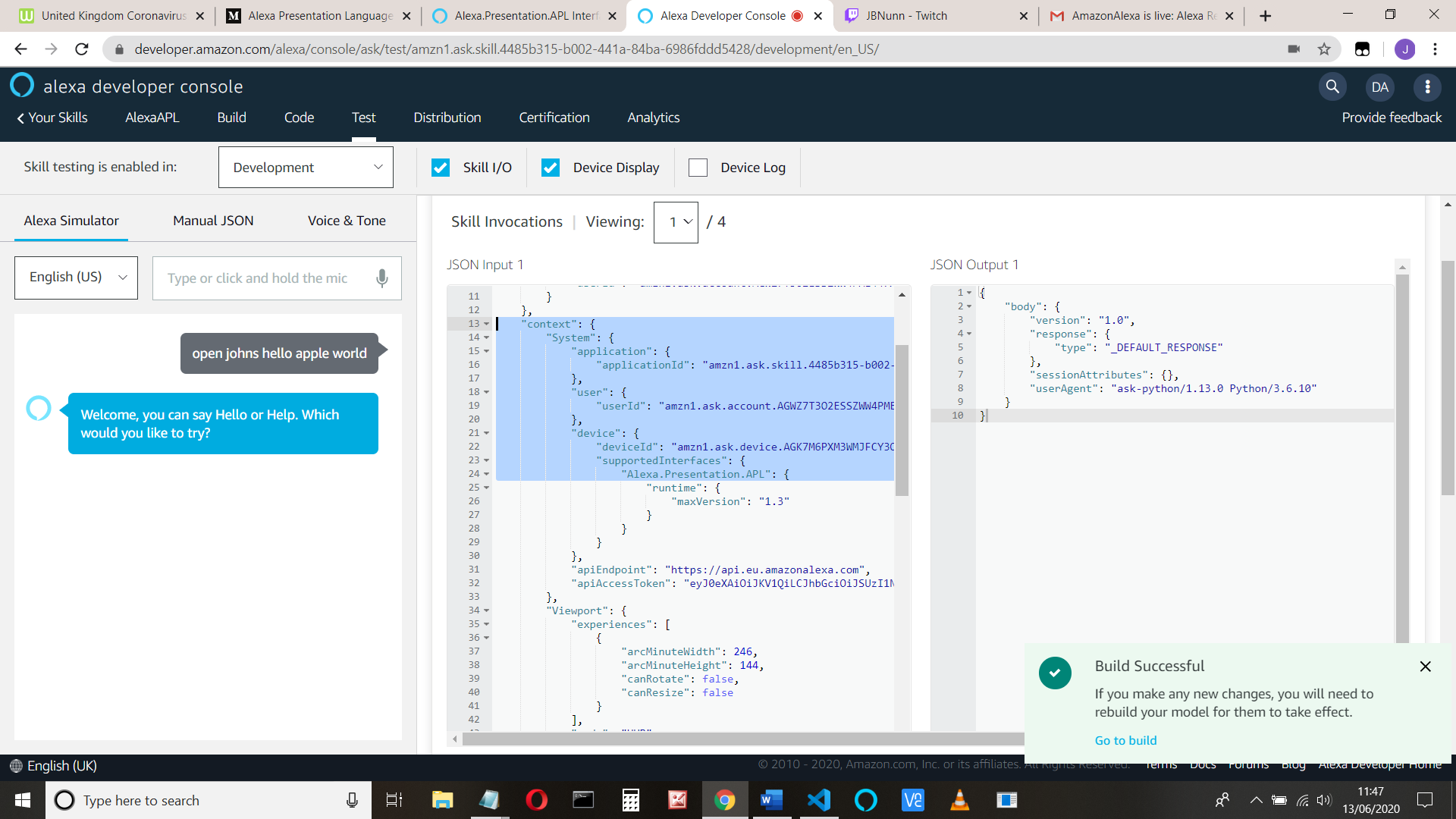


We can’t really do much here, but you can get to the CloudWatch if you are having errors.

**Adding APL code**

Now we know that our code works, let’s display something. You can build your display using the Display tab in the developer console, as Stuart Pocklington does, but that produced some complicated APL code, so I want to start simple.

Your code knows that APL is supported by the information sent to it in the json file, in context > System > device > SupportedInterfaces > AlexaPresentationAPL



You must check for this in your code, as you can’t send APL to a device that doesn’t have a screen

It also knows the type of device in the viewportProfile , so you can tailor your output to the device type, e,g using

if viewportProfile == @hubRoundSmall

You can see this in the SP example

I was really confused as to where all the APL code goes. Well it goes in the **directives** part of your json response, under a Alexa.Presentation.APL.RenderDocument directive.

Here’s a full response example that we will work to:

{

"body": {

"version": "1.0",

"sessionAttributes": {},

"userAgent": "ask-python/1.13.0 Python/3.6.10",

"response": {

"outputSpeech": {

"type": "SSML",

"ssml": "<speak>Hello Python World from Classes!</speak>"

},

"reprompt": {

"outputSpeech": {

"type": "SSML",

"ssml": "<speak>A reprompt to keep the session open</speak>"

}

},

"directives": [

{

"type": "Alexa.Presentation.APL.RenderDocument",

"token": "pagerToken",

"document": {

"type": "APL",

"version": "1.3",

"mainTemplate": {

"item": {

"type": "Text",

"text": "Hello, world"

}

}

},

"datasources": {}

}

],

"shouldEndSession": false

}

}

}

Most of this is written for you.

We need to add the "Alexa.Presentation.APL.RenderDocument" part. This is defined at:

<https://developer.amazon.com/en-US/docs/alexa/alexa-presentation-language/apl-interface.html#renderdocument-directive>

We need to generate the APL and add this as a RenderDocument directive:

Note the RenderDocument directive must have a ‘**type’**, a ‘**token’** and a ‘**document’** (of type APL document, see: <https://developer.amazon.com/en-US/docs/alexa/alexa-presentation-language/apl-document.html>)

Here’s the simplest response APL **document** (see where this is in the above json example):

{

"type": "APL",

"version": "1.3",

"mainTemplate": {

"item": {

"type": "Text",

"text": "Hello, world"

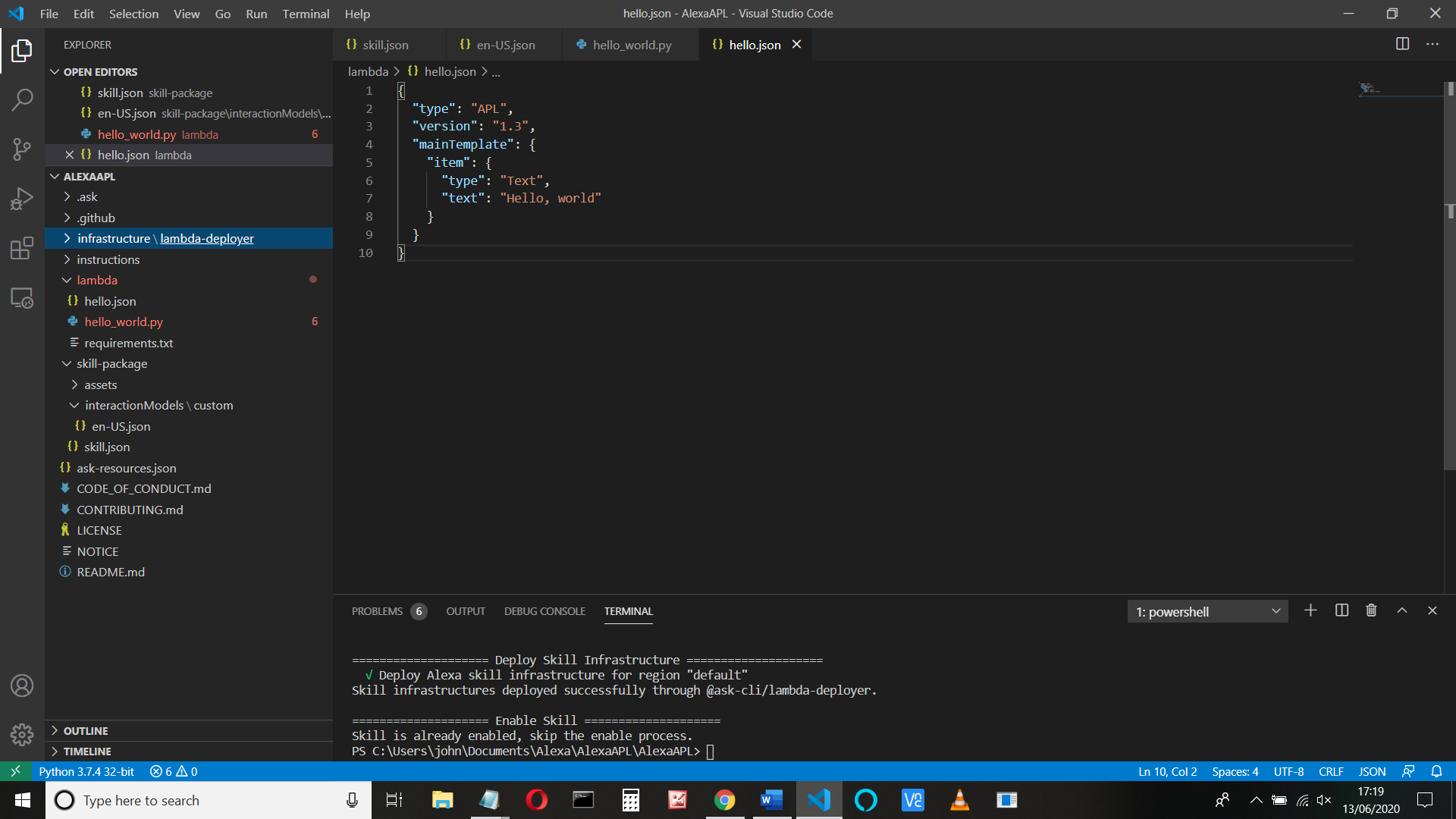
}

}

}

We need to add a RenderDocument directive with a ‘type’, a ‘token’ and an APL ‘document’ to our response. Fortunately, there is an **add\_directive** command we can use to do this. We need to add this command to our code though.

Firstly, in VS code, create a new file, paste the above code and save it as hello.json in the same folder and your hello.py code



Now return to the hello.py code.

Remember that we must only send our display information to a device with a display, so we must check for this.

The check for a display is provided in ask\_sdk\_core.utils. We need to add the following at the top of our code (put it somewhere near the other import code)

**from ask\_sdk\_core.utils import get\_supported\_interfaces**

(Thanks to Jeff Nunn at Amazon for this)

In our code we just say

*if get\_supported\_interfaces(handler\_input).alexa\_presentation\_apl is not None:*

*# we have a screen*

We also need to read our hello.json file we’ve just created. We’ll use “ \_load\_apl\_document “

Add the following at the top of your code (say before *class LaunchRequestHandler*)

**def \_load\_apl\_document(file\_path):**

**# type: (str) -> Dict[str, Any]**

**"""Load the apl json document at the path into a dict object."""**

**with open(file\_path) as f:**

**return json.load(f)**

To support this, we need to *import json*.

Add **import json** at the top of your code (don’t worry I’ll show this to you in a minute and provide the source)

To use the add\_directive, we need to import some more code, so add the following, under your import json

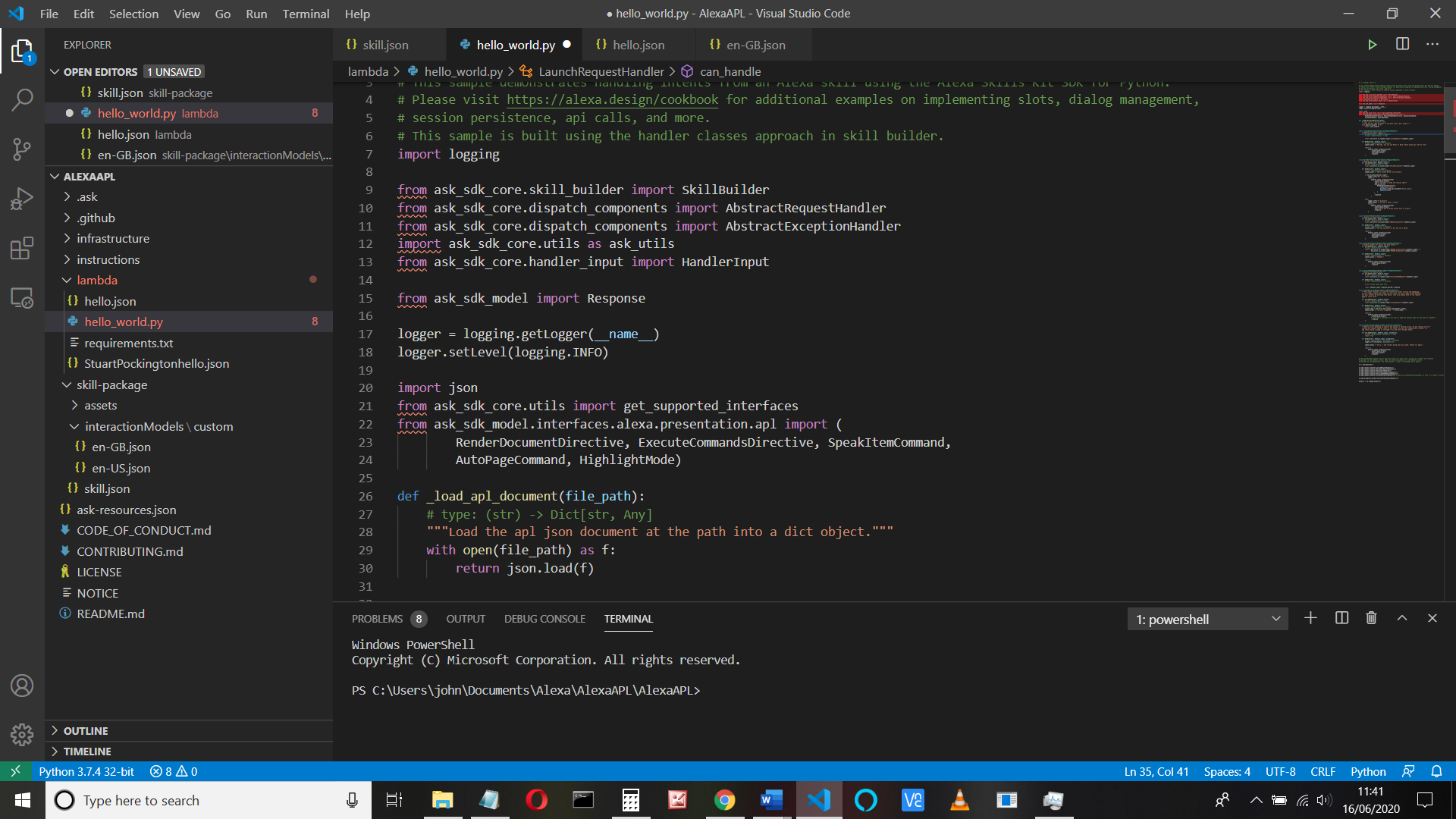
**from ask\_sdk\_model.interfaces.alexa.presentation.apl import (**

**RenderDocumentDirective, ExecuteCommandsDirective, SpeakItemCommand,**

**AutoPageCommand, HighlightMode)**

We only need the RenderDocumentDirective, but the others come free!

The top of your code should look like this:



Now find the HelloIntent code. Change the code after speak\_output = "Hello Python World from Classes!" so that it checks for a display and adds the directive if there is one, as follows:

**if get\_supported\_interfaces(handler\_input).alexa\_presentation\_apl is not None:**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**.ask("A reprompt to keep the session open")**

**.add\_directive(**

**RenderDocumentDirective(**

**token="pagerToken",**

**document=\_load\_apl\_document("hello.json"),**

**datasources={}**

**)**

**)**

**.response**

**)**

**else:**

**speak\_output = "You don't have a screen"**

**return (**

**handler\_input.response\_builder**

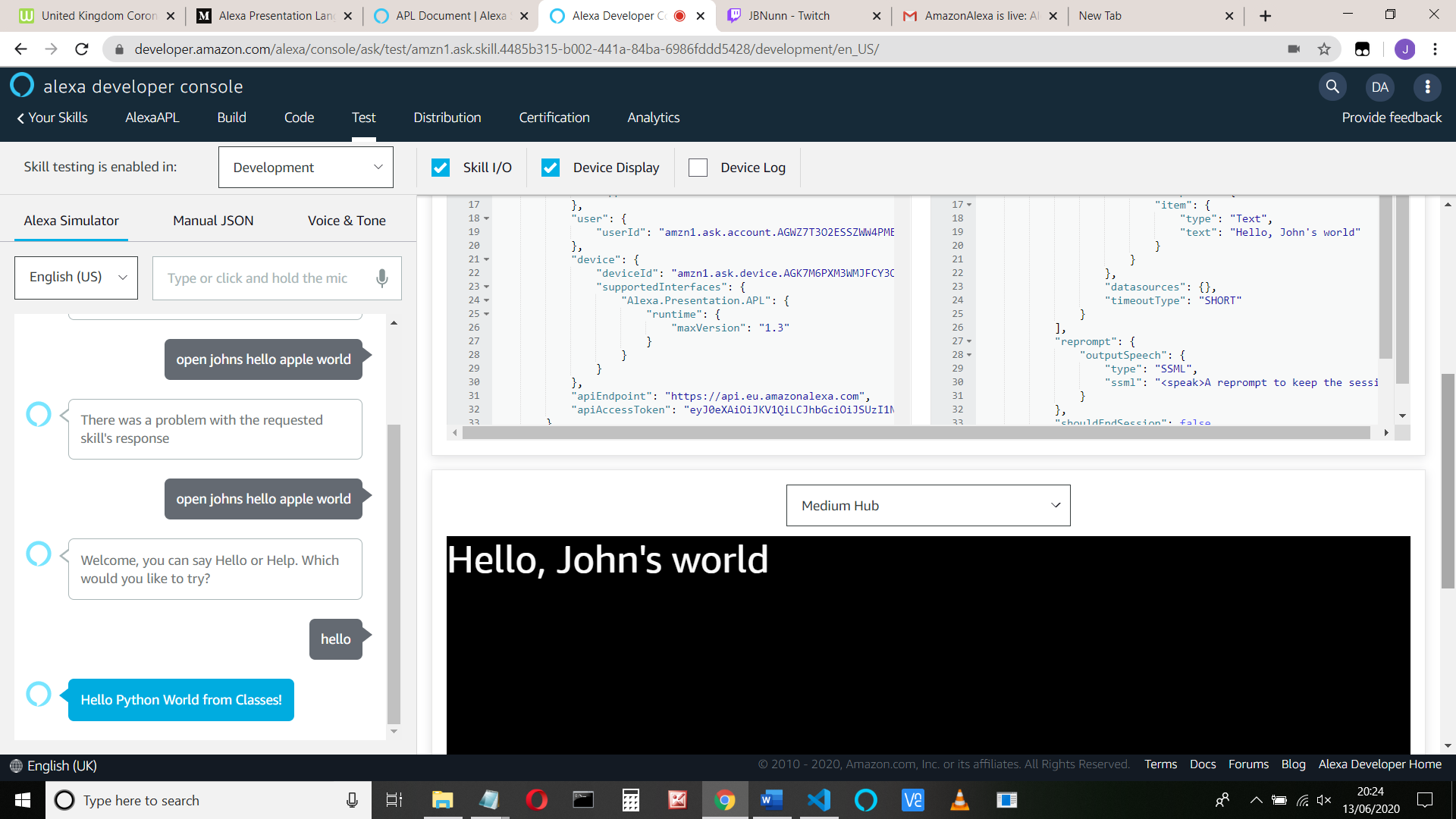
**.speak(speak\_output)**

**.ask("Please buy an Alexa device with a screen")**

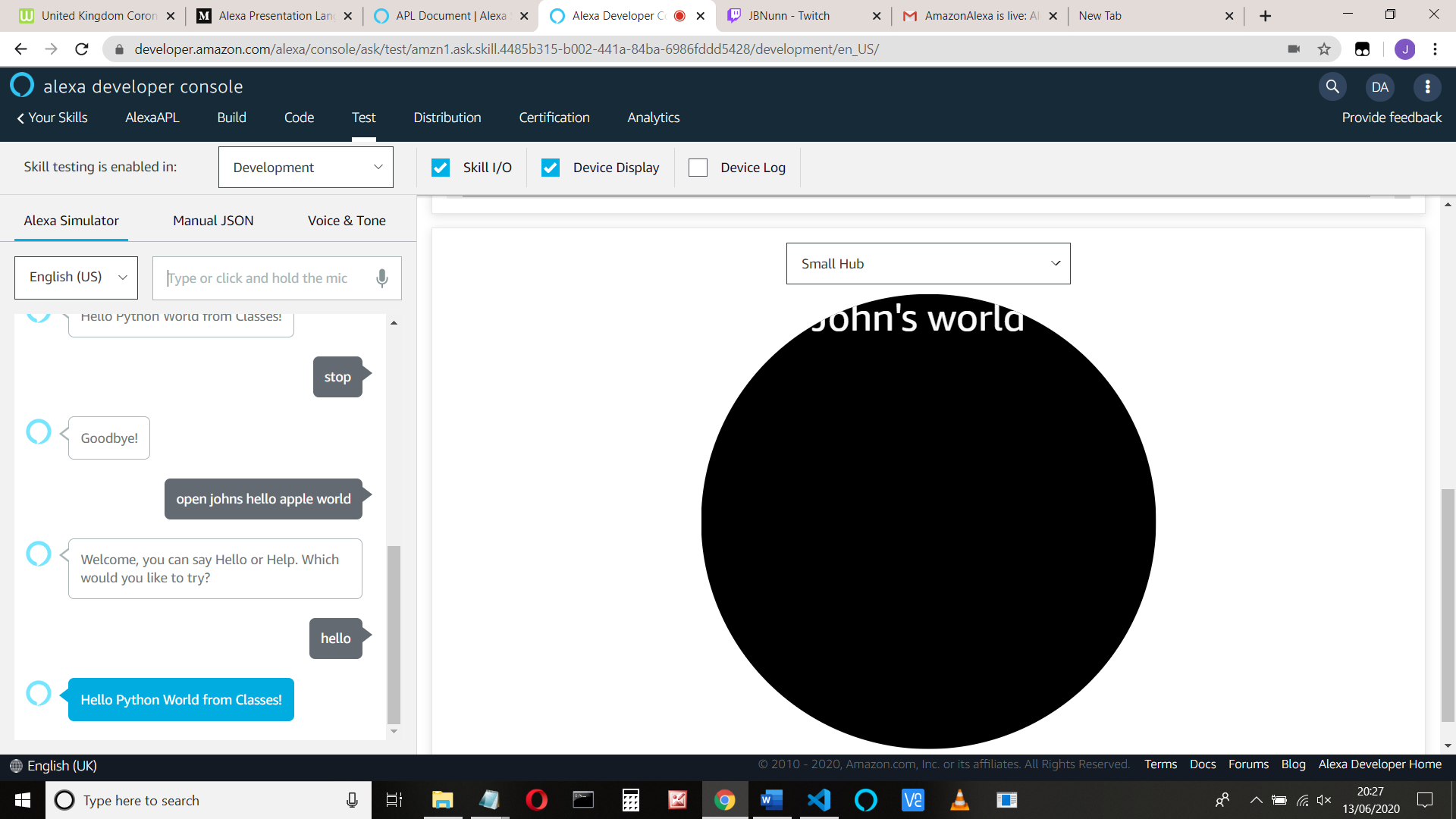
**.response**

**)**

Save and deploy your code. Type **ask deploy** in the terminal and test it out in the developer console. Invoke your skill and you should see the code displayed when you say (or type) ‘Hello’ (scroll down if necessary)



You can try out different screen sizes, try the small hub:



You can also try it in the CLI by typing **ask dialog**. Invoke your skill and type hello:



Now you know how to add APL json to your code, and if you didn’t before, take a look at the display templates and create your own displays at:

<https://developer.amazon.com/alexa/console/ask/displays?#/templates>

If you want a more complicated example, read the rest of Stuart Pockington’s blog.

You can copy, cut and paste his APL code into your hello.json and run it. (But I think it needs an extra closing brace ‘}’ )

That’s it, best of luck.

Here’ my full hello\_world.py:

**# -\*- coding: utf-8 -\*-**

**# This sample demonstrates handling intents from an Alexa skill using the Alexa Skills Kit SDK for Python.**

**# Please visit https://alexa.design/cookbook for additional examples on implementing slots, dialog management,**

**# session persistence, api calls, and more.**

**# This sample is built using the handler classes approach in skill builder.**

**import logging**

**from ask\_sdk\_core.skill\_builder import SkillBuilder**

**from ask\_sdk\_core.dispatch\_components import AbstractRequestHandler**

**from ask\_sdk\_core.dispatch\_components import AbstractExceptionHandler**

**import ask\_sdk\_core.utils as ask\_utils**

**from ask\_sdk\_core.handler\_input import HandlerInput**

**from ask\_sdk\_model import Response**

**logger = logging.getLogger(\_\_name\_\_)**

**logger.setLevel(logging.INFO)**

**import json**

**from ask\_sdk\_core.utils import get\_supported\_interfaces**

**from ask\_sdk\_model.interfaces.alexa.presentation.apl import (**

**RenderDocumentDirective, ExecuteCommandsDirective, SpeakItemCommand,**

**AutoPageCommand, HighlightMode)**

**def \_load\_apl\_document(file\_path):**

**# type: (str) -> Dict[str, Any]**

**"""Load the apl json document at the path into a dict object."""**

**with open(file\_path) as f:**

**return json.load(f)**

**class LaunchRequestHandler(AbstractRequestHandler):**

**"""Handler for Skill Launch."""**

**def can\_handle(self, handler\_input):**

**# type: (HandlerInput) -> bool**

**return ask\_utils.is\_request\_type("LaunchRequest")(handler\_input)**

**def handle(self, handler\_input):**

**# type: (HandlerInput) -> Response**

**speak\_output = "Welcome, you can say Hello or Help. Which would you like to try?"**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**.ask(speak\_output)**

**.response**

**)**

**class HelloWorldIntentHandler(AbstractRequestHandler):**

**"""Handler for Hello World Intent."""**

**def can\_handle(self, handler\_input):**

**# type: (HandlerInput) -> bool**

**return ask\_utils.is\_intent\_name("HelloWorldIntent")(handler\_input)**

**def handle(self, handler\_input):**

**# type: (HandlerInput) -> Response**

**speak\_output = "Hello Python World from Classes!"**

**if get\_supported\_interfaces(handler\_input).alexa\_presentation\_apl is not None:**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**.ask("A reprompt to keep the session open")**

**.add\_directive(**

**RenderDocumentDirective(**

**token="pagerToken",**

**document=\_load\_apl\_document("hello.json"),**

**datasources={}**

**)**

**)**

**.response**

**)**

**else:**

**speak\_output = "You don't have a screen"**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**.ask("Please buy an Alexa device with a screen")**

**.response**

**)**

**class HelpIntentHandler(AbstractRequestHandler):**

**"""Handler for Help Intent."""**

**def can\_handle(self, handler\_input):**

**# type: (HandlerInput) -> bool**

**return ask\_utils.is\_intent\_name("AMAZON.HelpIntent")(handler\_input)**

**def handle(self, handler\_input):**

**# type: (HandlerInput) -> Response**

**speak\_output = "You can say hello to me! How can I help?"**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**.ask(speak\_output)**

**.response**

**)**

**class CancelOrStopIntentHandler(AbstractRequestHandler):**

**"""Single handler for Cancel and Stop Intent."""**

**def can\_handle(self, handler\_input):**

**# type: (HandlerInput) -> bool**

**return (ask\_utils.is\_intent\_name("AMAZON.CancelIntent")(handler\_input) or**

**ask\_utils.is\_intent\_name("AMAZON.StopIntent")(handler\_input))**

**def handle(self, handler\_input):**

**# type: (HandlerInput) -> Response**

**speak\_output = "Goodbye!"**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**.response**

**)**

**class SessionEndedRequestHandler(AbstractRequestHandler):**

**"""Handler for Session End."""**

**def can\_handle(self, handler\_input):**

**# type: (HandlerInput) -> bool**

**return ask\_utils.is\_request\_type("SessionEndedRequest")(handler\_input)**

**def handle(self, handler\_input):**

**# type: (HandlerInput) -> Response**

**# Any cleanup logic goes here.**

**return handler\_input.response\_builder.response**

**class IntentReflectorHandler(AbstractRequestHandler):**

**"""The intent reflector is used for interaction model testing and debugging.**

**It will simply repeat the intent the user said. You can create custom handlers**

**for your intents by defining them above, then also adding them to the request**

**handler chain below.**

**"""**

**def can\_handle(self, handler\_input):**

**# type: (HandlerInput) -> bool**

**return ask\_utils.is\_request\_type("IntentRequest")(handler\_input)**

**def handle(self, handler\_input):**

**# type: (HandlerInput) -> Response**

**intent\_name = ask\_utils.get\_intent\_name(handler\_input)**

**speak\_output = "You just triggered " + intent\_name + "."**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**# .ask("add a reprompt if you want to keep the session open for the user to respond")**

**.response**

**)**

**class CatchAllExceptionHandler(AbstractExceptionHandler):**

**"""Generic error handling to capture any syntax or routing errors. If you receive an error**

**stating the request handler chain is not found, you have not implemented a handler for**

**the intent being invoked or included it in the skill builder below.**

**"""**

**def can\_handle(self, handler\_input, exception):**

**# type: (HandlerInput, Exception) -> bool**

**return True**

**def handle(self, handler\_input, exception):**

**# type: (HandlerInput, Exception) -> Response**

**logger.error(exception, exc\_info=True)**

**speak\_output = "Sorry, I had trouble doing what you asked. Please try again."**

**return (**

**handler\_input.response\_builder**

**.speak(speak\_output)**

**.ask(speak\_output)**

**.response**

**)**

**# The SkillBuilder object acts as the entry point for your skill, routing all request and response**

**# payloads to the handlers above. Make sure any new handlers or interceptors you've**

**# defined are included below. The order matters - they're processed top to bottom.**

**sb = SkillBuilder()**

**sb.add\_request\_handler(LaunchRequestHandler())**

**sb.add\_request\_handler(HelloWorldIntentHandler())**

**sb.add\_request\_handler(HelpIntentHandler())**

**sb.add\_request\_handler(CancelOrStopIntentHandler())**

**sb.add\_request\_handler(SessionEndedRequestHandler())**

**sb.add\_request\_handler(IntentReflectorHandler()) # make sure IntentReflectorHandler is last so it doesn't override your custom intent handlers**

**sb.add\_exception\_handler(CatchAllExceptionHandler())**

**handler = sb.lambda\_handler()**

and my hello.json

{

  "type": "APL",

  "version": "1.3",

  "mainTemplate": {

    "item": {

      "type": "Text",

      "text": "Hello, John's world"

    }

  }

}

And my en-US.json

**{**

**"interactionModel": {**

**"languageModel": {**

**"invocationName": "johns hello apple world",**

**"intents": [**

**{**

**"name": "AMAZON.CancelIntent",**

**"samples": []**

**},**

**{**

**"name": "AMAZON.HelpIntent",**

**"samples": []**

**},**

**{**

**"name": "AMAZON.StopIntent",**

**"samples": []**

**},**

**{**

**"name": "HelloWorldIntent",**

**"slots": [],**

**"samples": [**

**"hello",**

**"how are you",**

**"say hi world",**

**"say hi",**

**"hi",**

**"say hello world",**

**"say hello"**

**]**

**},**

**{**

**"name": "AMAZON.NavigateHomeIntent",**

**"samples": []**

**}**

**],**

**"types": []**

**}**

**}**

**}**

References:

Adding another language

I have a problem that I’m in the UK, where my devices are, but the skill is published for the US, so I want to add a UK version.

1. Copy the en-US.json and save it as en-GB.json. You can change the intent samples here (say to add ‘good morning’)
2. Edit the skill.json to add the en-GB locale so that it now starts:

**{**

**"manifest": {**

**"publishingInformation": {**

**"locales": {**

**"en-GB": {**

**"summary": "Sample Short Description",**

**"examplePhrases": [**

**"Alexa open hello world",**

**"hello",**

**"help"**

**],**

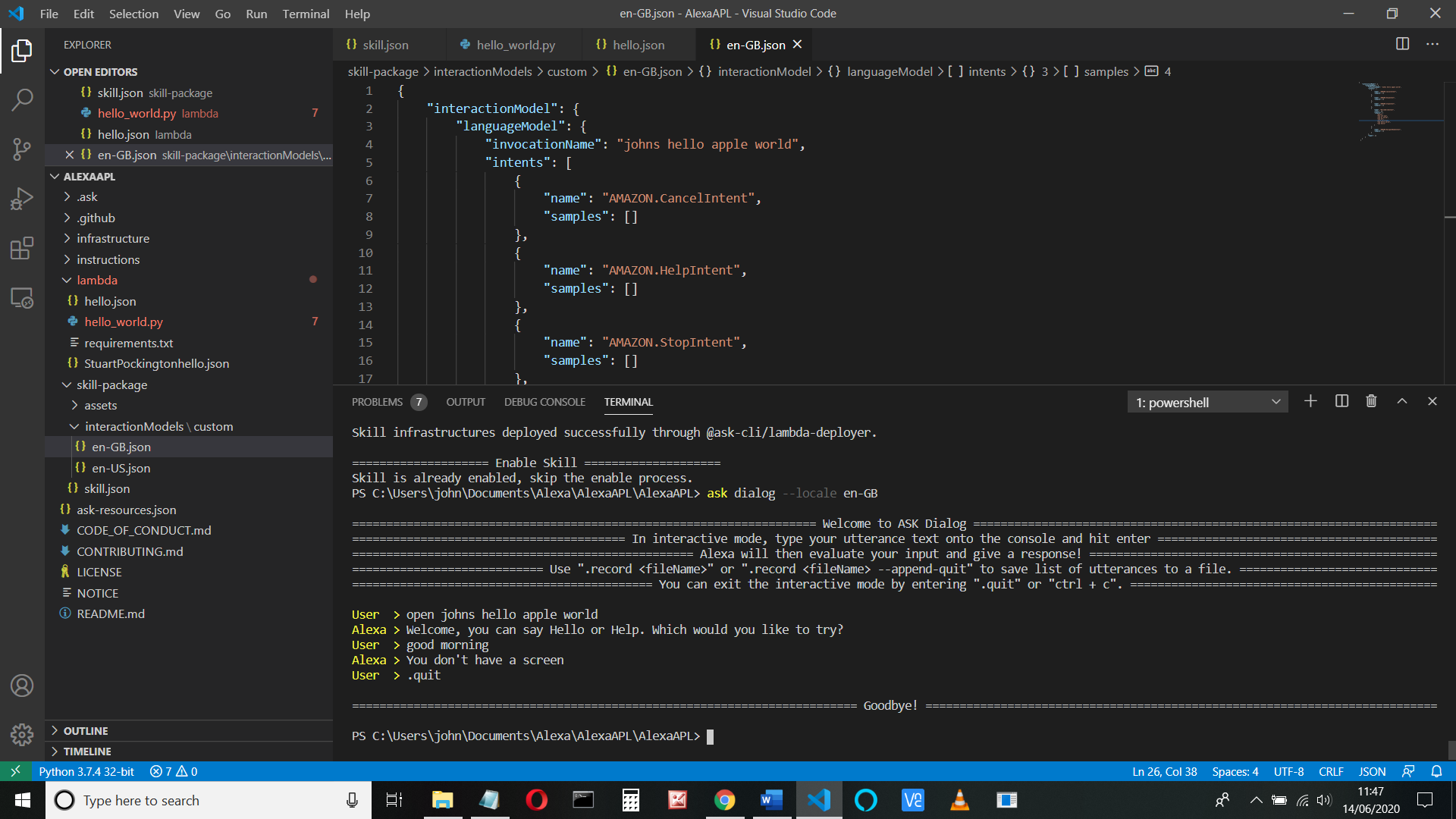
**"name": "AlexaAPL",**

**"description": "Sample Full Description"**

**},**

**"en-US": {**

Seems to work using **ask dialog –locale en-GB**



Other references:

Developing Your First Skill:

<https://developer.amazon.com/en-GB/docs/alexa/alexa-skills-kit-sdk-for-python/develop-your-first-skill.html>

Response Building:

<https://developer.amazon.com/en-GB/docs/alexa/alexa-skills-kit-sdk-for-python/build-responses.html>

A more complicated Alexa example python lambda function:

<https://github.com/alexa-labs/skill-sample-python-pager-karaoke/blob/master/lambda/py/lambda_function.py>