Crear y mantener una skill con múltiples idiomas

Iremos a build/lenguaje settings y añadiremos uno nuevo. Seleccionamos el que queramos y le damos al botón añadir. Esto nos habilitará un frontend en ese idioma, pero no un backend. Para ello necesitaremos copiar el json que teníamos predefinido y lo adaptaremos a este nuevo lenguaje.

No olvidar dar a Save/model con los cambios.

Si probásemos la aplicación en este punto, veremos que las respuestas se harán con una entonación en español, pero que seguirán con el texto inglés original. Encontraremos en la pestaña code los strings con esta info.

Deberemos detectar el locale (idioma por defecto del dispositivo) y cambiar entre un idioma u otro dependiendo de este. Utilizaremos para ello la librería **i18next**. Como es externa a Alexa, tendremos que traer esa dependencia en package.json

Cuando hagamos el Deploy, traerá las dependencias

Tras importarlas, haremos una estructura para los strings en el index con esta forma:

```
const i18n = require('i18next');
const sprintf = require('i18next-sprintf-postprocessor');

const languajeStrings={
    en:{
        translation:{
            WELCOME_MESSAGE: 'Welcome, you can say Hello or Help. Which would you like to try?'
        }
    },
    es:{
        translation:{
            WELCOME_MESSAGE: 'Bienvenido, puedes decir hola o ayuda. ¿Cual quieres?'
        }
    }
}
```

Utilizaremos **interceptors**, un código intermedio que se ejecuta entre el request y el response donde veremos el locale antes de mandar la respuesta. Dentro del método para interceptar, destaca la creación de un objeto de la librería i18next, con los siguientes atributos:

```
const localizationClient = i18n.use(sprintf).init({
    lng : handlerInput.requestEnvelope.request.locale,
    fallbacking : 'en',
    overloadTranslationOptionHandler : sprintf.overloadTranslationOptionHandler,
    resources: languageStrings,
    returnObjects:true
})
```

Donde “lng” será el lenguaje que queremos. En este caso, el locale. Tendremos un lenguaje por defecto en caso de que no se pueda detectar o que ocurra cualquier error, en este ejemplo, el de inglés. La herramienta que hará el paso de un string a otro, sprintf, y dónde almacenamos esa información con los String, en este ejemplo, sería en una variable llamada languageStrings.

Después obtendremos las características de la petición de entrada con un método propio de la herramienta de código de Alexa: attributesManager

```
const attributes = handlerInput.attributesManager.getRequestAttributes();
```

Y definimos una función dentro de este “attributes” que retornará el String específico a utilizar:

```
attributes.t = function (...args){
    return localizationClient.t(...args)
```

Una vez tengamos este método y lo hayamos incluido en los handlers a exportar, deberemos utilizarlo en los intent que vayamos a internacionalizar. Específicamente en su parte “handle”

Eliminaremos la línea marcada en gris y añadiremos las otras dos, que utilizan la clave “WELCOME_MESSAGE” para llamar al texto que corresponda:

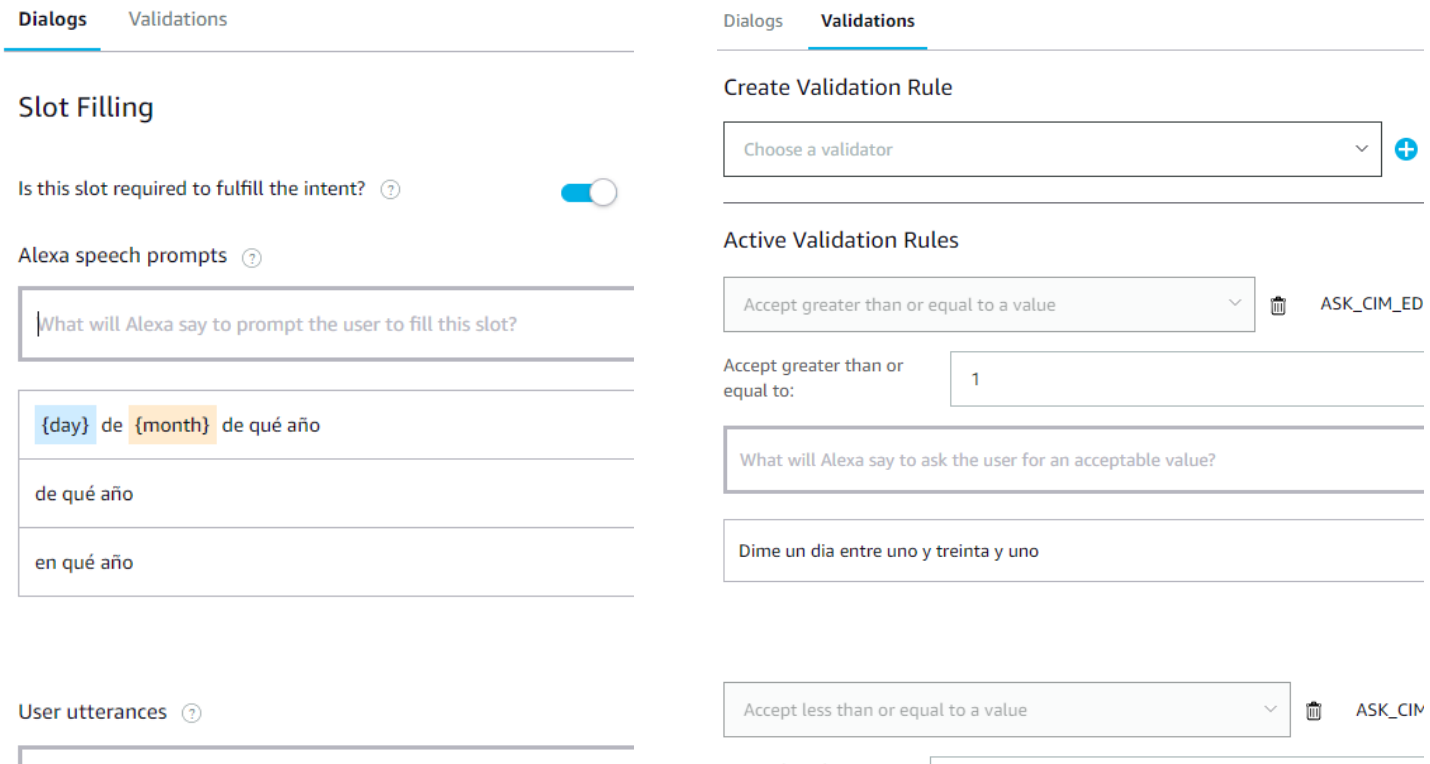
```
const requestAttributes = handlerInput.attributesManager.getRequestAttributes();
const speakOutput = 'Welcome, you can say Hello or Help. Which would you like to try?';
const speakOutput = requestAttributes.t('WELCOME_MESSAGE')
```

Manejo de diálogos

Esto nos permitirá mantener una conversación más natural con el usuario. Podemos definir estos slots debajo de las utterances. Al definirlo, tenemos varias opciones por defecto: number, month…

Pero si queremos algo que se adapte mejor, también podemos crear nuestros propios tipos. Podemos además marcar los slots como obligatorios. Por ejemplo, para saber el cumpleaños necesitaremos saber siempre el día. Por tanto, tendremos que escribir qué es lo que dirá Alexa si falta este dato y, por otro lado, manejar las posibles respuestas del usuario a esta petición de datos.

Además, podemos añadir validaciones a estos datos. Por ejemplo, que los números sean mayores a 1 en los días…



Toda esta información aparecerá reflejada en el JSON. De igual modo, podremos editar, añadir o quitar la información directamente desde el JSON, sin necesidad de pasar por estas ventanas. Aunque siempre es recomendable hacer los primeros elementos por aquí para que se cree automáticamente la estructura.

SI queremos que Alexa nos diga datos ya almacenados, como los de los slots, tendremos que indicar en el string un %s, y a la hora de enviar la información lo haremos mandando además como parámetros todos esos datos a rellenar:

```
const day = intent.slots.day.value;
const month = intent.slots.month.resolutions.resolutionsPerAuthority[0].values[0].value.name;
const year = intent.slots.year.value;

const speakOutput = requestAttributes.t('REGISTER_MSG', day, month, year);
```

Persistencia de datos

Para poder almacenar los datos y que no se pierdan durante la sesión, necesitaremos un mecanismo de persistencia. Para ello tendremos que importar varias librerías, que al igual que antes, indicaremos en el package.json:

```
"ask-sdk-s3-persistence-adapter": "^2.3.0",
"ask-sdk-dynamodb-persistence-adapter": "^2.3.0",
```

Añadiremos un método donde comprobaremos si estamos utilizando el sistema de Alexa Hosted o no, y dependiendo de ello, utilizar una persistencia u otra:

```
let persistenceAdapter = getPersistenceAdapter();

function getPersistenceAdapter() {
    function isAlexaHosted() {
        return process.env.S3_PERSISTENCE_BUCKET ? true : false;
    }

    const tableName = 'happy_birthday_table';
    if(isAlexaHosted()) {
        const {S3PersistenceAdapter} = require('ask-sdk-s3-persistence-adapter');
        return new S3PersistenceAdapter({
            bucketName: process.env.S3_PERSISTENCE_BUCKET
        });
    } else {
        // IMPORTANT: don't forget to give DynamoDB access to the role you're to run this lambda (IAM)
        const {DynamoDbPersistenceAdapter} = require('ask-sdk-dynamodb-persistence-adapter');
        return new DynamoDbPersistenceAdapter({
            tableName: tableName,
            createTable: true
        });
    }
}
```

Como se indica en los comentarios del código, AlexaHosted no necesita permisos, pero DynamoDB sí, por lo que deberemos aceptarlos antes de usarlo. En cualquier caso, deberemos definir este adapter en los exports: `whithPersistenceAdapter(persistenceAdapter)`

Utilizaremos interceptors en los métodos donde hay info que queramos guardar. Esto tendrá una doble cara. Por un lado, cargaremos los datos almacenados con cada nueva sesión, y por otro, los almacenaremos para que persistan al cierre de una sesión. Esto se refleja en dos métodos:

```
const LoadAttributesRequestInterceptor = {
    async process(handlerInput) {
        if(handlerInput.requestEnvelope.session['new']){ //is this a new session?
            const {attributesManager} = handlerInput;
            const persistentAttributes = await attributesManager.getPersistentAttributes() || {};
            //copy persistent attribute to session attributes
            handlerInput.attributesManager.setSessionAttributes(persistentAttributes);
        }
    }
};
```

```
const SaveAttributesResponseInterceptor = {
    async process(handlerInput, response) {
        const {attributesManager} = handlerInput;
        const sessionAttributes = attributesManager.getSessionAttributes();
        const shouldEndSession = (typeof response.shouldEndSession === "undefined" ? true : response.shouldEndSession);//is this a session end?
        if(shouldEndSession || handlerInput.requestEnvelope.request.type === 'SessionEndedRequest') { // skill was stopped or timed out
            attributesManager.setPersistentAttributes(sessionAttributes);
            await attributesManager.savePersistentAttributes();
        }
    }
};
```

Para trabajar con persistencia se utilizan los id de los elementos, por eso tendremos que duplicar la información anterior, por un lado, para sacar el nombre, y por otro, para obtener el id. El sistema de almacenamiento es clave valor, por lo que encontraremos este tipo de asignaciones:

```
sessionAttributes['day'] = day;
sessionAttributes['month'] = month;
sessionAttributes['monthName'] = monthName;
sessionAttributes['year'] = year;
```

Si quisiéramos trabajar con la fecha actual, para saber cuántos días quedan para el cumpleaños, por ejemplo, utilizaremos la librería moment.js.

```
"moment-timezone": "^0.5.23"
```

```
const moment = require('moment-timezone');
```

Esto se manifestará en el nuevo intent que creamos para esta acción de calcular cuánto queda para el cumpleaños. Añadimos aquí más lógica. Dependiendo de si tenemos fecha o no, de si ya hemos cumplido o no… Utilizando los datos que teníamos en persistencia:

```
let speechText;
if(day && month && year){
    const timezone = 'Europe/Madrid'; // we'll change this later to retrieve the timezone from the device
    const today = moment().tz(timezone).startOf('day');
    const wasBorn = moment(`${month}/${day}/${year}`, "MM/DD/YYYY").tz(timezone).startOf('day');
    const nextBirthday = moment(`${month}/${day}/${today.year()}`, "MM/DD/YYYY").tz(timezone).startOf('day');
    if(today.isAfter(nextBirthday)){
        nextBirthday.add('years', 1);
    }
```

En este fragmento vemos por ejemplo cómo establecer el timezone y sacar el día actual. Con startOf(“day”) haremos referencia a las 00:00, ya que no nos interesa tanto la hora como el día, estableciendo todas las fechas con esta misma hora. Por otro lado, vemos la toma de decisiones dependiendo de si es hoy el cumpleaños, si queremos modificarlo porque no es correcto…

```
    const age = today.diff(wasBorn, 'years');
    const daysLeft = nextBirthday.startOf('day').diff(today, 'days'); // same days returns 0
    speechText = requestAttributes.t('SAY_MSG', daysLeft, age + 1);
    if(daysLeft === 0) {
        speechText = requestAttributes.t('GREET_MSG', age);
    }
    speechText += requestAttributes.t('OVERWRITE_MSG');
} else {
    speechText = requestAttributes.t('MISSING_MSG');
}
```

Acceso a APIs de ASK

En este punto vemos inicialmente un cambio respecto al código anterior. Como la complejidad va aumentando, se recomienda segmentar más el código y distribuir en más archivos la funcionalidad. En este caso, nos llevaremos lo relacionado con persistencia e interceptors a nuevos ficheros. Tendremos que ponerlos como module.exports{}. Ahora haremos require() a sus ficheros y haremos referencia a ellos cuando pongamos los handler al final del index.

Queremos ahora crear la funcionalidad de saludar por su nombre al usuario. Para ello cogeremos este dato de su dispositivo y añadiremos el siguiente código a la LaunchRequest:

```
if(!sessionAttributes['name']){
    // let's try to get the given name via the Customer Profile API
    // don't forget to enable this permission in your skill configuratiuon (Build tab -> Permissions)
    // or you'll get a SessionEnndedRequest with an ERROR of type INVALID_RESPONSE
    try {
        const {permissions} = requestEnvelope.context.System.user;
        if(!permissions)
            throw { statusCode: 401, message: 'No permissions available' }; // there are zero permissions, no point in intializing the API
        const upsServiceClient = serviceClientFactory.getUpsServiceClient();
        const profileName = await upsServiceClient.getProfileGivenName();
        if (profileName) { // the user might not have set the name
          //save to session and persisten attributes
          sessionAttributes['name'] = profileName;
        }

    } catch (error) {
        console.log(JSON.stringify(error));
        if (error.statusCode === 401 || error.statusCode === 403) {
            // the user needs to enable the permissions for given name, let's send a silent permissions card.
          handlerInput.responseBuilder.withAskForPermissionsConsentCard(GIVEN_NAME_PERMISSION);
        }
    }
```

Para esta acción necesitaremos permisos, que habilitaremos en Build/Models/Permissions y seleccionaremos “given name”. Como estamos utilizando una API Client de Alexa, deberemos especificarlo en los handler:

```
.withApiClient(new Alexa.DefaultApiClient())
```

Como vemos al final del catch, mandaremos una tarjeta al JSON de salida mostrando los permisos necesarios de no haberlos obtenido.

Por otro lado, obtendremos el timezone del sistema de forma similar, con una diferencia:

No es necesario obtener permisos del usuario para obtener la zona horaria del dispositivo por eso no enviamos ninguna tarjeta ni verificamos si se ha otorgado el acceso

En la pestaña de Test, no obtendremos timezone del sistema. Por ello, para probar la aplicación deberemos hacer una condición donde, de no obtenerse, pongamos algún valor por defecto. Esto lo eliminaremos en el programa final, ya que no debería ser necesario.

Anotación, aunque sean APIs internas, también requieren un tiempo de respuesta, por eso es necesario establecer las llamadas como await en métodos async.

Envío de recordatorios

En esta parte veremos **SSML** para mejorar la interacción con sonidos. Y por otro lado los **Speechcons**, que nos permiten dar entonaciones.

Para lo primero, podremos obtener muchos efectos de la **Alexa Sound Library.** Ahí, además de cada sonido tendremos la etiqueta necesaria para acceder a ella:

Weather/Wind	Wind Whistling Desert (1)	2.15s

Source code: <audio src="soundbank://soundlibrary/nature/amzn_sfx_wind_whistling_desert_01"/> Copy to clipboard

Podemos utilizar cualquier sonido al que podamos acceder por http en formato mp3. Aunque hay restricciones de encoding y duración máxima.

Definiremos dentro de nuestro localisation una variable con el sonido, al nivel del resto de mensajes de WELCOME, REGISTER…

```
POSITIVE_SOUND: `<audio src='soundbank://soundlibrary/ui/gameshow/amzn_ui_sfx_gameshow_positive_response_02'/>`,
```

Y si lo vamos a aplicar antes o después de cualquier mensaje, lo añadiremos así:

```
GREET_MSG: '$t(POSITIVE_SOUND) $t(GREETING_SPEECHCON) {{name}}. '
```

Del mismo modo, tenemos una librería de Speechcons con palabras o expresiones con una entonación especial. Por ejemplo, si quisiéramos la entonación especial de “magnífico” que hemos encontrado en esta librería, pondríamos:

```
<say-as interpret-as="interjection">magnífico</say-as>
```

Y como hemos visto en el GREET_MSG, se añade del mismo modo que un sonido. Siguiendo con nuestro ejemplo de felicitación, lo definiríamos como:

```
GREETING_SPEECHCON: `<say-as interpret-as="interjection">felicidades</say-as>`
```

Reminders

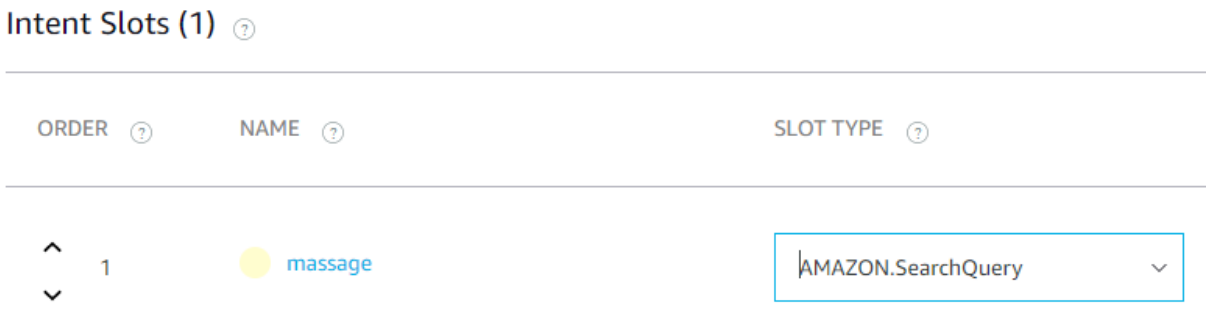
Tenemos dos tipos de notificaciones: los **reminders**, que son intrusivos. Se programan para un momento con un mensaje determinado, y estos saltarán cuando llegue el momento. Por otro lado, tenemos los **Proactive Events**. Que crearán una cola de notificaciones y esperará a que nosotros lancemos el intent de pedir que nos lea esas notificaciones para lanzarnos el mensaje. En un dispositivo Alexa, además, se indicará con una luz el momento en que tengamos notificaciones.

Para el ejemplo utilizaremos el primer modo, además, pediremos al usuario qué texto quiere que se diga cuando salte el recordatorio.

ANOTACIÓN: *Hasta ahora estamos utilizando Intents Custom, pero hay funcionalidades, como el repeat intent, donde podemos pedir que se nos repita el último mensaje que haya dicho Alexa, que*

están disponibles en la librería de Alexa. Al crearlo, seleccionaremos Use an existing intent, buscamos y añadimos, y después solo tendremos que incorporar un handler. Esto nos quita el trabajo de tener que pensar en todos los modos en que el usuario nos puede pedir que se repita, aunque aún así tendremos que crear la lógica de qué es lo que hacemos cuando llamamos a ese intent.

Siguiendo con los recordatorios, nos encontramos con la necesidad de registrar una frase que nos diga el usuario. No es algo tan puntual como un slot y puede ser muy variada, por lo que usaremos **Amazon Search Query**. Lo definiremos como un slot de este tipo:



Como es fundamental este mensaje para nuestro ejemplo y siempre tendremos que aportarlo, lo haremos obligatorio con **slot filling**, y, además, le añadiremos una **slot confirmation**, por si el usuario se ha confundido o no le convence.

Para los recordatorios, adicionalmente, deberemos habilitar su propio permiso en Models/Permission:

Necesitaremos el resto de datos para esta acción: day, mes, nombre… así que lo sacaremos de los atributos de sesión que ya hemos almacenado. Como curiosidad, ver el modo en que se puede coger el nombre o poner un texto vacío usando el operador ||

```
const day = sessionAttributes['day'];
const month = sessionAttributes['month'];
const year = sessionAttributes['year'];
const name = sessionAttributes['name'] || '';
let timezone = sessionAttributes['timezone'];
const message = Alexa.getSlotValue(requestEnvelope, 'message');
```

Guardamos el id en los atributos de sesión para luego poder borrar el recordatorio cuando creemos otro. Tenemos un *interceptor* que hará estos atributos persistentes aunque se cierre la sesión

```
sessionAttributes['reminderId'] = reminderResponse.alertToken;
```

APUNTE: Si tenemos un error de permiso lanzaremos la tarjeta. Esto se manifestará en el dispositivo del usuario pidiéndole ese permiso necesario para la aplicación. Lo vemos en nuestro código:

```
case 401: // the user has to enable the permissions for reminders, let's attach a permissions card
    handlerInput.responseBuilder.withAskForPermissionsConsentCard(constants.REMINDERS_PERMISSION);
    speechText = handlerInput.t('MISSING_PERMISSION_MSG');
    break;
```

Acceso a APIs externas

Esto permitirá que nuestra Alexa responda con información proveniente de terceros. En el tutorial utilizaremos **Axios**, como librería que nos permita trabajar con estas APIs, similar a Node-fetch pero que permite establecer **time out** en las llamadas por si alguna fallase.

Veremos aquí las **Progressive Response**, que son mensajes que dice Alexa o sonidos que emite mientras estamos esperando a que responda otro servicio.

Necesitaremos añadir varios métodos a nuestra clase logic. Por ejemplo, este para obtener la fecha de hoy (tener en cuenta que el índice de los meses empieza en cero, por eso le sumamos uno)

```
getAdjustedDate(timezone) {
    const today = moment().tz(timezone).startOf('day');

    return {
        day: today.date(),
        month: today.month() + 1
    }
},
```

Por otro lado, vemos cómo acceder a la aplicación externa, donde utilizaremos en el ejemplo wikidata, que para su consulta usa un lenguaje parecido a sql, “sparql”. Definiremos por tanto el endpoint y la Select:

```
const endpoint = 'https://query.wikidata.org/sparql';
// List of actors with pictures and date of birth for a given day and month
const sparqlQuery =
`SELECT DISTINCT ?human ?humanLabel ?picture ?date_of_birth ?place_of_birthLabel WHERE {
```

Por tanto, definiremos la url como:

```
const url = endpoint + '?query=' + encodeURIComponent(sparqlQuery);
```

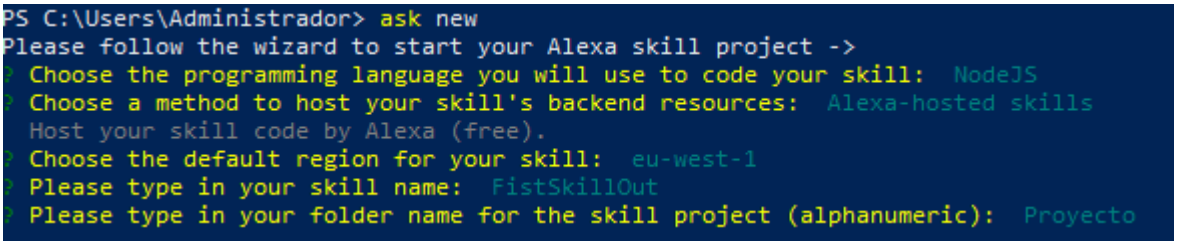
Por otro lado, indicamos en headers el tipo de consulta que hacemos, en este caso sparql, y el tipo de dato que nos tiene que devolver, json:

```
var config = {
    timeout: 6500, // timeout api call before we reach Alexa's 8 sec timeout
    headers: {'Accept': 'application/sparql-results+json'}
};
```

Si no ponemos que nos lo devuelva en json, por defecto lo traerá en xml.

ASK-CLI

Nos permite mediante línea de comandos crear, clonar, desplegar y testear skills. Esto nos permitirá editar el código con nuestro id preferido. Una vez ajustado podemos crear desde consola:



Si vamos a trabajar con la misma Skill desde un IDE y desde Alexa develop, podríamos desincronizar ambos proyectos. De fondo utiliza Git, por tanto, podremos utilizar los comandos de Git para subir el código o traerlo cuando queramos para trabajar a la par e ir actualizando.

Tenemos tres branch: **dev**, **maste**r y **prod**

> El primero es equivalente a cómo hemos estado trabajando hasta ahora en Alexa Develop cuando damos a **save**.
>
> El segundo sería el estado cuando hacemos **deploy.**
>
> La tercera rama se usará cuando nuestra skill esté publicada, en vivo, donde los usuarios la pueden usar. Es equivalente al botón de Alexa Develop **Promote to live.**