Developer Notes:

* **Alexa** is a cloud based service that handles all the speech recognition, machine learning, and Natural Learning Understanding for all Alexa enabled devices like Echo
* **Skills** are built in capabilities that define how you interact with the device ex: playing music, getting a weather forecast, etc.

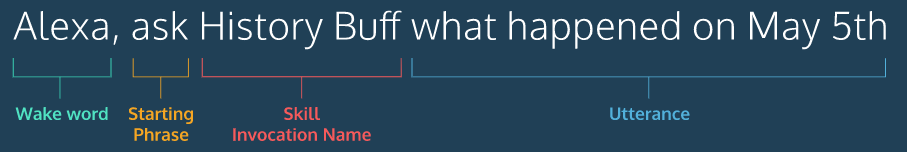
Skills Necessities:

* Must contain two parts:

1. **Interaction Model (Frontend)** is like the GUI of an app. Refers to the VUI (Voice User Interface) - It defines what functionalities or behaviors the skill is able to handle
2. **Hosted Service (Backend)** contains the programming logic, hosted on the internet, responds to users’ requests



How To interact with an Alexa Skill:



* **Wake word:** Wakes up the device and tells the device the user wants to talk to it
* **Starting Phrase:** Specifies the type of request they are using. **Examples:** Tell, Search, Ask, Open
* **Skill Invocation Name:** The specific skill you want to invoke
* **Utterance**

**Frontend**

Intents

* **Intents** are the behaviors your skill will have and it will execute some code based on it
* **Example:** Intent is called HelloIntent -- point of it is to respond and greet a user. Some utterances we would have with this intent are:
* How are you?
* Hello
* Howdy
* What’s up?

**Backend**

AWS Lambda Function

* A service-less computer service from AWS that runs your code when you trigger an event
* For our skill, we will trigger this function when you start the skill
* To connect the frontend with the backend, you must provide an **Amazon Resource Name** **(ARN)** which is a unique identifier that represents our Lambda function

Testing Our Skill

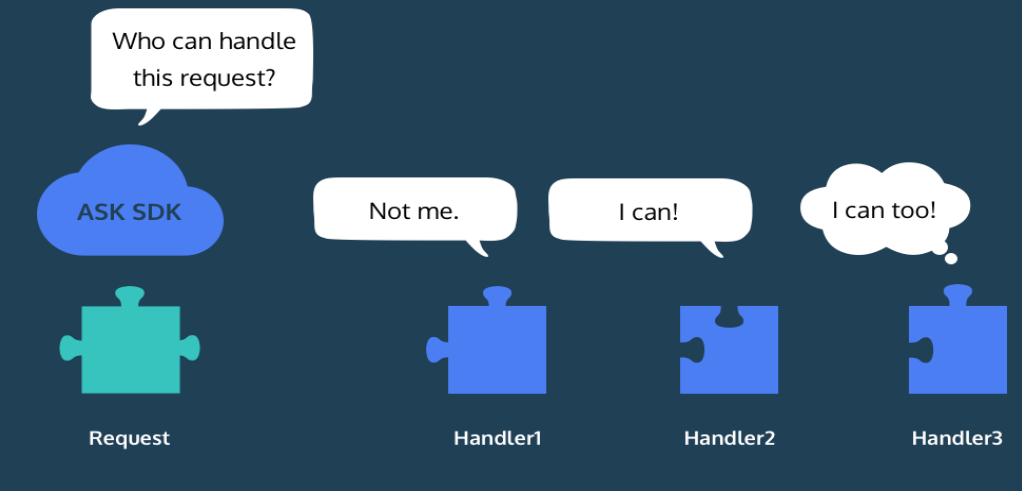
* You can test your new Alexa skill on this site where you can simulate an Amazon Alexa <https://echosim.io/>
* To test further, click on the test tab at the alexa developer console page. Here you must enable testing for the skill, and type in the skill you are testing as if you were saying it.
* **Example:** “Tell code academy hello” -- you must use the microphone to view the JSON I/O files
* It will process a JSON input file and JSON output file and as well give you the output expected or possibly an error message

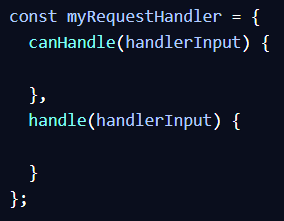
Creating Our Own Lambda Function

* In order to create our own Lambda function we must import the ASK SDK
* **In JavaScript: const Alexa = require(‘ask-sdk-core’);**
* Response Handlers
  + Our code will be organized into objects called response handlers
  + They:

1. Make our code easier to maintain
2. Allow us to define specific behavior to each situation

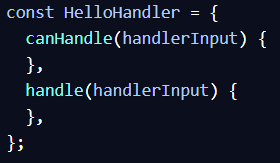
* When a request is received the SDK picks the first handler that can handle a request



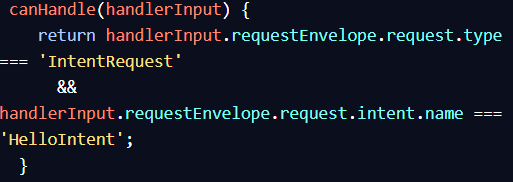
* We write request handlers like this:
* 

How to make a handler:

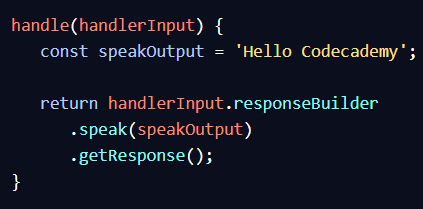
* You can name a handler anything, but for the example in codecademy they called the handler “HelloHandler”
* Below is the code to create a handler



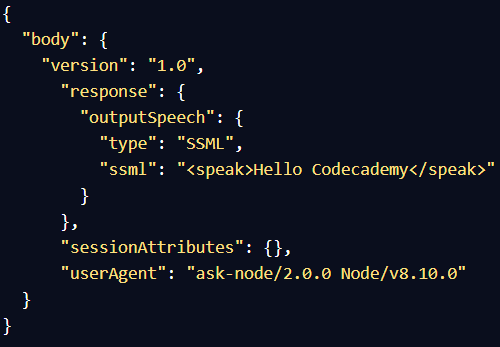
* The **canHandle** and **handle** are HelloHandler’s method’s with the argument handlerInput
* Within the **canHandle method**, or however many methods we require, you must put in a return statement that will tell the HelloHandler if this specific method can actually handle the request
* For example, within the **canHandle** method, we put….



* This will return **true** if the function can handle the handlerInput
* After we create this, we need to add the code that will formulate a JSON response -- this will be within the **handle()** method within the **HelloHandler** object
* Instead of creating a JSON file, we can use the **responseBuilder** which is inside of the **handlerInput** object



* **responseBuilder.speak()** specifies the text Alexa will say
* But the handle method must return a JSON object so we call **.getResponse()** to convert the object to JSON
* The **getResponse()** returns the following JSON object:



* **Ssml** is where the Alexa will actually speak

The following is how the overall code should look now:



Getting a response from our skill:

1. **Intent Request**

Ex: “Alexa, **tell** code academy hello”

* Alexa will recognize this as a specific intent to get from a skill and using the sample utterances that we got from the interaction model, maps it to **HelloIntent**

1. **LaunchRequest**

Ex: “Alexa, **open** code academy”

* To provide a response for this type of request, we need to add another handler called a **LaunchRequestHandler**
* Its **canHandle** method will return true if the request is a **LaunchRequest**

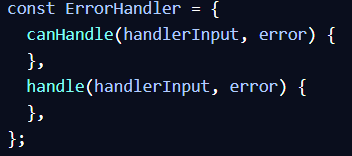
How the code should look now with an LaunchRequest handler



* Then you add a **handle** method very similar to the one in the **HelloHandler** handler that can handle such a request
* Has the same format, just a different speak output

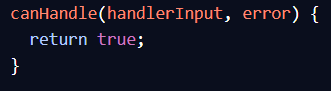
Error Handlers

* Similar to request handlers, except they are used to handle errors
* What if the user doesn’t use either the IntentRequest or the LaunchRequest, we need to handle these errors
* How to setup an ErrorHandler:

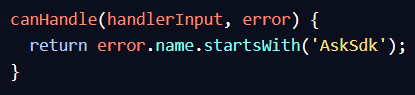


* Instead of **canHandle** and **handle** only having 1 parameter, handlerInput, it also includes an **error** parameter

Creating a canHandle that can catch errors:

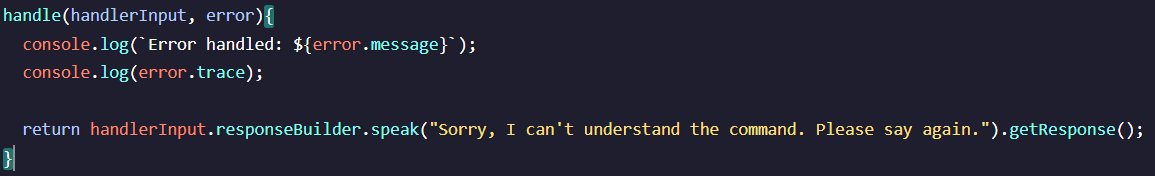


* This simply returns true if there is an error, and catches all errors
* If you want a **canHandle** that can catch specific errors:



Handle method for an ErrorHandler

* The handle method should output that Alexa “could not handle that request”
* Ex:

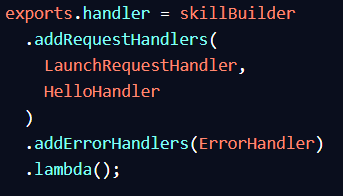


Register Handlers and Execute Lambda Function

* How does AWS Lambda know how to execute the handlers in the order we choose, and how does it call them
* We have to create a function that will access the handlers in an order we choose
* **How the function works:**
  + whenever a request is sent by Alexa AWS Lambda will call that function, pass it the JSON request, and the function will process that request, return a JSON response, and AWS Lambda sends it back to Alexa
* We don't have to build this function from scratch, we can use **Alexa.SkillBuilders** and use the **custom** version to handle simple skills



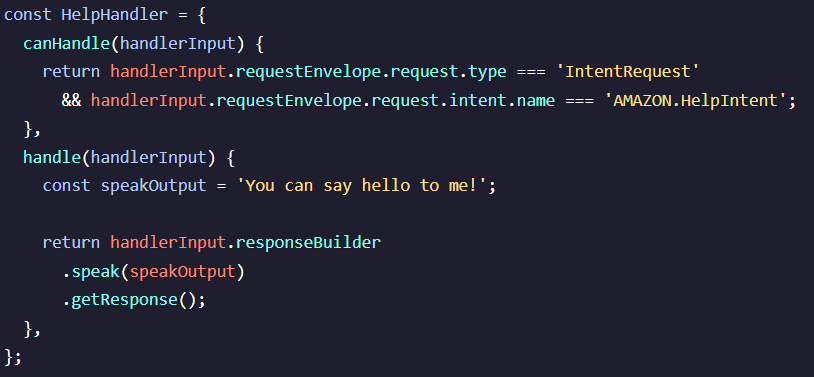
* To define the order we want as to how we want the handlers to be executed, we need to register the handlers to that function using the SkillBuilder’s **addRequestHandlers** and **addErrorHandlers** methods



* In the above code, **LaunchRequestHandler** would execute first and then **HelloHandler**, we then add the errorHandler **ErrorHandler** which would execute last
* The **lambda()** method at the end returns a function, and it is exported through the **exports.handler**

Other Handlers Required for any Basic skill

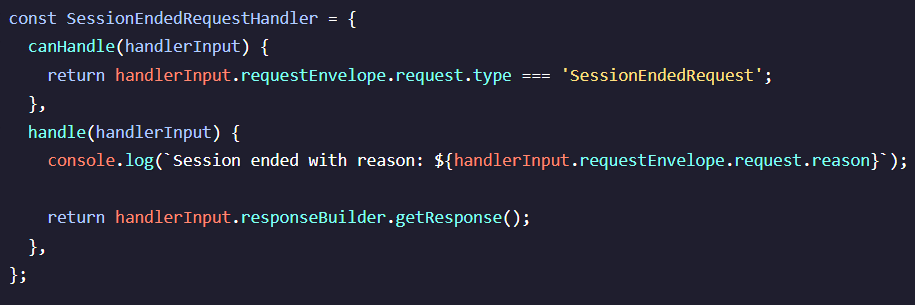
**HelpIntentHandler**



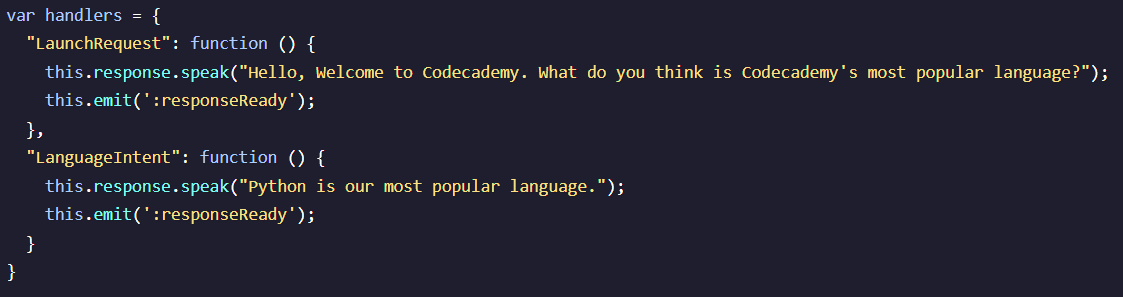
**CancelAndStopIntentHandler**

****

**SessionEndedRequestHandler**



Generating a Conversation With Alexa



* We create a variable that contains handlers which is a dictionary object
* Each key in the dictionary is a specific intent, so **LaunchRequest** is a specific intent
* The line that contains **this.response.speak()** contains text that Alexa would say out loud, it only generates a response and then ends that session
* **this.emit**() sends the response to user’s Alexa’s device
* To make the session not close, at the end of the **this.response.speak()** method line, we would add the method **.listen()** so the session continues
* Alexa is programmed to time out if a response isn’t given or she doesn’t understand the user’s question. To have her repeat a question add text within the **.listen(**“Text”**)**; to repeat the question