



Unión Europea

Fondo Social Europeo

"El FSE invierte en tu futuro"



Proyecto de desarrollo de aplicaciones multiplataforma JUEGO INTERACTIVO CON INTERFAZ CONVERSACIONAL

CICLO FORMATIVO DE GRADO SUPERIOR
Desarrollo de Aplicaciones Multiplataforma (IFCS02)

Curso 2021-22

Autor:
Mario González Gómez

Tutor:
José Luis González Sánchez

Departamento de Informática y Comunicaciones
I.E.S. Luis Vives

| | |
|---------------------------------|-----------|
| INTRODUCCIÓN | 3 |
| DESCRIPCIÓN | 3 |
| OBJETIVO | 3 |
| ALCANCE | 3 |
| JUSTIFICACIÓN | 4 |
| PLANIFICACIÓN Y ANÁLISIS | 5 |
| Planning | 5 |
| Análisis | 6 |
| IMPLEMENTACIÓN | 10 |
| DISEÑO | 10 |
| IMPLEMENTACIÓN | 14 |
| IMPLANTACIÓN | 25 |
| DOCUMENTACIÓN | 27 |
| TRABAJO FUTURO | 27 |
| CONCLUSIONES | 28 |
| BIBLIOGRAFÍA | 28 |

1 INTRODUCCIÓN

1.1 DESCRIPCIÓN

Se trata de un juego con un formato similar al de las **historias interactivas**, donde el usuario elige el camino que quiere seguir en una narración. En este caso, utilizando para su manejo la voz.

Para ello necesitaremos una herramienta que nos ayude con el **Speech to Text** y el **Text to Speech**: el motor de las **Skills de Alexa**. El proyecto actual busca ser un ejemplo o inicio de un nuevo tipo de juego o modo de jugar, que puede ser adaptado fácilmente a todo tipo de narrativas.

En nuestro caso en particular, la temática de la historia será el mundo mágico de **Harry Potter**, por ser un tema muy conocido y atractivo para los usuarios.



1.2 OBJETIVO

Con la realización de este proyecto se busca conocer las bases del uso de interfaces conversacionales de un modo práctico y entretenido, mediante la realización de un juego interactivo. Por otro lado, también servirá como pretexto para profundizar en el uso de distintos lenguajes de programación y otras tecnologías.



1.3 ALCANCE

- Crear una narrativa interactiva, que será la base del juego
- Sistema de reconocimiento de voz
- Sistema de Text to Speech
- Utilizar un asistente inteligente para procesar las respuestas del usuario
- Crear un pequeño registro para que el asistente nos llame por nuestro nombre/username
- Utilizar un sistema de almacenaje de datos local (Opcionalmente online)
- Obtener información del usuario con APIs propias de Alexa
- Obtener información para la historia de APIs externas
- Uso de efectos de sonido
- Manejo de entonaciones
- Internacionalizar la aplicación
- Opcionalmente, hacer un apartado gráfico que muestre imágenes o animaciones dependiendo del progreso en la historia

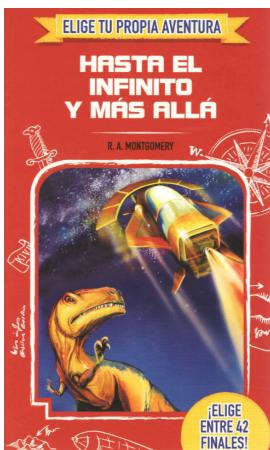
1.4 JUSTIFICACIÓN

Las **interfaces naturales** cada vez tienen más presencia, por su comodidad y su **accesibilidad**. Uno de los ejemplos más claros son las VUI o **interfaces por voz**. ¿Quién no conoce a alguien con un dispositivo Alexa en casa, o que utiliza el asistente de voz de Google en su móvil?

Con este proyecto se busca crear algo que pueda ser útil e innovador, descartando las ideas más habituales en este tipo de trabajos, como pueda ser hacer algún tipo de red social o un juego en 2D. Inicialmente me llamaba la atención esto último, ya que los juegos incluyen varias cosas que me gustan personalmente, como son la música y la narrativa. Finalmente, acabé uniendo estos dos conceptos (VUI y juego), llegando a la idea que en este proyecto se ha realizado.

Tras una pequeña investigación inicial, vi que apenas se ha utilizado este tipo de interfaces en el mundo de los videojuegos. Aunque pocos, sí que hay juegos en Android donde el mando principal es la voz del usuario, pero todos tenían elementos demasiado comunes: estilo de plataformas y mayor importancia del volumen con que se grita, que aquello que realmente se dice.

Tardé un tiempo en encontrar algo similar a lo que se me había ocurrido: un juego interactivo con VUI donde el centro del entretenimiento sea la narrativa, lo cual fue un aliciente para continuar con la idea, ya que era algo que no se había hecho demasiado y donde había todavía mucho espacio para innovar.



Tenía como referente aquellos juegos en consola de comandos donde se iba narrando una historia del tipo: “Te encuentras frente a una puerta”, y el usuario escribía “Abrir”, y la consola continuaba la acción, “Parece que algo la bloquea”... y poco a poco te adentrabas en la historia. También podría ser similar a juegos del estilo Dungeons and Dragons, a aquellos libros de “Elige tu propia aventura” o películas como Black Mirror: Bandersnatch.

También opté por esta idea por la documentación y cursos existentes, ya que facilitan el aprendizaje sobre este tipo de interfaces. Además, podemos utilizar tecnologías ya existentes, como el asistente de voz de Google o Alexa, para desarrollar el proyecto. Decidí optar por la segunda opción por los lenguajes que utiliza: JavaScript con Node.js, y Python.

En cuanto al alcance de la aplicación, buscaba crear un proyecto sencillo pero completo donde utilizar varias de las tecnologías y técnicas aprendidas durante el curso. Por ello, aparte de los requisitos más estrechamente ligados a las VUI, quería utilizar recursos que viniesen de API REST, añadir música y efectos, utilizar algún sistema de almacenaje de datos para poder guardar información como el nombre del usuario...

Finalmente, añadir que me parece un buen momento para aprender sobre VUI, ya que ahora mismo disponemos de suficiente material como para poder formarnos con mayor facilidad, y es una tecnología en crecimiento, además de estar muy ligada a otros campos cada vez más relevantes como pueden ser la domótica, la realidad virtual o la búsqueda de una mayor accesibilidad a la tecnología que incluya a invidentes, personas con movilidad reducida...

2 PLANIFICACIÓN Y ANÁLISIS

2.1 Planning

Inicialmente, se creó un planning orientativo, estimando el tiempo que se dedicaría a cada apartado:



Este esquema fue anterior a la investigación en profundidad de las tecnologías a utilizar, por lo que ha variado notablemente con el planteamiento real. No obstante, me parece interesante comentar la causa de estas diferencias. Por un lado, al querer profundizar en tecnologías no estudiadas o vistas de forma superficial durante el curso, el apartado de formación ha sido más extenso.

Sin embargo, no ha supuesto un problema para el desarrollo general del proyecto, ya que gracias a este exhaustivo aprendizaje, la parte de desarrollo ha sido más sencilla. Además, conocer el funcionamiento de las *Skills* de Alexa me ha permitido evitar crear mi propio sistema *Text to Speech / Speech to Text*, lo que me ha dado un tiempo que me ha permitido personalizar más la aplicación, mejorar la narrativa y crear una documentación más útil y específica, quedando así el esquema:



2.2 Análisis

2.2.1 Descripción general

Un juego inmersivo e interactivo que utiliza la voz para su manejo, utilizando **interfaces VUI**. Requiere de una herramienta que nos permita hacer tanto el *Speech to Text* como el paso inverso, el *Text to Speech*, es decir, que por un lado permita que nuestro sistema se comunique con el usuario mediante audio y, por otro lado, sea capaz de interpretar las órdenes que el usuario lanza mediante su voz.

Esta herramienta será el motor de las **Skills de Alexa**, que nos permite tanto desarrollar como probar nuevas aplicaciones con el reconocido sistema de Alexa. El código se desarrollará utilizando **Node.js**. Las herramientas de desarrollo de Alexa también nos ofrecen la opción de almacenar datos, de modo que no tenemos por qué implementar una base de datos ajena, a no ser que queramos tener un mayor control o una cantidad grande de información. En este caso, utilizaremos los atributos de sesión que nos ofrece el **Kit** para asegurar la persistencia de datos como el nombre del usuario, estado de la partida (momento de la historia en el que se quedó el jugador), zona horaria...

Veremos el funcionamiento general de la aplicación en los posteriores apartados. Adicionalmente a este funcionamiento base, estableceremos los diálogos en diferentes archivos por idioma, facilitando la internacionalización, además de usar los sonidos y entonaciones disponibles en las APIs propias de Alexa. También daremos riqueza con otros efectos o música de APIs externas.

2.2.2 Análisis de productos similares

Tras una investigación más en profundidad, se observa que actualmente hay esencialmente tres tipos de juegos relacionados con VUI:

En primer lugar encontramos aquellas **relacionadas con el volumen**, donde el mando principal es la potencia de voz y no lo que se dice. Suelen ser de formato plataformas y tienen una mayor presencia en plataformas como Android. Entre ellos, podemos destacar dos: Scream Hero y Yasuhati.



En segundo lugar tenemos aquellas aplicaciones basadas en **pregunta-respuesta**. Encontramos varios **Quiz** en las aplicaciones de Google con temáticas variadas: Juego de Tronos, Rompe Ralph...

En Alexa también tenemos apps similares, además de aquellas basadas en famosos concursos de preguntas como *Boom* o *Pasapalabra*; o en juegos de mesa como *Trivial Pursuit*.



Por último, aparecen algunos más similares a la idea de este proyecto, como por ejemplo *Escape Room* o *Mi héroe a la batalla*, ambos de la tienda de Alexa, donde existirá una narrativa y el usuario tendrá un cierto poder para elegir el destino de su personaje.



“Alexa, lanza mi héroe a la batalla”

En general, como pequeño resumen sobre los juegos actuales en VUI, suelen ser sencillos, pensados para ratos cortos, donde en apenas unos minutos podemos echar una partida. Las más habituales son aquellos basados en preguntas, aunque también encontramos “juegos” que nos ayudan a aprender idiomas, donde la app nos pide traducir palabras, repetir pronunciaciones... En el caso de Alexa, suelen estar diseñados para *Amazon Echo* u otros dispositivos de altavoz, es decir, sin añadir pantalla.

2.2.3 Tecnologías utilizadas

Para facilitar el proceso de *Text to Speech* y *Speech to Text*, lo más sencillo era utilizar motores a los que podemos tener acceso y sabemos que funcionan bien, como son Alexa o el asistente de voz de Google. La decisión de cuál de ellas tomar vino determinada por los lenguajes que tienen por detrás. Google utiliza C++, a través de su herramienta gratuita *Timer*, que utiliza también comunicación http para el paso de mensajes y un servidor en Node.js.

Alexa, por su parte, puede programarse con *Python* o *JavaScript/Node.js*. Personalmente, quería profundizar en estos dos últimos, por ello elegí el motor de Alexa Skills.

Además, Amazon ofrece herramientas de desarrollo para sus aplicaciones. Y estas serán las más utilizadas en el proyecto: developer console, Amazon Web Service (AWS), Alexa Skills Kit (ASK), con herramientas para la persistencia, internacionalización... Adicionalmente se usan servicios de API REST para obtención de datos, APIs internas y externas para añadir sonido y entonación, Git y GitHub...

2.2.4 Análisis DAFO



Analizando la situación del mercado y la propia aplicación que desarrollamos, podemos elaborar un pequeño gráfico DAFO.

En primer lugar, vemos que entre nuestra debilidades encontramos la dependencia de otros servicios, haciendo referencia a aquellos que nos dan el soporte para el manejo de voz. Esto nos limita el número de herramientas o lenguajes que podemos usar a los que utilizan estos motores. Además, como es un sector en crecimiento, se puede entender tanto como debilidad como oportunidad el constante cambio que sufre el modo de programar este tipo de aplicaciones.

En segundo lugar, nuestras amenazas pueden ser: que las herramientas de las que dependemos nos exijan un pago por su uso que no podemos afrontar; la competencia (ya sea de productos audiovisuales o sonoros similares) o que no se llegue a implantar el uso de VUI en móviles.

En cuanto a las fortalezas, sigue siendo una idea innovadora y poco explorada, con muchas posibilidades tanto creativas como económicas. Además, puede adaptarse a muchos tipos de narrativa, lo que la hace versátil. Finalmente, al utilizar el audio como medio, facilita su acceso a personas con problemas en la visión, dificultad de movimiento...

En último lugar tenemos las oportunidades, que van por la misma línea, el gran crecimiento que está sufriendo este sector, donde se ve un uso cada vez más común por parte de los usuarios, y un interés por parte de las empresas. Además, no solo las VUI, sino también el sector del videojuego buscan nuevas ideas y tienen un gran poder económico.

2.2.5 Requisitos funcionales

La aplicación debe poder:

- Utilizarse únicamente mediante la voz
- Obtener el nombre del usuario
- Obtener datos de APIs internas de Alexa
- Obtener datos de APIs externas
- Tener la capacidad de almacenar información
- Tener la capacidad de almacenar archivos

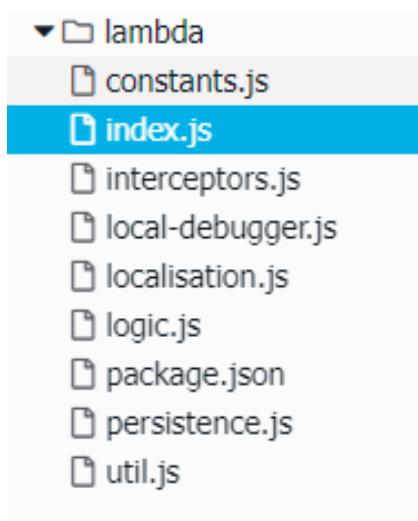
2.2.6 Requisitos no funcionales

- La narrativa debe ser fluida, conexa y con sentido
- El *Speech* se debe entender perfectamente
- Los tiempos de respuesta deberán ser cortos
- Ser mantenible y adaptable (de cara a añadir nuevos capítulos o funcionalidad)
- Compatibilidad con dispositivos Alexa, Android, Windows, Mac...

3 IMPLEMENTACIÓN

3.1 DISEÑO

3.1.1 Estructura y clases



La lógica de las skills de Alexa se aloja en la carpeta **lambda**. Por defecto, cuando creamos una nueva skill, tendremos una clase principal **index.js**, donde ya encontramos manejadores comunes como el de launch, ayuda, salir... **util.js**, que como veremos en el apartado de almacenamiento, tiene un método fundamental para el manejo de archivos multimedia, y **package.json**, que nos permite importar las librerías necesarias para el funcionamiento de la skill. Aquí tendremos por ejemplo el ask y el sdk, y añadiremos las importaciones necesarias para la persistencia, internacionalización, obtención de datos de timezone...

La clase index quedaría muy cargada si mantuviésemos únicamente estas clases, con funcionalidades muy dispares que se pueden extraer a otras nuevas. Nos quedaremos con la estructura de la imagen, destacando las siguientes clases:

interceptors.js: Contiene los distintos Interceptors, unos métodos que responderán a determinados eventos. Por ejemplo, al inicio de la sesión, comprobaremos el lenguaje en el que está configurado el sistema. También tendremos aquí una parte del proceso de almacenado y obtención de datos, o los interceptores necesarios para realizar los log, que nos permitirán ver en formato json tanto la entrada como la salida de datos durante toda la conversación con Alexa.

localisation.js: Por un lado, para no mezclar la lógica con los mensajes, y por otro, para facilitar la internacionalización, creamos esta clase. En ella encontraremos una serie de objetos (uno por cada lenguaje habilitado) con una serie de atributos parejos. Dependiendo de el lenguaje en el que estemos utilizando la aplicación, llamaremos a uno u otro objeto, de este modo, para un mismo atributo ('WELCOME_MESSAGE') podremos obtener respuestas en distintos idiomas. Profundizaremos en esto en el apartado de implementación.

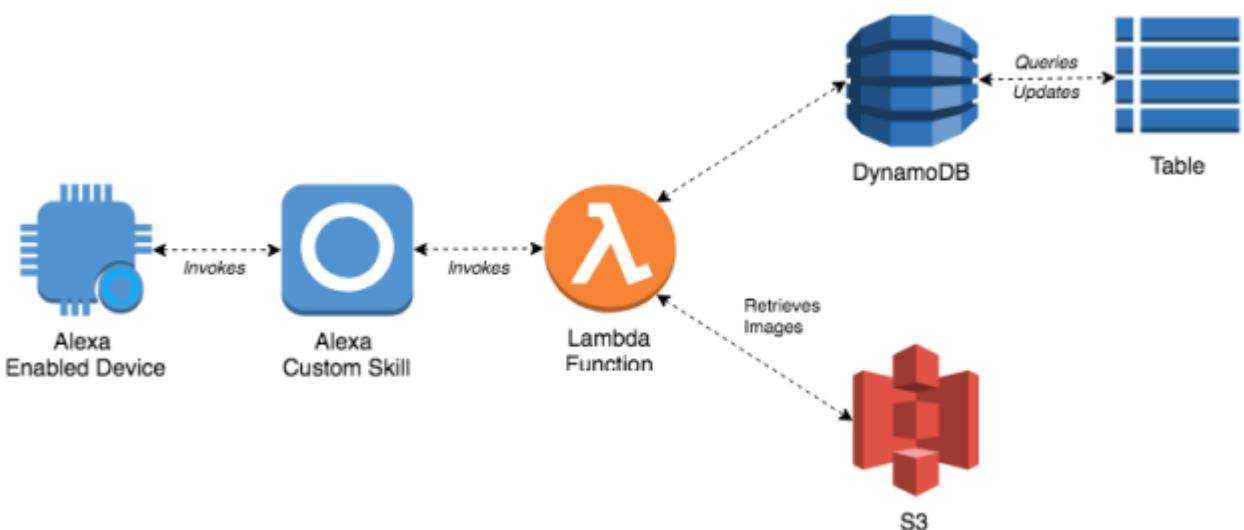
logic.js: Un conjunto de métodos útiles que se salen del uso por defecto de una skill. Por ejemplo, hemos añadido aquí los métodos necesarios para la consulta de una API Rest externa a los servicios de Alexa, o un método que nos devolverá escrito un mes a partir de su código numérico, para facilitar el speech.

persistence.js: aquí añadiremos la lógica de nuestro adaptador de persistencia. Lo analizaremos en más detalle en el apartado de almacenamiento de datos.

Sacando toda esta funcionalidad de nuestro index, nos quedaríamos con una clase principal mucho más legible que principalmente contenga los **handler**, es decir, que posea la lógica necesaria para responder a las distintas interacciones del usuario.

3.1.2 Almacenamiento de datos

Alexa facilita las herramientas necesarias para la persistencia de datos e incluso el almacenamiento de archivos en el **ASK** (Alexa Skills Kit). La aplicación desarrollada en este proyecto se ha hecho utilizando el sistema de Alexa hosted-skills. Esto nos permite almacenar tanto nuestro código como cualquier otro recurso en **AWS** (Amazon Web Service). Este almacenamiento se divide en dos partes, por un lado, la persistencia de datos, y por otro, el almacenamiento de ficheros pesados (que en nuestro caso en específico, serán aquellas pistas de sonido ajenas a las APIs de Alexa).



Cuando se crea una hosted-skill en Alexa, se obtiene acceso a una tabla de **Amazon DynamoDB** para guardar los **datos persistentes**. Esta tabla de DynamoDB es parte del nivel gratuito de AWS, que proporciona almacenamiento gratuito dentro de unos límites de uso. Todos los datos que el usuario almacene en su tabla de DynamoDB están completamente encriptados con claves administradas por AWS. Del mismo modo, al crear este tipo de skills, tendremos acceso a un depósito para el almacenamiento de archivos multimedia, denominado **Amazon S3**. Vemos un ejemplo de uso ambas:

Para comenzar a usarlas, tendremos que importar sus respectivos adapters, así que deberemos añadir estas líneas a nuestro archivo package.json

```

"ask-sdk-s3-persistence-adapter": "^2.7.0",
"ask-sdk-dynamodb-persistence-adapter": "^2.7.0",
  
```

Separaremos la lógica de la persistencia en una clase a parte, que nos permitirá acceder con facilidad a los recursos en nuestra tabla o en nuestra base de datos S3. Una vez importada esta clase a nuestra clase principal “index.js”, la estableceremos como Custom Persistence, y ya podremos utilizar los diferentes modos de persistencia. Por un lado, utilizaremos un sistema de clave-valor para almacenar datos en DynamoDB. Para guardar estos datos, utilizaremos las funciones del attributesManager:

```
module.exports={

  getPersistenceAdapter(tableName) {
    function isAlexaHosted() {
      return process.env.S3_PERSISTENCE_BUCKET;
    }
    if (isAlexaHosted()) {
      const {S3PersistenceAdapter} = require('ask-sdk-s3-persistence-adapter');
      return new S3PersistenceAdapter({
        bucketName: process.env.S3_PERSISTENCE_BUCKET
      });
    } else {
      const {DynamoDbPersistenceAdapter} = require('ask-sdk-dynamodb-persistence-adapter');
      return new DynamoDbPersistenceAdapter({
        tableName: tableName || 'una_aventura_magica_table',
        createTable: true
      });
    }
  }

  const {attributesManager} = handlerInput;
  const requestAttributes = attributesManager.getRequestAttributes();
  const sessionAttributes = attributesManager.getSessionAttributes();
}
```

El guardado y la obtención de un dato almacenado quedaría así:

```
sessionAttributes['nombre'] = nombre;
```

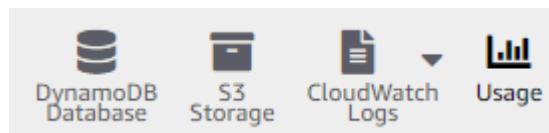
En este caso crearemos, de no existir, una clave nombre relacionada con nuestra variable local.

```
const nombre = sessionAttributes['nombre']
```

Por el contrario, podemos almacenar en una variable local el dato guardado en nuestra tabla indicando su clave, en este caso '**nombre**'.

El almacenamiento de **archivos multimedia** tiene una mayor complejidad. Nos centraremos sobre todo en los archivos de audio mp3, ya que son los que hemos utilizado en este proyecto.

Subir un archivo es sencillo e intuitivo. En la parte superior de nuestra consola de desarrollador, veremos una serie de iconos correspondiente con las diferentes herramientas disponibles, entre ellas, el almacenamiento S3.



Una vez pulsamos, accederemos a nuestros Media. En esta pantalla podremos ver aquellos archivos que ya hemos subido (también podremos ver detalles más específicos si los clickeamos), además de opciones para subir nuevos elementos, eliminar, distribuir en carpetas...

The screenshot shows the AWS S3 console with the 'Media' bucket selected. The 'Objetos' tab is active, displaying two items:

| Nombre | Tipo | Última modificación | Tamaño | Clase de almacenamiento |
|----------------|------|-----------------------------|----------|-------------------------|
| intro.mp3 | mp3 | 7 Jun 2022 11:23:25 AM CEST | 104.5 KB | Estándar |
| wimwardiun.mp3 | mp3 | 7 Jun 2022 11:27:25 AM CEST | 15.0 KB | Estándar |

Antes de volver a nuestro código, deberemos tener en cuenta dos cosas:

- Los audios mp3 no pueden tener una duración superior a 90 segundos
- Aunque tengan esta extensión, puede que no estén formateados en un modo que Alexa entienda, por eso es necesario hacer una transformación previa a su utilización. Existen herramientas como [jovo.tech](#) que adaptan los mp3 a los estándares necesarios en Alexa, Google Assistant...

Los archivos que subimos a S3 se almacenan de forma privada por defecto. Para utilizarlos sin necesidad de cambiar su visibilidad, disponemos del código autogenerado en util.js:

```
module.exports.getS3PreSignedUrl = function getS3PreSignedUrl(s3ObjectKey) {
    const bucketName = process.env.S3_PERSISTENCE_BUCKET;
    const s3PreSignedUrl = s3SigV4Client.getSignedUrl('getObject', {
        Bucket: bucketName,
        Key: s3ObjectKey,
        Expires: 60*1
    });
    console.log(`Util.s3PreSignedUrl: ${s3ObjectKey} URL ${s3PreSignedUrl}`);
    return s3PreSignedUrl;
}
```

Este es el método principal que encontraremos en util.js, que nos permite obtener un objeto privado de nuestro almacenamiento de S3 mediante su clave. Además, podemos establecer un tiempo de expiración, por si ocurriese cualquier error, poder manejarlo. Las claves tendrán un formato **Media/archivo.mp3**. Como pueden ocurrir problemas con caracteres especiales como “&”, se recomienda también añadir un replace, quedando el código de obtención de la ruta de un objeto multimedia así:

```
const intro = Util.getSignedUrl('Media/intro.mp3').replace(/&/g, '&');
```

Una vez tengamos esto, solo tendremos que añadirla al igual que el resto de audios, utilizando etiquetas <audio> con un atributo “src” indicando su ruta:

```
`<audio src="${intro}">`.
```

Como curiosidad, comentar que Alexa solo admite rutas https, por lo que si tenemos algún archivo multimedia que nos interese en una web http, la mejor opción será descargarlo y utilizar el almacenamiento en AWS con S3 para acceder a él.

3.2 IMPLEMENTACIÓN

3.2.1 Funcionamiento general de las VUI y de las skills de Alexa

Como hemos utilizado una tecnología fuera de lo que habitualmente se estudia en DAM, explicaremos en primer lugar el funcionamiento de las interfaces conversacionales, en relación con las Alexa-skills y, posteriormente, veremos varios fragmentos de código característicos de nuestra aplicación.

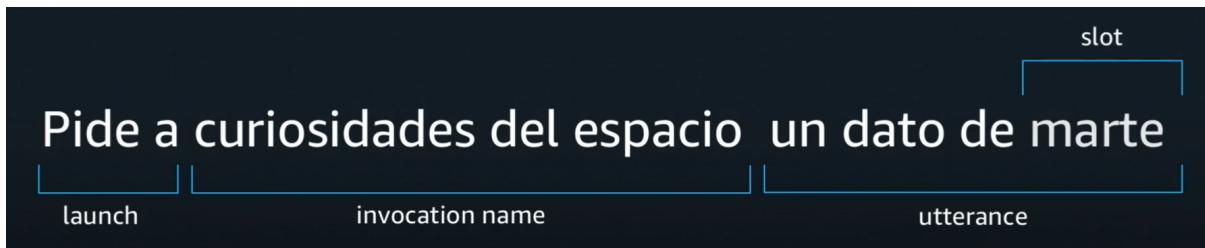
Las Skills de Alexa se alimentan de un servicio en la nube, donde se realiza una etapa ASR (Automatic Speech Recognition) para el reconocimiento de voz, y NLU (Natural Language Understanding) que buscan el significado de esas frases detectadas.

En líneas generales, debemos tener en mente que una skill tendrá dos partes diferenciadas:

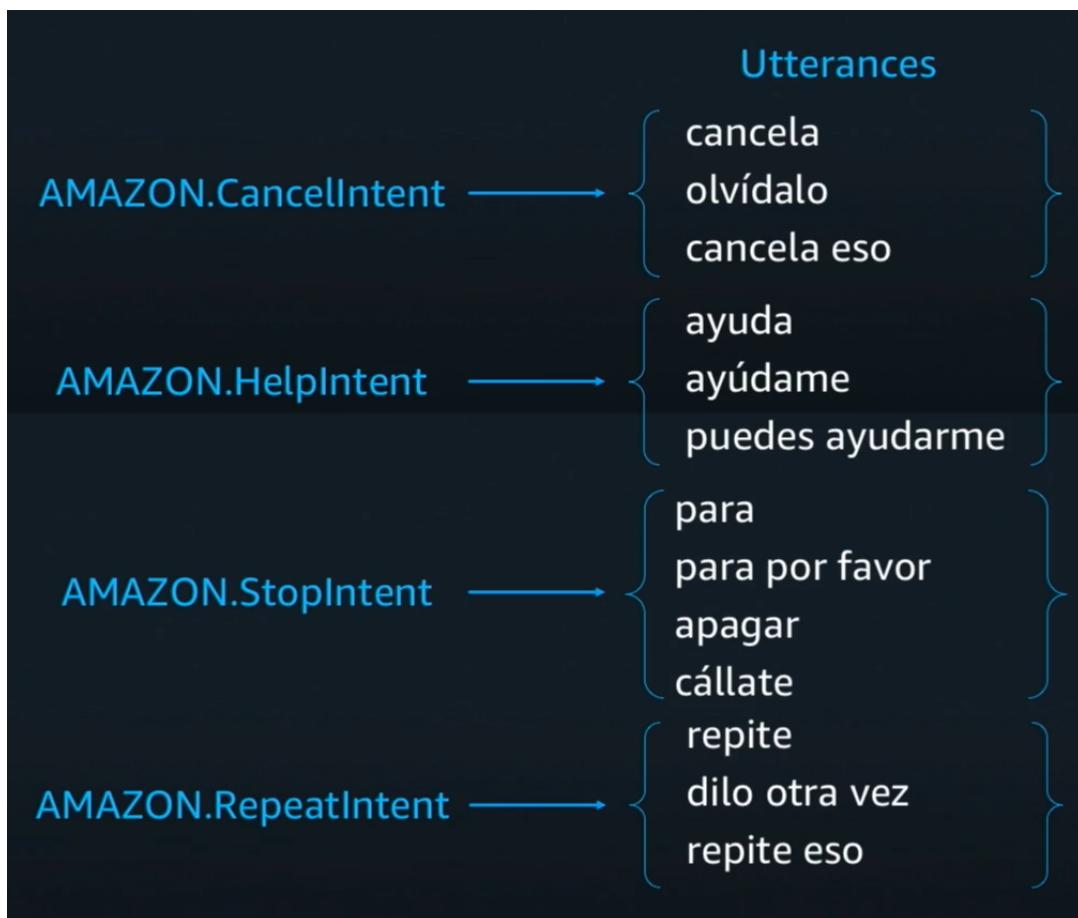
Front-end: ¿Qué es lo que tu skill tiene que comprender cuando el usuario hable?

Back-end: ¿Cómo responde nuestra skill a las peticiones del usuario?

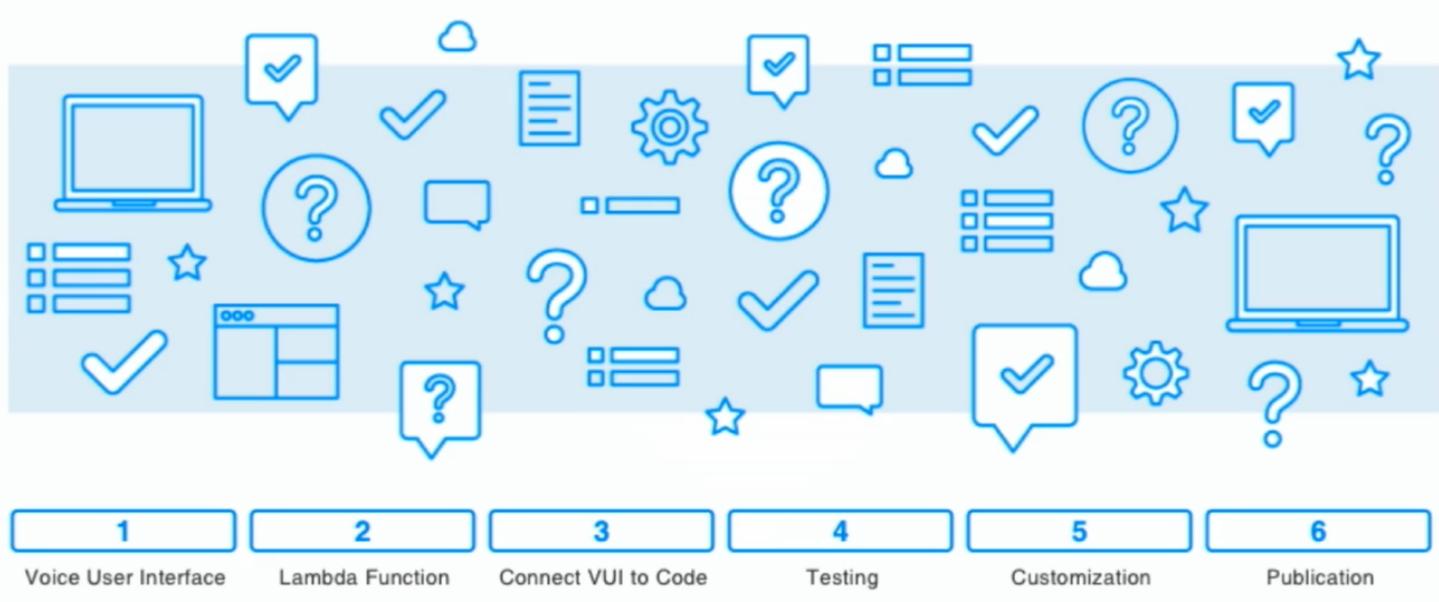
Cuando nos comunicamos con una interfaz conversacional, lo primero que tenemos que decir es la **palabra de lanzamiento o despertar (wake word)**, normalmente, se utiliza “Alexa”, aunque es configurable. Luego viene la **acción (launch)**: empieza, inicia, reproduce... y después tenemos la **invocation name**, aquello sobre lo que queremos realizar la acción. Esta se puede especificar con **enunciados o utterance**. Dentro de ellos podemos sacar datos concretos en forma de variable o **slot**. Veremos un ejemplo sencillo cuando hablemos de la obtención del nombre en nuestra aplicación. Podemos ver la estructura de una petición en este ejemplo:



Como estas frases buscan una acción, reciben el nombre de **intenciones** o **intents**. Estas serán la base de la interacción entre el usuario y la skill. Cuando el sistema detecte un intent de un tipo determinado, tendrá que realizar una acción consecuente. Por ejemplo, si el usuario dice “cerrar”, nuestra skill deberá responder a esa petición cerrando la aplicación. Para ello utiliza los **manejadores** o **handlers**, que contendrán la lógica necesaria para, en este caso, cerrar el programa. Alexa incorpora unos cuantos intent habituales, con sus respectivos handler, al crear una nueva skill.



Podríamos distinguir estos 6 pasos en el desarrollo de una skill:



- La creación de la VUI, donde tendremos que tener en cuenta para cada idioma qué es lo que se debe entender
- La lógica de la aplicación
- Conectamos ambas partes
- Hacemos una fase de testing
- Corregimos errores y adaptamos la funcionalidad
- Publicación o distribución (Si queremos publicarla en la tienda de Skills de Alexa, deberá pasar por un proceso de certificación)

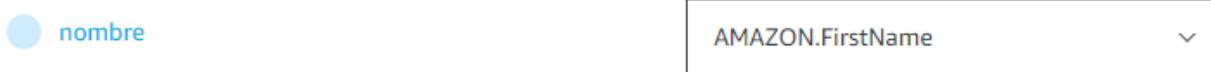
*Si quieres saber más del funcionamiento de las Alexa skills, consulta el [pequeño tutorial](#) que encontrarás en mi GitHub.

3.2.2 Código característico de nuestra aplicación

En este proyecto hemos querido explorar las distintas posibilidades que ofrece Alexa, por eso incluimos funcionalidades diversas, además de distintas soluciones posibles para problemas similares. Por ejemplo, usar tanto APIs internas como externas, usar el almacenamiento que ofrece Alexa en lugar de uno al que ya estuviésemos acostumbrados. Del mismo modo, usar las herramientas de código que nos ofrecen y su modo de testeo...

Pasaremos brevemente en este apartado por los aspectos más característicos del código:

Slots: Podríamos entenderlos como pequeños fragmentos de información que nos aporta el usuario que queremos almacenar de forma local para trabajar con ellos. El ejemplo más clásico es el del nombre. Dentro del intent donde pediremos su nombre al usuario, definiremos un nuevo slot, al que asignaremos un identificador (nombre) y un tipo (FirstName). Contamos con una serie de tipos predefinidos por Amazon, pero también podemos crear nuestros propios tipos personalizados.



Podríamos utilizar esta variable dentro del utterance (la petición del usuario), de este modo:

cambiar nombre por {nombre}

O definir que esta “variable” debe ser rellenada obligatoriamente. De modo que si no viene implícita en la acción (por ejemplo si el usuario solo dice “cambiar nombre”), Alexa deberá pedirnos específicamente que digamos esta información. Ahí entran en juego los denominados **speech prompts**, que serán las frases utilizadas por Alexa para pedirnos este dato. Siguiendo con el ejemplo, podríamos tener prompts como estos:

¿Cuál es el nuevo nombre?

¿Qué nombre desea para su personaje?

De modo que si el nombre no viene especificado, Alexa nos preguntará por él utilizando aleatoriamente alguna de las frases que hemos definido en estos prompts.

Del mismo modo, el desarrollador de la Skill tendrá que manejar un abanico de posibles respuestas (utterances) del usuario.

User utterances ?

What might a user say in response?

lo llamaré {nombre}

me llamo {nombre}

se llama {nombre}

se llamará {nombre}

{nombre}

Adicionalmente podemos requerir una confirmación, para asegurarnos de que realmente el dato captado por Alexa es el que el usuario quiere. Añadiremos entonces una serie de preguntas de sí o no para confirmarlo:

Slot Confirmation

Does this slot require confirmation? ?

Alexa Speech Prompt ?

What will Alexa say to prompt the user to confirm this slot?

¿Seguro que quieres {nombre} como nombre?

Por otro lado, podemos manejar las posibles respuestas estableciendo unos posibles valores aceptados o definiendo otro tipo de validaciones, como en este ejemplo, donde si el usuario nos da un número mayor a 77, Alexa lo dará como no válido y hará una nueva petición:

Active Validation Rules

Accept less than a value

Delete Prompts for a new value when greater than or equal to the specified number

Accept less than:
than:

77

What will Alexa say to ask the user for an acceptable value? +

Lo siento, este número es demasiado alto, tendrás que especificar uno menor Delete

Para crear una restricción teniendo en cuenta una serie de valores, podemos crear un tipo customizado de slot. En nuestro proyecto hemos utilizado esta opción para hacer un pequeño juego de adivinanzas, donde las respuestas válidas tienen que estar en la lista que definamos. Aquí podemos añadir identificadores para acceder más fácilmente a los elementos, o incluso sinónimos, como podemos ver en este ejemplo:

fuego

5

Add synonym

+

llama

Delete

Para ver la utilización de estos slots en el código, volveremos al ejemplo de la elección de nombre:

```
const CambiarNombreIntentHandler = {
  canHandle(handlerInput) {
    return Alexa.getRequestType(handlerInput.requestEnvelope) === 'IntentRequest'
      && Alexa.getIntentName(handlerInput.requestEnvelope) === 'CambiarNombreIntent';
  },
  handle(handlerInput) {
    const {attributesManager} = handlerInput;
    const requestAttributes = attributesManager.getRequestAttributes();
    const sessionAttributes = attributesManager.getSessionAttributes();
    const {intent} = handlerInput.requestEnvelope.request
    const nombre = intent.slots.nombre.value;

    sessionAttributes['nombre'] = nombre;

    const speakOutput = requestAttributes.t('NAME_CONFIRMATED_MESSAGE', nombre);

    return handlerInput.responseBuilder
      .speak(speakOutput)
      .reprompt(speakOutput)
      .getResponse();
  }
};
```

Este handler contendrá la lógica de los IntentRequest de tipo CambiarNombreIntent. Se llamará a este código cuando el usuario diga alguna de las utterances esperadas como “deseo cambiar mi nombre”, “elegir nuevo nombre”... Utilizaremos entonces los métodos del handlerInput para obtener la información del intent, del cual especificamos que queremos dentro de sus posibles slots, el valor del que se llama “nombre”. Como nos hemos asegurado de que vamos a tener este valor (pues lo hemos puesto como obligatorio) no nos hará falta manejar un posible null.

En este mismo handler vemos varias posibilidades de utilización de esta variable obtenida del slot. Podríamos guardarla como atributo de sesión, para asegurar su persistencia y, como ocurre en nuestra aplicación, poder recibir en el futuro al usuario por su nombre. También podemos añadir su valor a un speak para que Alexa nos llame con nuestro nombre. Al introducir esta variable como parámetro, sustituiremos el %s del texto por su valor:

```
NAME_CONFIRMATED_MESSAGE: 'Es un placer conocerte, %s'.
```

Si cambiásemos nuestro nombre a “Mario”, por ejemplo, al lanzar el mensaje, la skill nos diría : “Es un placer conocerte, Mario”. Esta es una opción muy buena para personalizar nuestros mensajes.

Explorando otro modo de validación de slots, hemos añadido un pequeño reto de adivinanzas donde el usuario nos dará una respuesta, y manejaremos si es válida o no a nivel de código, con una estructura similar a la siguiente:

```
const {intent} = requestEnvelope.request;
const respuesta = Alexa.getSlotValue(requestEnvelope, 'Respuesta');
let validas=['fuego','llama','febrero','nombre','vela'];

if(validas.includes(respuesta)){ ...}
```

Vemos aquí otro modo de acceder al valor de un slot, con el método getSlotValue() de Alexa.

SpeakOut: Una vez recibida una intención del usuario, la manejaremos en el handler. Aquí haremos los cambios en variables, peticiones a servicios internos o externos... y por supuesto, daremos una respuesta al usuario. Para ello contamos con el método speak(), que recibirá un objeto compuesto por los strings que queremos que diga el dispositivo Alexa, a los que podemos añadir si queremos música, efectos o entonaciones.

*Las **entonaciones** o **Speechcons** son palabras, onomatopeyas o frases especiales que Alexa pronuncia de manera más expresiva. Las utilizaremos para hacer más cercano y humano los mensajes. Disponemos de toda una [librería de speechcons](#) con opciones para varios idiomas. En nuestra skill utilizamos algunas de estas entonaciones especiales. Vemos que estas se añaden utilizando unas etiquetas específicas. Como son más parecidos a un audio mp3 que a una lectura de palabras, Alexa no tiene definido un tiempo de separación entre el speechcon y lo que venga después. Para hacer esta pausa, tendremos que añadirla mediante una etiqueta <break>*

```
speakOutput += '<say-as interpret-as="interjection">ay, qué susto</say-as>'  
speakOutput += '<break time=\"1s\"/>'
```

Para entender el código de nuestra skill, también debemos conocer el método de internacionalización:

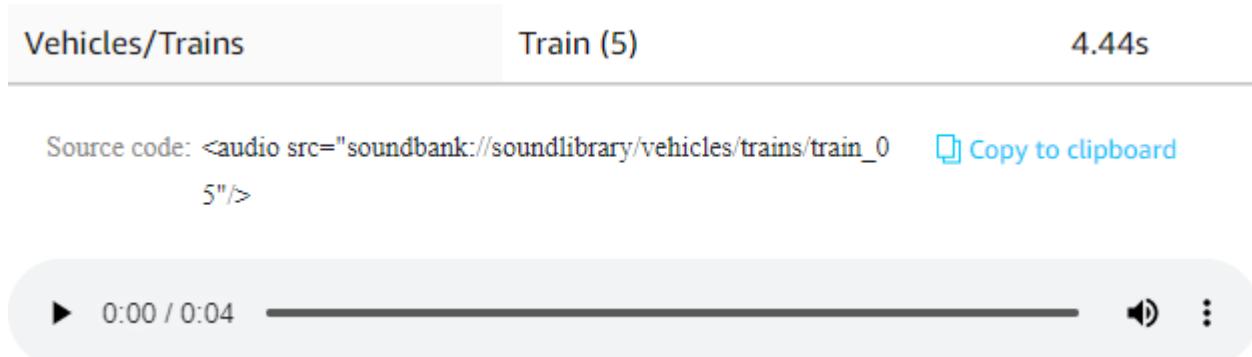
*Con **internacionalización** nos referimos a la posibilidad de disfrutar de la aplicación en varios idiomas. Para ello, además de hacer unas pequeñas configuraciones para añadir la posibilidad de cambio entre idiomas en Alexa, deberemos tener todos los textos disponibles en los distintos idiomas a utilizar. Con este fin creamos la clase localisation.js, que contendrá todas estas cadenas. Para facilitar su comprensión vemos un ejemplo más reducido de internacionalización:*

```
const i18n = require('i18next');  
const sprintf = require('i18next-sprintf-postprocessor');  
  
const languageStrings={  
    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. ¿Cuálquieres?'  
        }  
    }  
}
```

Podemos apreciar que necesitaremos las librerías de i18n para conseguirla. El centro de este proceso serán estos objetos (en, es..) que contienen el texto adaptado a cada idioma. Detectaremos el idioma el sistema con un interceptor, y dependiendo de cual sea, si disponemos de una opción para él, llamaremos a un "WELCOME_MESSAGE" u otro.

Finalmente para completar los speakOut, además de texto internacionalizado y entonaciones especiales, hemos añadido música y efectos. Para utilizar estos utilizaremos la etiqueta <audio> con un atributo "src" en el que añadiremos la ruta donde se encuentra el audio en formato mp3.

Alexa pone a nuestra disposición una [galería de sonidos](#) con multitud de efectos que podemos traer fácilmente a nuestro código, ya que dispone de una opción de copiado que incluye ya la etiqueta <audio> con el src correspondiente. Vemos por ejemplo el efecto del tren:



Vehicles/Trains Train (5) 4.44s

Source code: <audio src="soundbank://soundlibrary/vehicles/trains/train_05"/> [Copy to clipboard](#)

▶ 0:00 / 0:04

Con el buscador de que dispone, localizamos sonidos de tren. Encontraremos una selección que podemos desplegar y que nos indica el tiempo de la pista, su ruta, nos da la opción de copiarla directamente al portapapeles, reproducirla e incluso descargarla.

Podríamos entonces añadir este sonido a nuestra skill utilizando esa sentencia que nos facilita:

```
SOUND_TRAIN : '<audio src="soundbank://soundlibrary/vehicles/trains/train_05"/>'.
```

De un modo similar, podríamos utilizar otros mp3 que encontremos de forma online, sin necesidad de descarga, especificando su ruta en src, eso sí, ten en cuenta que solo se admitirán aquellos que vengan de páginas con protocolo https, y que deben cumplir las condiciones que comentábamos en el apartado de *Almacenamiento de datos*.

La otra opción sería utilizar archivos mp3 que tengamos a nivel local. Lo más sencillo es subirlo al almacenamiento que nos ofrece de forma gratuita Alexa en S3. Veámos cómo subir y acceder a esos audios en ese mismo apartado.

Vemos ahora un ejemplo de un speakOut complejo que incorpora muchas de estas opciones:

```
const intro = Util.getS3PreSignedUrl('Media/intro.mp3').replace(/&/g, '&');

let speakOutput = requestAttributes.t('SOUND_HORSE2');
speakOutput += requestAttributes.t('ARRIVE_MESSAGE');
speakOutput += `<audio src="${intro}">`;
speakOutput += requestAttributes.t('ARRIVE_MESSAGE2');
speakOutput += requestAttributes.t('ARRIVE_MESSAGE3', nombre);
speakOutput += requestAttributes.t('SOUND_OPEN_DOOR');
speakOutput += requestAttributes.t('ARRIVE_MESSAGE4');

return handlerInput.responseBuilder
    .speak(speakOutput)
```

API Rest: una opción que podemos utilizar para completar nuestros textos, es utilizar información sacada de una API Rest externa. Utilizaremos la librería axios. En nuestro caso, queríamos obtener información de los hechizos de una API:

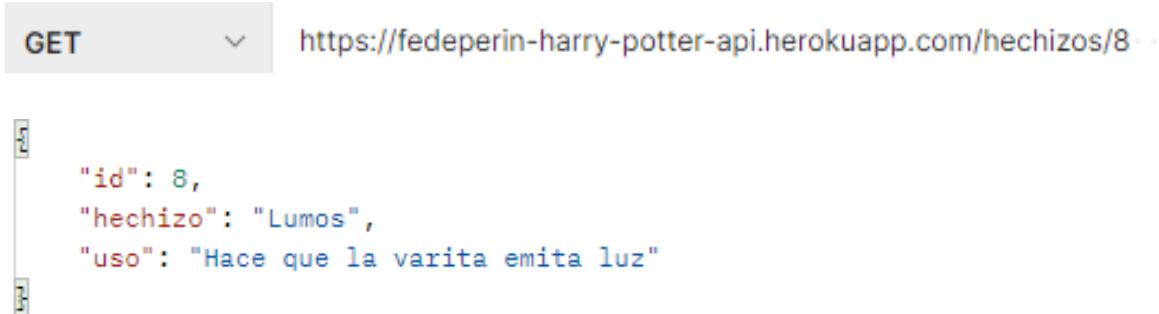
```
fetchHechizo(numero_hechizo){
    const url = 'https://fedeperin-harry-potter-api.herokuapp.com/hechizos/' + numero_hechizo;

    var config = {
        timeout: 2000,
    };

    async function getJsonResponse(url, config){
        const res = await axios.get(url, config);
        return res.data;
    }

    return getJsonResponse(url, config).then((result) => {
        return result;
    }).catch((error) => {
        return null;
    });
}
```

Estableceremos un timeout, y si obtenemos respuesta en ese tiempo, la devolveremos en forma de Json. Para ello, únicamente necesitaremos la url de la API. En este ejemplo en específico, podemos especificar el hechizo con una id numérica incremental, por lo que adaptamos el método para que pueda recibir un número que se añadirá al endpoint. Vemos la petición desde postman para comprender la estructura de la API:



Esto lo utilizaremos en varias ocasiones, siendo la más destacada la clase de encantamientos, donde dependiendo de la página que visitemos (que se corresponderá con el id del hechizo) obtendremos una información u otra. Es característico de este código, además, el uso de promesas, ya que, al no saber si la API externa está funcionando, si la petición es correcta... deberemos manejar el caso de que no tengamos una respuesta satisfactoria. El código quedaría así:

```
async handle(handlerInput) {
  const {attributesManager, requestEnvelope} = handlerInput;
  const requestAttributes = attributesManager.getRequestAttributes();
  const sessionAttributes = attributesManager.getSessionAttributes();
  let pagina = Alexa.getSlotValue(requestEnvelope, 'Page');

  let respuesta = await logic.fetchHechizo(pagina)
    .then(function(response) {
      return response;
    })
    .catch(function(error) {
      return null;
    });

  let speakOutput;
  if(respuesta==null){
    speakOutput=requestAttributes.t('ENCHANTMENT_FAIL_MESSAGE');
  }else{
    speakOutput=requestAttributes.t('ENCHANTMENT_MESSAGE', pagina, respuesta.hechizo, respuesta.uso);
  }
}
```

Obtención de datos del sistema: como hemos comentado en algunos puntos, hay datos que podemos extraer directamente desde el dispositivo: nombre, timezone, idioma...

Por ejemplo, en el caso de la fecha, podríamos establecer en el código cual es la que queremos usar:

```
timezone = timezone? timezone: 'Europe/Paris';

// getting the current date with the time
const currentDate = new Date(new Date().toLocaleString("en-US", {timeZone: timezone}));

const dia = currentDate.getDate();
const mesNum = currentDate.getMonth();
const mes = logic.getMesEscrito(mesNum);
```

Pero esto supondría un problema, ya que si estuviésemos fuera de la zona horaria de "Europe/Paris", nuestra aplicación tendría una fecha que no se correspondería con la del usuario. Por ello, es mejor obtener esta zona horaria del dispositivo. Para ello usaremos el siguiente código.

```
const deviceId = handlerInput.requestEnvelope.context.System.device.deviceId;

let timezone;

try{
  const upsServiceClient = serviceClientFactory.getUpsServiceClient();
  timezone = await upsServiceClient.getSystemTimeZone(deviceId);
} catch(error){...}
```

No se necesitan permisos especiales para obtener el timezone, por eso basta solo con esto, pero, por ejemplo, para obtener el nombre del usuario, sí que necesitaríamos que el usuario nos diese permiso.

```

try{
    const {permissions} = requestEnvelope.context.System.user;
    //Si no tenemos ningún tipo de permiso daremos error
    if(!permissions){
        throw{statusCode: 401, message: 'No permissions available'};
    }
    const upsServiceClient = serviceClientFactory.getUpsServiceClient();
    const profileName = await upsServiceClient.getProfileGivenName();

    if(profileName){
        sessionAttributes['nombre'] = profileName;
    }else{
        sessionAttributes['nombre'] = '';
    }
}catch(error){
    if(error.statusCode === 401 || error.statusCode === 403){
        handlerInput.responseBuilder.withAskForPermissionsConsentCard(constants.GIVEN_NAME_PERMISSION);
    }
}
}

```

Para que el usuario nos pueda dar este permiso, estableceremos en la pestaña de *Permissions* aquellos que queremos solicitar. De no haberlo obtenido, podemos pedírselo posteriormente. Esta es la opción que vemos en el último catch, donde lanzaremos una tarjeta al usuario pidiéndole ese permiso de *GIVEN_NAME*.

Permissions

Request users to access resources and capabilities. [?](#)

Device Address

- Full Address [?](#)
- Country/Region & Postal Code Only [?](#)

Customer Name

- Full Name [?](#)
- Given Name [?](#)

Continuidad de la narrativa: podemos establecer la continuidad de la historia basándonos en los intents lanzados por el usuario (si nos pide que saludemos, la skill saludará y quedará a la espera de nuevas órdenes o se cerrará). También tenemos la opción de navegar entre intents a través del código, lo que nos permite ligar varios intent de forma automática, sin necesitar una acción del usuario. Un ejemplo donde se ha utilizado esta segunda técnica es el handler de continuar:

```
if(!sessionAttributes['episodio']){
    return ComenzarAventuraIntentHandler.handle(handlerInput);
}else{
    episodio=sessionAttributes['episodio'];
    switch(episodio){
        case 1:
            return EscenaTrenIntentHandler.handle(handlerInput);
        case 2:
            return LlegadaHogwartsIntentHandler.handle(handlerInput);
        case 3:
            return SombreroSeleccionadorIntentHandler.handle(handlerInput);
        case 4:
            return CalizIntentHandler.handle(handlerInput);
```

En el resto de “escenas” se utiliza el primer método, ya que ayuda al usuario a introducirse más en la historia y a sentirse más partícipe. Además, al fragmentar en más intents la narrativa, podemos hacer métodos como el anterior que nos permitan ir a dónde lo dejamos con más exactitud, o métodos de ayuda más adaptados.

3.3 IMPLANTACIÓN

Al igual que con el resto del proceso, Amazon nos ayuda con la distribución de nuestra aplicación, incluyendo *guidelines* y herramientas en su ASK. Ofrece esencialmente dos modos para compartir nuestra custom skill.

Privada: estableceremos una url propia para nuestra *skill*, que podremos compartir con quien queramos. Todo aquel que acceda mediante este link podrá descargar la *skill*, por lo que tendremos que tener cuidado con quien lo compartimos.

Pública: una vez terminada, podremos subir a la *store* de Alexa nuestra skill, que podrá ser descargada por cualquier usuario que lo desee. Podemos establecer algunos límites, como por ejemplo, limitar los países o regiones donde se distribuye.

En ambos casos necesitaremos primero pasar un proceso de certificación. Nuestra skill será testeada en materia de seguridad, funcionalidad y experiencia de usuario. Si pasa todos los filtros, ya podremos publicarla.

Envío



Tu skill está lista para enviarse. Una vez enviada la skill, realizaremos una última prueba funcional.

Preferencias de publicación

Certificar y publicar

Publicamos automáticamente tu skill en la tienda de Skills de Alexa después de que supere la certificación.

Solo certificar

Puedes elegir cuándo publicar tu skill después de que haya aprobado la certificación. [Learn more](#)

Seguramente estos test nos den errores inicialmente, ya que tendremos que llenar unos formularios aportando datos ajenos al código como: **iconos** de la skill, **nombre público y descripción, categoría, privacidad y disponibilidad**. Estos dos últimos puntos son los más característicos, ya que tendremos que responder en un formulario si utilizamos información personal, si añadimos publicidad, etc. Podríamos entenderlo como la parte legal de la aplicación. En nuestro caso, como queremos obtener el nombre del usuario para personalizar la interacción con la skill, tendremos que añadir un documento con la **política de privacidad**.



Privacy & Compliance

Gettin' legal with it.

Does this skill allow users to make purchases or spend real money? *

- Yes
 No

Does this Alexa skill collect users' personal information? *

For example: anything that can identify the user such as name, email, password, phone number, birth date, etc.

- Yes
 No

Your skill can collect users' personal information only if required to support and improve the features and services provided by your skill.

Is this skill directed to or does it target children under the age of 13? *

Please indicate if this skill is directed to children under the age of 13 (for the United States, as determined under the [Children's Online Privacy Protection Act \(COPPA\)](#)). Not sure? [Learn More](#).

- Yes
 No

Does this skill contain advertising? *

- Yes
 No

Además de la limitación geográfica en la distribución, es interesante en la parte de disponibilidad, la opción de permitir **Beta Test**. Esto permite enviar la skill a personas específicas que podrán ir probando el producto mientras este sigue en fase de desarrollo.

Beta Test

- Give access to skill beta testers by adding their email addresses. Once Beta Test is enabled, you can continue to edit your skill until you are ready to submit your skill for Amazon review.

3.4 DOCUMENTACIÓN

Podréis encontrar toda la documentación en la wiki de GitHub:
<https://github.com/MarioGonzalezGomez/JuegoConAlexaSkills/wiki>

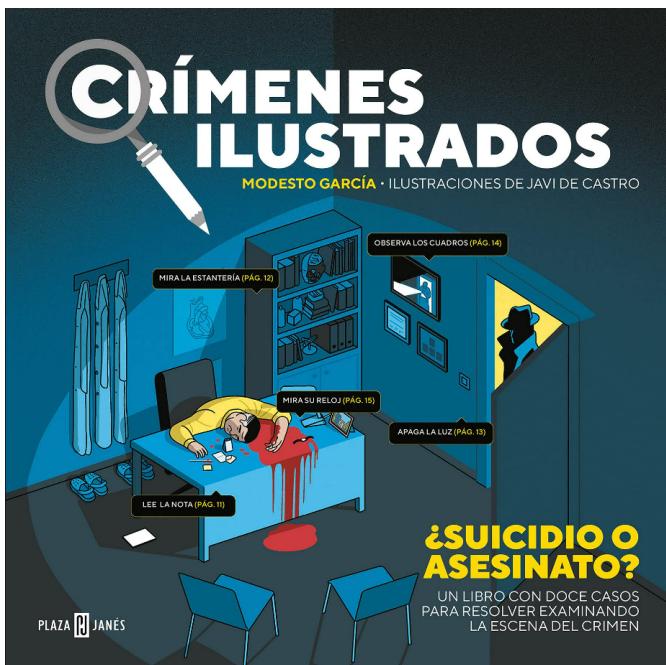
Durante el periodo de formación, se desarrolló una documentación técnica que explica todo lo utilizado en esta aplicación. Decidí cuidar más esos apuntes que tomaba a nivel personal para poder ofrecerlos como una especie de tutorial o manual técnico que fuese útil para todos aquellos que quieran realizar un proyecto con esta tecnología. También lo encontraréis en mi github.

4 TRABAJO FUTURO

Al trabajar con dispositivos Alexa, no solo disponemos de audio, si no que también hay casos en los que tenemos acceso a una pantalla. Esto nos permitiría mediante APL trabajar también con la parte gráfica. No obstante, la idea de este proyecto es hacer un juego interactivo que sea accesible también para personas con movilidad reducida o con problemas visuales. Teniendo en cuenta a estos últimos, lo ideal sería que la parte gráfica no tuviese relevancia en el funcionamiento o la dinámica de la historia, quedando como un elemento principalmente estético.

Esta skill buscaba ser un ejemplo, más que una aplicación completa en sí misma, por lo que podría dedicarse tiempo a su mejora (extensión de la historia, más posibilidades de elección...). Por otro lado, también se podría añadir funcionalidad como, por ejemplo, la de permitir, que en un mismo dispositivo pueda haber varias “partidas guardadas”.

En la narrativa de este proyecto no encajaba otra de las funciones más usadas en Alexa, los recordatorios. Sería interesante buscar un modo de incluirlos.



Finalmente, creo que sería interesante mezclar esta idea con proyectos como el de Crímenes Ilustrados, donde necesitamos recursos fuera del libro, o en nuestro caso la skill, para resolver los enigmas. Por ejemplo, que necesitemos buscar las coordenadas de un punto para poder lanzar un intent específico (teniendo que recurrir a Google maps o el medio que se prefiera para obtenerlas), dar mensajes secretos en código morse que tendrá que descifrar el usuario...

5 CONCLUSIONES

Considero que ha sido un proyecto enriquecedor, que me ha ayudado a aprender mucho sobre interfaces conversacionales mientras me divertía. Además, de paso, me ha servido para reforzar y ampliar los conocimientos obtenidos durante el ciclo, ya que ha sido muy completo, incluyendo acceso a datos, programación en lenguajes menos vistos en el curso como JavaScript y Node...

A nivel personal, creo que era una idea que se adapta a mí, ya que podía unir con ella la programación y la narración e, incluso, añadir música.

El tutorial principal que he seguido para su realización me ha ayudado mucho. Es la primera referencia en la bibliografía. Con él pude hacer un recorrido completo de la creación de una skill, aprendiendo de todo el proceso y pudiendo adaptarlo a mi proyecto.

Como mayor dificultad pondría la búsqueda de ejemplos, solución de problemas... ya que es una programación bastante específica y no hay mucho material ajeno al propio de la compañía. Aunque la documentación de Amazon es muy buena en general, a veces cuesta encontrar una respuesta que se adapte realmente a lo que necesitas.

6 BIBLIOGRAFÍA

Viscuso, G. Desarrolla tu Alexa Skill [en línea]. *Cursos Online de Programación | KeepCoding*. [Consultado el 20 de abril de 2022]. Disponible en: <https://plataforma.keepcoding.io/courses>

Juárez, M. R. (2021, 10 febrero). Historias interactivas, cuando te conviertes en el protagonista del cuento. *Canal Diseño y Arquitectura*. [Consultado el 3 de mayo de 2022] Disponible en: <https://revistadigital.inesem.es/diseno-y-artes-graficas/historias-interactivas/>

Sánchez, B. C. (2020, 9 noviembre). *Acertijos difíciles con respuesta*. [Consultado el 6 de mayo de 2022] Disponible en: <https://www.mundodeportivo.com/uncomo/ocio/articulo/acertijos-dificiles-con-respuesta-50602.html>

Fedeperin (2021). API Rest de Harry Potter. *GitHub*. [Consultado el 12 de mayo de 2022] Disponible en: <https://github.com/fedeperin/harry-potter-api>

Amazon, (2022). Alexa Skills Kit Sound Library | Alexa Skills Kit. Amazon (Alexa). [Consultado el 9 de junio de 2022] Disponible en:
<https://developer.amazon.com/en-US/docs/alexa/custom-skills/ask-soundlibrary.html>

Amazon, (2022). Speechcon Reference (Interjections): Spanish (ES) | Alexa Skills Kit [en línea]. *Amazon (Alexa)*. [Consultado el 9 de junio de 2022]. Disponible en: <https://developer.amazon.com/en-US/docs/alexa/custom-skills/speechcon-reference-interjections-spanish.html>

Amazon, (2022). Use Media Files with Your Alexa-Hosted Skill | Alexa Skills Kit [en línea]. *Amazon (Alexa)*. [Consultado el 9 de junio de 2022]. Disponible en: <https://developer.amazon.com/en-US/docs/alexa/hosted-skills/alexa-hosted-skills-media-files.html>

Amazon, (2022). Use DynamoDB for Data Persistence with Your Alexa-hosted Skill | Alexa Skills Kit [en línea]. *Amazon (Alexa)*. [Consultado el 9 de junio de 2022]. Disponible en: <https://developer.amazon.com/en-US/docs/alexa/hosted-skills/alexa-hosted-skills-session-persistence.html>

Dabble Lab, (2020). GitHub - dabblelab/alexa-hosted-s3-audio-example-skill: Alexa-Hosted S3 Audio Template [en línea]. *GitHub*. [Consultado el 9 de junio de 2022]. Disponible en: <https://github.com/dabblelab/alexa-hosted-s3-audio-example-skill>