Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

UVaInformer: Una aplicación conversacional basada en Alexa Skills©

Autor:
D. Álvaro González-Iglesias González

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

UVaInformer: Una aplicación conversacional basada en Alexa Skills©

Autor:
D. Álvaro González-Iglesias González

Tutor:
D. Valentín Cardeñoso Payo

Índice general

Índice de cuadros V

Índice de figuras VII

Resumen IX

Agradecimientos XI

1 Introducción 1
- 1.1 Motivación . . . 1
- 1.2 Objetivos . . . 1
- 1.3 Contenidos de la memoria . . . 2
- 1.4 Herramientas empleadas . . . 2

2 Metodología y Planificación 3
- 2.1 Metas . . . 3
- 2.2 Restricciones . . . 3
- 2.3 Metodología adoptada . . . 4
  - 2.3.1 Roles Scrum . . . 4
  - 2.3.2 Scrum adaptado . . . 4
- 2.4 Riesgos . . . 5
- 2.5 Matriz de impacto / probabilidad . . . 5
- 2.6 Seguimiento . . . 7
  - 2.6.1 Sprint 0: Estudio teórico . . . 7
  - 2.6.2 Sprint 1: Primera *skill* . . . 7
  - 2.6.3 Sprint 2: UVa Informer . . . 8

3 Fundamentos teóricos 9
- 3.1 Sistemas de Diálogo Hablado . . . 9
- 3.2 Arquitectura . . . 9
  - 3.2.1 Task-oriented dialogue agents . . . 10
  - 3.2.2 Chatbots . . . 12
- 3.3 Componentes esenciales . . . 14
  - 3.3.1 Reconocimiento de Habla . . . 14
  - 3.3.2 Comprensión del lenguaje . . . 16
  - 3.3.3 Gestión de Diálogo . . . 16
  - 3.3.4 Comunicación con el sistema externo . . . 16
  - 3.3.5 Generación de respuestas . . . 16
  - 3.3.6 Salida vocal . . . 17

4 Plataformas de desarrollo 19
- 4.1 Análisis de plataformas existentes. . . . 19
  - 4.1.1 Siri . . . 19
  - 4.1.2 Google Assistant . . . 19
  - 4.1.3 Cortana . . . 20
  - 4.1.4 Alexa . . . 20

5 Anatomía de una *skill* 21
5.1 Proceso de vida de una *skill* . . . 21
5.1.1 Reconocimiento vocal . . . 21
5.1.2 Procesamiento de datos . . . 22
5.2 Partes de una *skill* . . . 22
5.2.1 Wake word . . . 22
5.2.2 Inicialización . . . 22
5.2.3 Invocation Name . . . 23
5.2.4 Intents . . . 23
5.2.5 Slots . . . 23
5.2.6 Utterances . . . 24
5.3 Tipos de *skill* . . . 24
5.3.1 Custom . . . 24
5.3.2 Flash Briefing . . . 24
5.3.3 Smart Home . . . 25
5.3.4 Video . . . 25

6 Estudio de alternativas 27
6.1 Datos de interés . . . 27
6.2 Estudio de viabilidad . . . 27
6.2.1 Existencia y ubicación de libros . . . 27
6.2.2 Horario de tutorías . . . 28
6.2.3 Información básica . . . 28
6.2.4 Disponibilidad horaria . . . 28
6.2.5 Guías docentes . . . 29
6.3 Decisión final . . . 29
6.3.1 Datos . . . 29
6.3.2 Skill . . . 29

7 Desarrollo 31
7.1 Análisis y arquitectura . . . 31
7.1.1 Intent 1: Información de profesores . . . 31
7.1.2 Intent 2: Horarios de tutorías . . . 33
7.2 Diseño vocal . . . 37
7.2.1 Intent 1: Información de profesores . . . 37
7.2.2 Intent 2: Horarios de tutorías . . . 39
7.2.3 Arranque de *skill* . . . 40
7.2.4 Interacción con *skill* iniciada . . . 40
7.3 Implementación . . . 40
7.3.1 Archivos auxiliares . . . 40
7.3.2 Intent 1: Información de profesores . . . 41
7.3.3 Intent 2: Horarios de tutorías . . . 44
7.4 Pruebas . . . 48

8 Conclusiones y trabajo futuro 51
8.1 Conclusiones . . . 51
8.2 Líneas de trabajo futuro . . . 51

A Tutorial: Creación de una Alexa Skill 53
A.1 Build . . . 55
A.1.1 Custom . . . 56
A.1.2 Invocation . . . 56
A.1.3 Interaction Model . . . 56
A.1.4 Assets . . . 57
A.2 Code . . . 57
A.2.1 util.js . . . 58
A.2.2 package.json . . . 58
A.2.3 local-debugger.js . . . 58

A.2.4 index.js 58
A.3 test 59

B Tutorial: Ampliación de la skill **61**
B.1 Diseño vocal 61
B.2 Código 64

Bibliografía **67**

Índice de cuadros

2.1 Plantilla para los riesgos 5
2.2 Riesgo 1 5
2.3 Riesgo 2 5
2.4 Riesgo 3 6
2.5 Riesgo 4 6
2.6 Riesgo 5 6
2.7 Tabla de riesgos 6
2.8 Matriz de impacto / probabilidad 7

Índice de figuras

3.1 Modelo simple de una cadena oculta de Markov (McTear 2002)[1] 15

7.1 Diagrama conceptual de funcionamiento 32
7.2 Diagrama de secuencia de un intent cualquiera 32
7.3 Panel de búsqueda 33
7.4 Consola de desarrollo 34
7.5 Orden CURL sobre directorio UVA 34
7.6 Resultado web de una búsqueda en directorio UVA 35
7.7 Respuesta a la orden GET realizada sobre directorio UVA 35
7.8 Página web alfabética de la UVA como respuesta a petición GET 35
7.9 Página web de la Universidad de Valladolid con el listado de grados 36
7.10 Segmento en html que define el enlace al grado 36
7.11 Página web de la Universidad de Valladolid con el listado de horarios de tutorías 37
7.12 TeacherHandler: Código del manejador del intent Información de profesores . 41
7.13 setSlots: Código de la función encargada de extraer los slots 42
7.14 executeTeacher: Código de la función principal del intent de Información de profesores 43
7.15 CheckName: Código de la función encargar de generar el path para el intent de Información de profesores 43
7.16 getName: Código de la función encargar de realizar la petición GET a directorio uva. 44
7.17 HorariosHandler: Código del manejador del intent Horarios de tutorías. . . . 45
7.18 executeHorarios: Código de la función principal del intent de Horarios de tutorías. 46
7.19 getWebPage: Código de la función encargar de realizar la petición GET al dominio uva.es 47
7.20 searchStringInArray: Código de la función que busca una cadena de texto dentro de las cadenas de texto de un array de cadenas de texto 47
7.21 extractTimeTables: Código de la función que extrae la información de horarios de un profesor de la tabla general en formato html 48
7.22 timeTableJSON: Código de la función que da formato JSON a la información de horarios de profesores extraída de la tabla html 48

A.1 Interfaz principal de la consola de desarrollo de Alexa *skills* 53
A.2 Selección de nombre e idioma 54
A.3 Selección de tipo de *skill* 54
A.4 Selección de aprovisionamiento de backend 55
A.5 Selección de tipo de plantilla para el desarrollo de la *skill* 55
A.6 Barra de herramientas del modelo. 55
A.7 Pantalla de desarrollo de HelloWorldIntent 56
A.8 Pantalla de creación de un slot 57
A.9 Pantalla de definición de un slot 57
A.10 Opciones de guardado y despliegue. 58

A.11 Archivos predefinidos en la *skill*. 58
A.12 Activación de testeo 59

B.1 Selección del tipo de intent 61
B.2 Slots que hemos generado para el intent de Viajes 62
B.3 Estructura de las peticiones recibidas desde Alexa 65
B.4 Primera parte del código del manejador del intent Viajes 65
B.5 Segunda parte del código del manejador del intent Viajes 65
B.6 Tercera parte del código del manejador del intent Viajes 66
B.7 Cuarta parte del código del manejador del intent Viajes 66

Abstract

Desde la aparición de los asistentes virtuales a finales de los 90, los entornos de aplicación de este tipo de servicios son cada vez más amplios y dinámicos. La progresiva universalización de esta forma de interacción hace que aparezcan plataformas hardware y software que aspiran a facilitar el desarrollo de asistentes virtuales a perfiles de usuarios y desarrolladores no especializados, más allá del círculo de las grandes compañías en que se experimentó su uso al principio. De esta forma, han aparecido diversos kits de desarrollo de software (SDK) que acercan el desarrollo de este tipo de sistemas a desarrolladores que, si bien deben contar con conocimientos de programación generales, no necesariamente deben ser especialistas en el desarrollo de sistemas de procesamiento de lenguaje natural (NLP).

En este trabajo de fin de grado abordamos el análisis de algunas de las plataformas existente para el desarrollo de sistemas conversacionales, centrándonos en una de ellas (Alexa, de Amazon) para realizar un estudio más detallado de la misma y desarrollar con ella un sistema conversacional (Alexa *skill* en la terminología de Amazon) que permita ilustrar las posibilidades y limitaciones de esta tecnología para el desarrollo de interfaces avanzados persona computadora basados en sistemas conversacionales. Aunque la aplicación desarrollada podrá y deberá ser aún ampliada y mejorada antes de poder ser distribuida a potenciales usuarios finales, servirá de prueba de concepto y prototipo funcional básico para aproximarnos a las posibilidades de este tipo de sistemas. La aplicación desarrollada se ajusta a una plantilla típica de sistema de consulta de datos que, en este caso, se refiere a aspectos académicos relacionados con la oferta de asignaturas de la Universidad de Valladolid. Mientras su desarrollo nos ha permitido detectar los aspectos más complejos del desarrollo de este tipo de sistemas, confiamos en que la documentación del mismo aportada en esta memoria pueda servir de base para posteriores aplicaciones desarrolladas con esta tecnología.

Agradecimientos

Al amor de mi vida, Jennifer, por aguantarme durante tanto tiempo, por estar ahí en los buenos momentos y por ayudarme a superar los malos, por darme tu cariño y amor de manera incondicional y ser siempre mi soporte sin importar lo que pasase.

A mi tutor Valentín, por todo lo que he aprendido y por no desesperarse conmigo durante todo el proceso, aunque no te lo haya puesto precisamente fácil, y ayudarme con cualquier duda que me haya podido surgir.

Capítulo 1

Introducción

1.1. Motivación

La Importancia de entornos de interacción vocales está relacionada con la facilidad y naturalidad de uso que proporciona el modelo de interacción basado en diálogo. De esta manera, los usuarios pueden entablar una conversación y obtener la información deseada, imitando lo que sería una interacción típica cara a cara con una persona.

Actualmente la interacción entre personas y ordenadores se está transformando, más allá de los paradigmas clásicos basados en pantalla, teclado y ratón: sistemas de realidad virtual (gafas de realidad virtual) o de realidad aumentada (pantallas portátiles que muestran información añadida a la vista normal, o sistemas de información auditiva), y sistemas de interacción hablada, entre otros, que están imponiéndose en entornos domésticos, de ocio y profesionales.

Entre las ventajas de los asistentes vocales como paradigma de interacción, no solo debemos resaltar la naturalidad y facilidad de la interacción: los asistentes vocales también destacan por permitir un en escenarios de "manos libres": durante la conducción, caminando, corriendo o en cualquier tipo de desplazamiento, sin tener que distraer la atención visual y facilitando así aproximaciones multimodales al diseño y desarrollo de sistemas de interacción.

Con la aparición de los sistemas vocales y sus respectivas aplicaciones personalizadas, se ha detectado una carencia de éstas en los ámbitos educativos, más concretamente en las universidades, donde cada alumno cuenta con una gran cantidad de recursos (algunos de ellos descentralizados).

Con este proyecto se pretende poner una primera piedra para lo que podrá ser un asistente vocal (y posiblemente visual) completo para los alumnos de la Universidad de Valladolid.

1.2. Objetivos

En este trabajo de fin de grado vamos a intentar alcanzar una serie de objetivos:

- Comprender y describir la arquitectura y funcionamiento típico de los sistemas conversacionales.

- Desarrollar un prototipo funcional de una aplicación vocal, implementada sobre uno de los sistemas comerciales más utilizados actualmente.

- Dotar de funcionalidad efectiva al prototipo desarrollado para ser utilizado por el alumnado de la Universidad de Valladolid.

1.3. Contenidos de la memoria

Esta memoria está estructurada en ocho capítulos y dos apéndices, que cubren los siquientes aspectos:

Capítulo 1: Se presenta el proyecto planteado, su motivación y los objetivos que pretende alcanzar.

Capítulo 2: Planteamiento y seguimiento del proceso de gestión que seguirá el proyecto.

Capítulo 3: Exposición de conceptos teóricos sobre los sistemas conversacionales.

Capítulo 4: Análisis de las plataformas de desarrollo existentes en el mercado.

Capítulo 5: Descomposición y explicación de una Alexa *skill*

Capítulo 6: Conclusiones del estudio sobre las diversas alternativas de desarrollo

Capítulo 7: Diseño e implementación de la aplicación.

Capítulo 8: Conclusiones finales y propuestas de desarrollo.

Apéndice A: Guía paso a paso para la implementación de una custom *skill* a partir de una plantilla.

Apéndice B: Ejemplo de desarrollo de una custom *skill*.

Bibliografia

1.4. Herramientas empleadas

En este proyecto se han utilizado las siguientes herramientas y plataformas:

Plataforma de desarrollo

- *Alexa developer console* (Plataforma de desarrollo de Alexa): Herramienta de desarrollo interactiva que incluye las cuatro etapas básicas de desarrollo de las *skills* (build, code, test y deploy).
- Lenguaje Javascript: Para el desarrollo de las funcionalidades, así como la obtención de datos se ha elegido la variante basada en Javascript frente a las alternativas Python y Java (sólo para offline).

Preparación de la Memoria

- Overleaf: Editor on-line de LaTeXque permite almacenar y compartir los archivos en la nube, lo que facilita poder trabajar simultáneamente a varias personas desde distintos lugares.
- paint.net: Editor gráfico empleado en algunas de las imágenes presentes en la memoria.

Capítulo 2

Metodología y Planificación

En este capítulo se describe y justifica la metodología seguida en el desarrollo de la *skill*, así como metas que se desean alcanzar y los posibles problemas que surjan durante dicho desarrollo.

2.1. Metas

Este proyecto se ha desarrollado por una única persona, con todo lo que ello conlleva dado que esa misma persona deberá hacerse cargo de la gestión, dirección, desarrollo y pruebas de todo el proyecto, no obstante el modelo a seguir procurará imitar en la medida de lo posible un proyecto típico desarrollado por un equipo, intentando de esta manera alcanzar los mismos objetivos.

- Calidad: objetivo básico de cualquier proyecto que se precie, ligado directamente al desarrollo, no se considerará como producto finalizado si no cumple unos mínimos de calidad. Este objetivo implica la eliminación de fallos a lo largo de la vida del proyecto así como asegurarse del correcto funcionamiento del producto final.

- Productividad: Se procurará cumplir con el tiempo establecido para la entrega del producto.

- Reducción de riesgos: Con la intención de cumplir con los plazos, se desarrollará un plan de contingencia para cada riesgo que pueda surgir por el camino.

2.2. Restricciones

Factores que limitan el trabajo de desarrollo de la *skill*.

- Tiempo: el tiempo dado para el desarrollo de la *skill* solo puede extenderse hasta mediados de Octubre.

- Conocimientos: dado el carácter novedoso de la propuesta y la carencia de conocimientos previos, así como práctica en el desarrollo de este tipo de software, cualquier avance conlleva un estudio previo y por consiguiente un tiempo añadido.

- Equipo: como se ha descrito con anterioridad el trabajo será desarrollado por una única persona.

2.3. Metodología adoptada

Dado el "pequeño" tamaño del proyecto, así como el reducido equipo de trabajo, se ha adoptado una metodología ágil, más concretamente Scrum, ya que simplifican de manera significativa todo el proceso de gestión.

2.3.1. Roles Scrum

En la metodología Scrum intervienen principalmente 3 roles, que son los siguientes:

- Scrum Master: Responsable del cumplimiento de la metodología, dado que es la persona con mayor conocimiento sobre esta.
- Equipo de desarrollo: Personal encargado del desarrollo del producto (AKA: desarrolladores.)
- Product Owner: Máximo interesado en el proyecto, normalmente el cliente, también se encuentra definido este rol como representante de los stakeholders.

Aparte de los roles principales, la metodología Scrum está caracterizada por una serie de eventos, a saber:

- Scrum meeting: Reuniones, típicamente diarias, entre los miembros del equipo.
- Sprint: Marco temporal que marca el desarrollo de distintas partes del proyecto, típicamente predefinidos a una duración fija, y de los que se espera que al acabar se haya desarrollado una parte funcional del trabajo.
- Sprint review: Reunión que toma lugar al finalizar un sprint, se revisa el desarrollo del mismo y se comprueba si los objetivos prefijados se han alcanzado.
- Sprint retrospective: Reunión que también toma lugar al finalizar un sprint, pero que tiene como finalidad analizar los objetivos no cumplidos para poder subsanarlos y mejorarlos en el siguiente sprint.

2.3.2. Scrum adaptado

Como hemos dicho previamente, y tal y como viene reflejado en las restricciones la metodología utilizada se ha tenido que adaptar a las circunstancias.

Los roles se han visto modificados de la siguiente manera:

- Scrum Master/equipo de desarrollo: Se aúnan los 2 roles en uno, dado que los ejecutará la misma persona, Álvaro González-Iglesias González.
- Product Owner: En este caso no será un cliente, pero si la persona más interesada en el proyecto, el tutor del trabajo Valentín Cardeñoso Payo

Los eventos que definimos previamente, se van a ver también modificados.

Para empezar como es obvio, no se realizarán los Scrum *meetings*, dado que el equipo de desarrollo está compuesto por una única persona.

En cuanto a los *sprints*, se ha acordado estructurar el proyecto en 3 *sprints*, con una duración estimada de 1 mes por sprint (englobando de esta manera los 3 meses de desarrollo de proyecto).

Añadido a estos *sprints* de 1 mes, se realizarán reuniones de toma de contacto cada 1/2 semanas, en función de la disponibilidad tanto por parte del Product Owner, como del equipo de desarrollo (esto es debido a que coincide con el periodo vacacional, y otra serie de obligaciones, que se detallarán más adelante en los riesgos).

2.4. Riesgos

Debido a que existen diversos factores que pueden tener un impacto negativo en el tiempo de desarrollo del proyecto, es importante realizar un estudio previo para conocer dichos factores y poder tomar medidas en el caso de que sucedan.

Este estudio nos permitirá identificar, definir, y planificar los riesgos que puedan surgir, y para ello nos serviremos de la siguiente plantilla:

Identificador	Identificador único del riesgo
Descripción	Descripción del riesgo
Probabilidad	Estimación de la probabilidad de ocurrencia del riesgo
Consecuencias	Consecuencias que puede tener la ocurrencia del riesgo
Impacto	Nivel de impacto que tendría en el proyecto el riesgo
Plan de contingencia	Medidas a adoptar para minimizar el impacto en caso de ocurrir el riesgo
Comentarios	Notas sobre el riesgo

Cuadro 2.1: Plantilla para los riesgos

A continuación se enumeran los riesgos que se han detectado en el estudio:

Identificador	1
Descripción	Contracción de enfermedad
Probabilidad	Media
Consecuencias	Bajar o parar el ritmo de trabajo
Impacto	Medio
Plan de contingencia	No existe un plan de actuación para este riesgo ya que una vez sucedido no es posible establecer una contramedida adecuada
Comentarios	Habitualmente la probabilidad de ocurrencia es baja, pero debido a la pandemia se ha elevado a media

Cuadro 2.2: Riesgo 1

Identificador	2
Descripción	Imposibilidad de desarrollo de las ideas predefinidas
Probabilidad	Alta
Consecuencias	Replantear el objetivo de desarrollo
Impacto	Medio
Plan de contingencia	Establecer varias vías de desarrollo para poder cambiar en el caso de que alguna de estas acabe en fracaso
Comentarios	De hacerse un buen estudio previo de las posibilidades el impacto podrá cambiar de medio a bajo

Cuadro 2.3: Riesgo 2

2.5. Matriz de impacto / probabilidad

En la siguiente tabla se resumen los riesgos planteados previamente, y además se indica una estimación de la probabilidad de la ocurrencia de cada riesgo, así como una clasificación de su impacto.

Identificador	3
Descripción	Retrasos en el desarrollo
Probabilidad	Media
Consecuencias	No alcanzar los objetivos de desarrollo en los tiempos estipulados
Impacto	Muy Alto
Plan de contingencia	Aumentar el número de horas diarias de desarrollo
Comentarios	Debido a la pandemia la probabilidad se ha aumentado, ya que las circunstancias propician que puedan surgir distracciones o problemas

Cuadro 2.4: Riesgo 3

Identificador	4
Descripción	Imposibilidad de acceso a los servicios
Probabilidad	Baja
Consecuencias	No poder realizar parte del desarrollo
Impacto	Alto
Plan de contingencia	Dividir el desarrollo en secciones on-line y off-line, lo que permitirá seguir con la parte off-line mientras los servicios se restablecen
Comentarios	Debido a que parte del desarrollo debe hacerse en plataformas on-line existe la posibilidad de que dicho servicio se vea interrumpido por tareas de mantenimiento

Cuadro 2.5: Riesgo 4

Identificador	5
Descripción	Necesidad excesiva de estudio
Probabilidad	Alta
Consecuencias	El tiempo previo al desarrollo se aumenta, por lo que incurrimos en una necesidad de aumentar las horas del mismo posteriormente
Impacto	Muy Alto
Plan de contingencia	Intentar establecer objetivos más al alcance, y redirigir el desarrollo acorde a las capacidades del equipo
Comentarios	El carácter novedoso del proyecto causa una necesidad muy alta de estudio sobre el mismo

Cuadro 2.6: Riesgo 5

Identificador	**Riesgo**	**Impacto**	**Probabilidad**
1	Contracción de enfermedad	2	33 %
2	Imposibilidad de desarrollo de las ideas predefinidas	2	45 %
3	Retrasos en el desarrollo	4	25 %
4	Imposibilidad de acceso a los servicios	3	10 %
5	Necesidad excesiva de estudio	4	60 %

Cuadro 2.7: Tabla de riesgos

Partiendo de la tabla anterior se ha podido desarrollar la siguiente matriz de riesgos

Impacto/probabilidad	Muy baja	Baja	Media	Alta	Muy alta
Despreciable					
Marginal		1	2		
Crítico	4				
Catastrófico		3		5	

Cuadro 2.8: Matriz de impacto / probabilidad

2.6. Seguimiento

A continuación se ha recogido brevemente el seguimiento real de los *sprints* realizados, así como el contenido abordado/desarrollado entre reuniones.

2.6.1. Sprint 0: Estudio teórico

El primer sprint se ha planteado como una fase de estudio sobre las tecnologías relevantes para el proyecto, tanto a nivel teórico como las implementaciones comerciales más actuales.

Semanas 1 y 2

Respecto al estudio teórico, se ha procedido a investigar sobre el funcionamiento general de los sistemas de dialogo hablado como se ha abordado en el **Capítulo 3**

Semanas 3 y 4 (1ª mitad)

Se finalizó el estudio teórico.

Una vez realizado el estudio de los fundamentos teóricos se procedió al análisis de las tecnologías comerciales existentes como se detalla en el **Capítulo 4**

De este estudio se concluyo que la plataforma idónea para el proyecto que se plantea es Alexa de Amazon

Semanas 4 (2ª mitad) y 5

Una vez decidida la plataforma sobre la que se iba a desarrollar el proyecto, se realizo un estudio exhaustivo de su funcionamiento como se puede ver en el Capítulo 5

Con la plataforma decidida y conociendo las capacidades de la misma se planteo, analizó y eligió el contenido del prototipo a desarrollar, como se ve en el Capítulo 6

2.6.2. Sprint 1: Primera *skill*

Con el estudio realizado y las decisiones sobre el contenido del prototipo tomadas, el sprint 1 se plantea como fase de toma de contacto con la plataforma de desarrollo y comprobación real de las capacidades tanto del desarrollador como de la plataforma.

Semanas 6 y 7

La primera toma de contacto se realiza siguiendo la documentación aportada por la propia plataforma de desarrollo, creando una skill de alexa a partir de una plantilla y siguiendo una guía.

Una vez entendido el funcionamiento de la plataforma de desarrollo, se realizaron más pruebas no abarcadas en la documentación propia de la plataforma, con el fin de encontrar el límite de las capacidades de la misma.

2.6.3. Sprint 2: UVa Informer

El último sprint se ha dedicado al desarrollo de la aplicación planteada, así como de la memoria actual.

Dado que el tiempo que se había planteado en un principio para la toma de contacto con la plataforma de desarrollo (el sprint 1 completo, aproximadamente 4 semanas), en la práctica a resultado menor, se ha procedido a iniciar el desarrollo del prototipo con antelación.

Semanas 8 y 9

Inicio del proceso de desarrollo del prototipo, diseño de la interfaz vocal y funciones auxiliares del código.

Semana 10

Se ha encontrado un problema en la conexión de datos entre Alexa (y el entorno de aws) y las páginas web de la Universidad de Valladolid, debido a un problema de Cors (Cross-Origin Resource Sharing) que impedía establecer cualquier tipo de conexión entre ambas.

Se consiguió solucionar dicho problema y continuar con el desarrollo.

Semana 11

Se finaliza el prototipo funcional de la Alexa *skill* y se realiza una batería de pruebas.

Semana 12

Elaboración de la memoria del proyecto.

Capítulo 3

Fundamentos teóricos

Desde la antigüedad el ser humano a tratado de interactuar con seres artificiales de manera natural,prueba de ello la encontramos en la mitología (Los autómatas de Hefesto, Talos, Golem,...) pero no es hasta el siglo diecinueve cuando se logra realmente los primeros avances en el campo.

A principios del siglo veinte se desarrollaron los primeros sistemas eléctricos capaces de reproducir cualquier sonido, seguido por las primeras computadoras en los años cuarenta, y es ya en la década de los sesenta cuando se desarrollan los primeros sistemas basados en lenguaje como por ejemplo ELIZA [2] que operaba con una entrada por teclado.

Con el paso de los años y la evolución tecnológica aparecen los primeros proyectos en el campo del reconocimiento vocal como es el proyecto ESPRIT SUNDIAL[3] o DARPA Spoken Language System (SLS) [4]

3.1. Sistemas de Diálogo Hablado

Entendemos como sistema de diálogo hablado (Spoken Dialogue System [SDS])/agente conversacional a cualquier interfaz informática con la que interactuamos mediante el lenguaje natural hablado, siguiendo un orden por turnos entre quien utiliza dicho sistema de dialogo, y la propia maquina.

En base a su funcionamiento podemos clasificar los SDS en dos grandes grupos[5]:

Task-oriented dialogue agents: Este tipo de sistema se enfoca en la realización de tareas concretas, desde crear eventos en un calendario, realizar llamadas, obtener información concreta o realizar reservas en diversos servicios. Necesita de respuestas muy concretas y enfocadas a la tarea que se desea resolver, y suelen desarrollarse dentro de un marco de acción muy bien definido.

Chatbots: Estos sistemas están especializados en alcanzar conversaciones más largas imitando “chats” reales entre personas, en las que el usuario podría estar interactuando con otro usuario en lugar de con una máquina. Se puede deducir por lo tanto que su ámbito se asimila más al del entretenimiento que al de solución de problemas.

3.2. Arquitectura

Dentro de cada gran clase existen diferentes arquitecturas que permiten alcanzar el objetivo de funcionamiento. Comenzaremos con los sistemas orientados a tareas.

3.2.1. Task-oriented dialogue agents

Finite state/graph - based

Como su propio nombre indica, esta arquitectura se constituye por una serie de pasos o estados que conforman una estructura cerrada por la que el usuario navega en función de sus respuestas.

El sistema dirige la conversación preguntando al usuario y recogiendo sus respuestas. Analiza dichas respuestas y las acepta o rechaza pasando así a un nuevo estado.

Esto quiere decir que el sistema cuenta con una serie de preguntas fijas que esperan una respuesta valida predefinida. Dado que el orden de las preguntas o las respuestas esperadas están fijadas de antemano, el usuario pierde la posibilidad de tomar la iniciativa en la conversación u obtener más información de la que se ha preestablecido, así mismo cualquier respuesta fallida, ya sea por no haberse emitido de manera correcta (errores ortográficos o de pronunciación), o por haberse reconocido de manera incorrecta (esto puede deberse a agentes externos a la interacción como pueda ser el ruido ambiente) caen en la ruta de fallo, forzando al usuario a alcanzar de nuevo el estado en el que se encontraba o reiniciando la interacción en el peor de los casos.

Este problema se puede reducir o solucionar si al sistema le añadimos una memoria sobre las respuestas correctas emitidas, o un componente de procesamiento del lenguaje natural, aunque la opción más sencilla y habitual suele ser una confirmación sobre todas las respuestas emitidas por el usuario previo cambio de estado en el grafo.

Esta arquitectura la encontramos normalmente en sistemas de reserva, venta de billetes o similar, dado que su limitación tanto en las respuestas como en el ámbito de actuación permiten que el desarrollo no sea muy costoso.

Un ejemplo de interacción con esta arquitectura sería el siguiente:

Sistema: ¿A dónde desea viajar ?
Usuario: Madrid.
Sistema: ¿Ha dicho Madrid?
Usuario: Si.
Sistema: ¿Qué día desea viajar?
Usuario: Martes.
Sistema: ¿Ha dicho miércoles?
Usuario: No
Sistema: ¿Qué día desea viajar?
...

frame/template - based

Esta arquitectura utiliza el concepto de plantilla para obtener la información del usuario. El sistema parte de una plantilla en la que encuentra distintos “huecos” (slots) que deberá rellenar para poder llevar a cabo la tarea, para ello realiza preguntas al usuario y extrae dicha información de las respuestas dadas, si el usuario no aporta todas la información necesaria en sus respuestas el sistema pasa a realizar preguntas más concretas sobre estas. A diferencia de la arquitectura previa el usuario goza de mayor libertad en sus respuestas, tanto en contenido como en longitud o complejidad, ya que es el sistema el encargado de identificar que parte de la respuesta ofrecida contiene la información que su plantilla necesita.

En este tipo de sistemas el rumbo de la conversación no está predefinido, depende completamente de la cantidad de información que el usuario es capaz de aportar en sus respuestas. Un ejemplo de interacción con esta arquitectura sería el siguiente:

Sistema: ¿A dónde desea viajar?
Usuario: A Madrid
Sistema: ¿Qué día desea viajar?
Usuario: El martes 27.
...

Aunque la conversación se asemeja a la arquitectura previa, podemos observar una serie de detalles que marcan la diferencia, como pueda ser el hecho de que la comprobación de errores se puede posponer, las respuestas admiten cierta variación o que una respuesta pueda ampliar la información solicitada.

Sobre esta misma conversación podríamos responder también de la siguiente manera:

Sistema: ¿A dónde desea viajar?
Usuario: A Madrid el martes 27 por la tarde.
Sistema: Aquí tiene los siguientes horarios de tarde ....
...

Como podemos observar se ha proporcionado todas la información necesaria en una única respuesta, lo que permite al sistema proceder directamente con la tarea que corresponde.

Para poder detectar y corregir errores en las respuestas proporcionadas es necesario que se utilice procesamiento del lenguaje natural, aunque en estos sistemas realmente valdría con ser capaces de reconocer la información útil que se necesita para completar los huecos de la plantilla.

dialogue-state

Entendida como una mejora de la arquitectura anterior, esta nueva arquitectura utiliza técnicas de aprendizaje automático para lograr el mismo objetivo.

A la hora de operar, la arquitectura anterior parte de una pregunta no muy específica para iniciar la conversación y obtener información, a partir de ahí en función de los datos que no ha conseguido obtener procede a realizar preguntas más específicas (en la mayoría de los casos hueco por hueco) para completar dicha información, una vez ha obtenido la información la procesa y rellena una frase predefinida para ofrecer la respuesta. En la nueva arquitectura se introduce un nuevo *componente* el acto de dialogo (dialogue-act). Este nuevo componente representa la intención del dialogo (la intención del dialogo nos indica que objetivo tiene la sentencia: saludo, información, pregunta, confirmación, selección, etc...) actual y los huecos detectados. Un ejemplo de dialogue-acts obtenidos de una conversación podría ser el siguiente:

	Sentencia	**Dialogue-act**
Usuario:	Hola, tengo hambre.	hello(task=find, type=restaurant)
Sistema:	¿Qué tipo de comida te apetece?	confreq(type=restaurant, food)
Usuario:	Me apetece sushi por aquí cerca	inform(food=japanese, location=near)
Sistema:	Shushipon tiene una buena crítica y se encuentra en el mismo barrio	inform(name=”Shushipon”, type=restaurant, location=neighborhood, critic= good)
Usuario:	¿Es barato?	confirm(price=low)
Sistema:	No, está en el rango medio de precio	negate(price=moderate)
Usuario:	¿Cuál es su página web?	request(webpage)
Sistema:	La página web de shushipon es www.shushipon.com	inform(name=”Shushipon”, webpage=”www.shushipon.com”)
Usuairo:	Muchas gracias, adios.	bye()

A parte de los dialogue-acts, en la nueva arquitectura se utilizan métodos de aprendizaje automática para detectar los datos necesarios para los huecos, así como el campo en el que se enmarca la conversación y su intención, esto quiere decir que podemos detectar si estamos hablando de una reserva de un vuelo (campo: aviación, intención: reserva) o una consulta sobre horarios de un restaurante (campo: hostelería, intención: consulta).

Como ahora contamos con información sobre la intención que tiene cada turno en la conversación, podemos cambiar el comportamiento que va a tener el sistema en función de esta, de esta manera, en lugar de simplemente realizar una batería de preguntas hasta haber rellenado todos los huecos de la plantilla, podemos realizar confirmaciones si el numero de datos aportados en una misma sentencia es demasiado alto, o realizar preguntas con la intención de rellenar varios huecos a la vez.

Como hemos destacado esta es una versión actualizada de la arquitectura basada en marcos, y gracias a las técnicas de aprendizaje automático podemos adaptarnos mejor a la hora de generar las sentencias de salida, de manera que sean más “humanas” ya que no dependemos de una plantilla de texto, si no que podemos realizar un análisis de la información solicitada, así como del ámbito e intención en el que se ha desarrollado la conversación para que esta resulte más fluida.

3.2.2. Chatbots

A diferencia de los sistemas enfocados en tareas, las diferentes arquitecturas de los chatbots están marcadas por como se adaptan al lenguaje del usuario e interactuan con este, en lugar de como pueden obtener mejor la información ya que no pretenden resolver una tarea.

rule-based

La arquitectura basada en reglas tiene su origen en el sistema ELIZA[2], ELIZA es un chatbot que pretende simular la psicología rogeriana en la cual asumimos la posición de no saber prácticamente nada sobre el mundo real, lo que permite a la otra parte explicar su punto de vista sin que nosotros partamos de una idea polarizada.

El sistema de reglas de ELIZA analiza una sentencia de entrada en función de las palabras que la componen, buscando palabras que puedan resultar más relevantes y aplicando reglas asociadas a dichas palabras. Para entenderlo mejor, existe un ranking de palabras en función de su importancia en una oración, de manera que palabras más especificas tienen un valor más alto en el ranking, mientras que palabras más genéricas tendrán un valor menor. Unidas a estas palabras se definen reglas que transforman la oración recibida en una sentencia de respuesta.

Si seguimos el ejemplo de Joseph Wezenbaum sobre ELIZA, este asemeja su comportamiento al de una persona que no entiende el idioma hablado por completo, y solo es capaz de extraer palabras sueltas de las sentencias que escucha, y a partir de estas genera sus respuestas.

Tomamos como ejemplo la frase: **Parece que tú me odias**

El sistema es capaz de fragmentar en cuatro piezas la sentencia.

1	2	3	4
Parece que	tú	me	odias

Suponemos que la persona que no entiende el idioma del todo solo conoce el significado de las palabras “tú” y “me”, por lo que es capaz de traducir dichas palabras (“tú” a “yo” y “me” a “te”) y utilizar una formula aprendida para componer la frase: **¿Qué te hace pensar que yo te odio?**

Utilizando una notación más formal la descomposición de la formula utilizadas sería:

(0 TÚ ME 0)
Y la regla aplicada a las palabras encontradas sería:
(¿QUÉ TE HACE PENSAR QUE YO TE 4)

Donde el 0 en la descomposición hace referencia a un número indefinido de palabras, y el 4 en la composición hace referencia al cuarto componente de la descomposición.

corpus-based

A diferencia de la arquitectura anterior los chatbots basados en corpus de texto en lugar de utilizar un sistema de reglas, procesan gran cantidad de conversaciones entre personas para encontrar la respuesta más adecuada.

Estos sistemas se fundamentan en los datos aportados para su entrenamiento, que típicamente son conversaciones entre personas, ya sean chats de texto, de voz, diálogos de películas, de series o interacciones en las redes sociales.

Similar al sistema basado en reglas, los chatbots basados en corpus ignoran en gran medida el contexto previo de la conversación buscando generar una respuesta sobre la última sentencia introducida por el usuario.

Una vez se han cargado los corpus de texto en el sistema existen dos clases principales de modelos de generación de respuestas:

information retrieval

Los chatbots basados en la recuperación de información se caracterizan por responder al turno de conversación X de un usuario con el correspondiente turno Y de uno de los corpus de texto definidos anteriormente.

Para hallar que turno Y es el más apropiado los sistemas basados en la recuperación de información pueden utilizar cualquier tipo de algoritmo que se plantee, como ejemplo vamos a describir brevemente dos de los más simples.

1. Devolver la respuesta al turno más similar: Dada una sentencia p y un corpus de texto C, debemos buscar el turno t en C que es más similar a p, y devolvemos el turno siguiente.

2. Responder con el turno más similar: Equivalente al anterior, solo que en esta ocasión devolvemos t en lugar de su respuesta, ya que una buena respuesta muchas veces comparte palabras o estructuras semánticas con la sentencia a la que se pretende responder.

encoder-decoder

Esta es la más moderna de las arquitecturas y tal vez la que plantee un aspecto más novedoso, ya que en lugar de realizar una búsqueda directa en el corpus de texto se asemeja más a un sistema de reglas. La arquitectura de codificador-decodificador plantea que la obtención de una respuesta se trata de un trabajo de *transducción* de la pregunta, como si aplicásemos machine learning al sistema de reglas, mediante esta arquitectura el sistema aprende gracias al corpus suministrado a transducir una respuesta a partir de la pregunta.

Inicialmente se propuso la idea de adaptar los sistemas de traducción automática de frases [6] para generar respuestas en base al turno previo suministrado, pero más tarde se observo como hemos mencionado que las respuestas no siempre comparten contenido con las

preguntas, de esta manera se propuso aplicar modelos de transducción [7] generados a través de modelos de codificación-decodificación.

3.3. Componentes esenciales

Existen una serie de componentes eseciales comunes a todos los SDS, ya que independientemente de su arquitectura u objetivo todos deben realizar una serie de pasos comunes para poder funcionar: Recoger la información, traducirla, procesarla, generar la respuesta y emitirla para que el usuario la reciba.

A continuación vamos a detallar brevemente en que consisten dichos componentes.

3.3.1. Reconocimiento de Habla

Este componente se encarga de convertir la sentencia pronunciada por un usuario (como una secuencia de sonidos) en una lista de palabras que se puedan procesar.

A la hora de realizar esta conversión hay que tener en cuenta que existen un gran número de variables que intervienen y dificultan la identificación de fonemas o palabras:

Variación lingüística: Efectos sobre la señal acústica causados por diversos fenómenos lingüísticos

Distintos hablantes: Debido a las características de cada hablante como puedan ser la edad, sexo, morfología vocal, etc... se producen cambios en las ondas acústicas que aunque pronunciando el mismo fonema su representación en forma de onda difiere.

Canal de emisión: La presencia de ruido de fondo, o el sistema de recepción (un micrófono, un teléfono,...) generan sonidos no deseados.

Tamaño del vocabulario: Distintos sistemas juegan con distintos tamaños de vocabulario, ya que podemos implementar desde sistemas que solo admitan un grupo reducido de palabras, a sistemas conversacionales que admitan todas las palabras conocidas como un chatbot.

Para poder realizar su función el reconocedor de voz se modela como un proceso estocástico en el cual el sistema trata de encontrar el fonema/palabra/frase $\hat{W}$ de todos los posibles fonemas/palabras/sentencias del lenguaje L que tiene la mayor probabilidad de coincidir con lo emitido por el usuario X.

$$\hat{W} = \underset{W \in L}{argmax} P(W|X) \tag{3.1}$$

Donde la función argmax devuelve el valor más alto de su argumento $P(W|X)$. W es una secuencia de símbolos $W = w_1, w_2, w_3, ..., w_n$ en el modelo acústico, y X es una secuencia de segmentos de la entrada $X = x_1, x_2, x_3, ..., x_n$. Esta ecuación nos da la probabilidad de la cadena W dada la entrada X. Como no podemos obtener directamente $P(W|X)$ aplicamos el teorema de Bayes:

$$\hat{W} = \underset{W \in L}{argmax} \frac{P(X|W)P(W)}{P(X)} \tag{3.2}$$

donde $P(X)$ puede ser ignorado ya que la entrada va a permanecer constante, por lo que simplificando:

$$\hat{W} = \underset{W \in L}{argmax} P(X|W)P(X) \tag{3.3}$$

En esta última ecuación $P(X|W)$ es la probabilidad observada, mientras que $P(X)$ es la probabilidad a priori. $P(X|W)$ representa el modelo acústico, y $P(X)$ representa el modelo de lenguaje.

El **modelado acústico** consiste en mapear la señal de voz continua a los sonidos individuales de las palabra. El modelo acústico de una palabra se representa en cadenas de markov ocultas (Hidden Markov Models [HMM]).

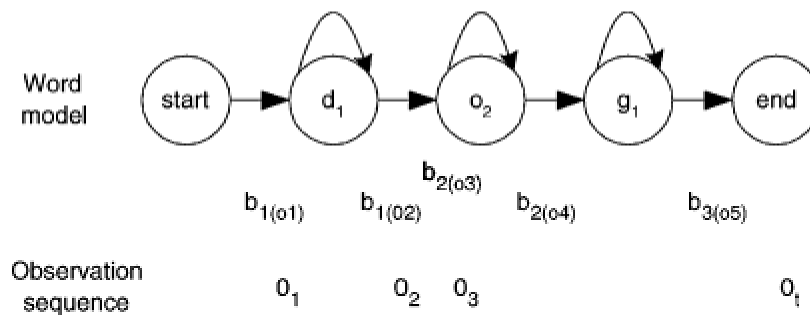

Figura 3.1: Modelo simple de una cadena oculta de Markov (McTear 2002)[1]

En el modelo de la figura 3.1 cada estado representa un fonema de la palabra "dog", mientras que cada transición marca la probabilidad de cambiar de un estado al siguiente. Debido a las distintas duraciones de cada sonido, un mismo sonido puede prolongarse a lo largo de varias unidades de tiempo, por lo que los estados pueden volver a sí mismos para mantenerse en el mismo estado.

Una HMM es un autómata de estado finito determinista consistente en estados y transiciones entre estados, como también es doblemente estocástica, ya que a parte de las probabilidades de transición, la salida de cada estado es también probabilística.

La entrada de una HMM es una secuencia de vectores de características observadas $o_1, o_2, o_3, ..., o_n$ y se busca encontrar la mejor secuencia de estados que haya podido producir la entrada, cada estado en la HMM tiene una distribución de probabilidad de posibles salidas $(b_1(o_1), b_1(o_2), b_2(o_3), ...)$.

Mientras el sistema procesa de izquierda a derecha desde el estado s_i al estado s_{i+1} el sonido en s_i se compara con un vector de rasgos o_t y se devuelve la salida con mayor probabilidad. Como hay una distribución de probabilidad de salidas en cada estado y cada salida puede ocurrir en cualquier estado, es posible obtener salidas del modelo, pero es imposible conocer de que estado es cada salida, por esta razón el modelo es "oculto".

Dado que la salida del modelo acústico es una suposición de palabras, podemos observar estas y elegir la palabra (como secuencia de sonidos detectados) que mejor se ajusta utilizando el modelo de lenguaje $P(X)$.

El **modelo de lenguaje** informa de que palabras tienen un resultado más probable dada una secuencia de letras o fonemas, y para ello existen dos posibilidades.

Si definimos de antemano todas las frases que se esperan como entrada, podremos utilizar una **red de estados finita**, ya que predice todas las posibles secuencias de palabras en el modelo de lenguaje.

El problema estriba en que sentencias que son correctas a efectos de significado o intención, si no están definidas de antemano, serán rechazadas.

Otra opción con la que contamos para poder tener en cuenta todas las sentencias correctas sin necesidad de definirlas con antelación sería utilizar **modelos de n-grama**, ya que los modelos de n-grama permiten calcular la probabilidad de una secuencia de palabras como el producto de las probabilidades de cada palabra, asumiendo que la ocurrencia de cada palabra viene dada por las $N-1$ palabras previas.

$$P(W) = P(w_1, ..., w_n) = \prod_{n=1}^{N} P(w_n | w_1, ..., w_{n-1}) \quad (3.4)$$

3.3.2. Comprensión del lenguaje

Una vez hemos transcrito la entrada proporcionada por el usuario el siguiente paso consiste en entender que hemos recibido, en este componente entran en acción las diferentes arquitecturas que hemos explicado anteriormente.

3.3.3. Gestión de Diálogo

Con la entrada transcrita y procesada, debemos decidir como actuar. Al igual que con el componente anterior, el funcionamiento de este componente se ha explicado en la sección previa de arquitectura.

3.3.4. Comunicación con el sistema externo

Por lo general los SDS necesitan conectarse con alguna fuente de información (tipicamente una base de datos) y para ello existen diversas formas de conseguirlo. Este componente es el que se encarga precisamente de ello, la comunicación con fuentes de información, y para ello no existe una estrategia, metodología definida, dependiendo de como funcione nuestro SDS y la fuente de información a la que queramos engancharnos, deberemos adaptar un puente de comunicación u otro.

El uso más habitual que se le da a este componente es el de obtención de los datos solicitados por el usuario para componer la respuesta, con los datos que se le han solicitado por ejemplo en un sistema de reserva de vuelos, se generará una query para obtener los datos de los vuelos que se enmarquen en sus preferencias, para poder solicitar al usuario que escoja el que necesita.

3.3.5. Generación de respuestas

Este componente aunque tiene la misma finalidad tanto para los chatbots como para los agentes orientados a tareas, funciona de distinta manera. Aunque dentro de cada apartado ya se definió su funcionamiento vamos a recordarlo brevemente.

Task-oriented dialogue agents: Lo más habitual es que contemos con una plantilla de respuesta que se rellenará con los datos obtenidos en el componente anterior, aunque si bien es verdad, podemos incluir procesamiento con aprendizaje automático para poder generar respuestas más parecidas al lenguaje humano y que se adapten mejor a las características particulares de cada sentencia de entrada.

Chatbots: Los chatbots como hemos reiterado ya en varias ocasiones, intentan imitar una interacción humana lo más creíble posible, por y para ello se valen de técnicas de aprendizaje automático entrenadas sobre conversaciones reales para lograrlo, de esta manera adaptan cada salida hacia el usuario de manera particular. Como hemos dicho el funcionamiento más preciso ya se ha explicado en secciones previas.

3.3.6. Salida vocal

Una vez ya tenemos toda la información que queremos devolver, y hemos generado la estructura de respuesta, solo nos queda pasar dicha sentencia al medio hablado. Una de las maneras más simple consiste en almacenar grabaciones de voz de personas, consistentes en todas las palabras necesarias para generar las distintas respuestas, y después componer palabra a palabra (grabación a grabación) la sentencia completa.

Un sistema más avanzado sería grabar cada posible fonema para poder componer cualquier tipo de palabra posible y no estar limitados por el contexto (tanto a la hora de grabar como de componer).

Como se puede observar existen múltiples de técnicas de composición de voz, y con el avance de la tecnología se ha ido superando la barrera de lo “robótico” para llegar a sistemas de voz más naturales y creíbles.

Capítulo 4

Plataformas de desarrollo

A continuación se va a proceder a explicar una serie de conceptos necesarios para entender el desarrollo de la *skill*.

Comenzaremos con un breve análisis de las diversas plataformas vocales del mercado, siguiendo con los pasos típicos en el desarrollo de una aplicación vocal.

Por último enunciaremos las herramientas que se han utilizado a lo largo del proyecto.

4.1. Análisis de plataformas existentes.

Las principales plataformas que existen actualmente en desarrollo son las pertenecientes a las grandes compañías, como son: "Alexa"[8] de Amazon, "Google Assistant"[9] de Google, "Cortana"[10] de Microsoft, "Siri"[11] de Apple.

4.1.1. Siri

Siri es considerado el primero de los asistentes virtuales comerciales, desarrollado actualmente por la empresa Apple, nació como una aplicación comercial del proyecto CALO. CALO fue un proyecto de DARPA que buscaba crear un sistema autónomo de análisis y clasificación de la información (CALO: Cognitive Assistant that Learns and Organizes), antes de que este fuese abandonado, 3 investigadores de SRI International crearon los primeros primeros prototipos de Siri, la cual estaba pensada para ser integrada todas las plataformas móviles, pero tras ser adquirida por Apple pasó a ser de uso exclusivo para sus sistemas móviles.

El planteamiento original de Siri consistía en ayudar a los desarrolladores a la hora de generar ontologías para los SDS.

Después de ser adquirida por Apple, el enfoque de su desarrollo comenzó a cambiar, pasando de ser un sistema de reglas puro a convertirse en un híbrido entre chatbot y sistema de reglas. Siri sigue funcionando como asistente que devuelve la información solicitada o ejecuta tareas siguiendo una arquitectura basada en marcos, pero también es capaz de gestionar solicitudes que caen fuera de los dominios preestablecidos, Gracias a esto consiguió adaptarse mejor a cualquier tipo de usuario.

4.1.2. Google Assistant

Google Assistant (sucesor de Google Now) es un asistente virtual desarrollado principalmente para dispositivos móviles y dispositivos de domótica inteligente.
A diferencia de su predecesor Google Assistant es capaz de participar en conversaciones bidireccionales, así como interactuar con diversas aplicaciones o dispositivos del hogar.
En sus últimas versiones se ha podido ver como Google Assistant a evolucionado su capacidad conversacional llegando a cotas de autonomía tales como poder concertar una cita en una peluquería vía telefónica con un interlocutor humano[12].

4.1.3. Cortana

Cortana nació como el asistente virtual de la compañía Microsoft desarrollado para los sistemas operativos Windows y Windows phone, aunque debido a la descontinuación de este último su desarrollo se ha orientado a nuevos objetivos[13].
La principal vía de desarrollo de Cortana está orientada a la suite de Office 365 de Microsoft ofreciendo una capacidad de reconocimiento del lenguaje que le permite responder a preguntas sobre los datos que la suite maneja, aunque no por ello carece de las funcionalidades más comunes en los asistentes virtuales, como son la capacidad de establecer recordatorios y notas, así como interactuar con diversas aplicaciones, si bien es verdad que su rango de acción es menor en este aspecto en comparación con los demás asistentes mencionados.

4.1.4. Alexa

Amazon define Alexa como "el servicio de voz ubicado en la nube de Amazon disponible en los dispositivos de Amazon y dispositivos de terceros con Alexa integrada.[8]"
Inicialmente Alexa solo se encontraba disponible en los dispositivos Echo (Exclusivos de Amazon), pero posteriormente se expandió su desarrollo a dispositivos móviles, televisores, manos-libres para vehículos...
Alexa se caracteriza por sus amplias capacidades de interacción con diversas aplicaciones, desde la reproducción de música a la creación de alarmas, listas de la compra, etc... También es reconocida por su facilidad de desarrollo por parte de usuarios menos expertos gracias a su plataforma de desarrollo, y la amplia documentación existente.

Capítulo 5

Anatomía de una *skill*

Para poder entender las decisiones que se tomarán posteriormente en el análisis es necesario entender y conocer como funciona y se desarrolla una *skill* para Alexa.

5.1. Proceso de vida de una *skill*

Para comenzar a entender el proceso de desarrollo de una *skill* de amazon tenemos que conocer las distintas partes que componen dicho proceso. Por un lado tenemos la parte de reconocimiento vocal y respuesta y por otra el procesamiento de los datos.

5.1.1. Reconocimiento vocal

A los distintos "programas" que ejecutamos con Alexa se les llama *skills*, existen diversos tipos de *skills* enfocadas a distintos usos del sistema, pero en la que nosotros nos vamos a enfocar son las "custom *skills*".

Las custom *skills* se caracterizan por permitir desarrollar prácticamente cualquier tipo de interacción sin restricciones previas.

Cuando interactuamos con una *skill*, todo comienza por una palabra clave de arranque, que recibe el nombre de "wake word" (en este caso concreto la wake word es "alexa", para google assistant sería "ok, google"), una vez pronunciada la wake word, alexa entiende que estamos interactuando con ella, y procede a escuchar el resto de la frase.

Una vez hemos obtenido "la atención" de nuestro asistente debemos indicarle que rutina/aplicación de todas las disponibles queremos ejecutar, para ello deberemos referirnos al nombre de la *skill* o "Invocation name". Dicho invocation name puede ser usado de 2 maneras distintas:

- Arranque & petición: pedimos al asistente que inicialice el servicio y quede a la espera de nuevas ordenes

  Ejemplo: Alexa inicia uvainformer.

- Petición directa: pedimos al asistente directamente los datos que necesitamos del servicio

  Ejemplo: Alexa pide a uvainformer...

A partir de aquí entramos en lo que es el grueso de nuestra *skill*, ya que es ahora cuando podremos especificar que información queremos, pero para ello deberemos utilizar una serie de formulaciones predefinidas en la *skill*.

5.1.2. Procesamiento de datos

Una vez Alexa a terminado con la parte del reconocimiento vocal, los datos que se han extraído se paquetizan y se procesan.

El resultado de ese procesamiento es una cadena de texto que es devuelta a Alexa para ser enunciada con los datos solicitados.

5.2. Partes de una *skill*

Todas las *skills* de Alexa se componen de las mismas partes. En algunas podremos editarlas todas mientras que en otras solo algunas de ellas.

5.2.1. Wake word

Como ya se definió previamente, la wake word es la palabra que "despierta" a nuestro asistente, esta palabra viene ya definida y puede ser "Alexa", "Amazon" o "Echo", una vez se pronuncia el asistente comenzará a escuchar todo lo que se diga a continuación.

5.2.2. Inicialización

Como hemos mencionado con anterioridad, las *skills* de alexa se pueden inicializar de dos maneras distintas:

Inicialización sin petición

Podemos inicializar la *skill* de Alexa sin ninguna petición definida si utilizamos cualquiera de los siguientes modelos:

- <invocation name>
- <palabra de arranque><invocation name>

Donde las palabras de arranque pueden ser:

Empiece, lanza, abre, inicia, pon, comienza, usa.

Por ejemplo: Alexa, lanza uvainformer.

Inicialización con petición

Si queremos inicializar la *skill* y obtener información en una única petición podemos utilizar un nuevo conjunto de palabras de arranque:

Pregúntale a, pídele a, dile a

Por ejemplo: Alexa, **pídele a** uvainformer datos sobre....

O utilizar algunas de las que ya hemos nombrado en conjunción con un nuevo grupo de palabras:

Y, para, para que.

<palabra de arranque><invocation name><palabra de conexión><alguna acción>

Por ejemplo: Alexa, inicia uvainformer y pide los datos de ...

5.2.3. Invocation Name

El invocation name es el nombre por el cual se identifica la *skill*, no tiene por qué coincidir con el nombre que se la da a la *skill* (aunque suele ser lo habitual).

Existen ciertas reglas a la hora de elegir el invocation name, entre ellas destacamos

- El nombre no puede contener ninguna de las wake words
- El nombre no puede contener ninguna de las palabras de inicialización
- Los nombres compuestos por una única palabra solo se permiten previa acreditación de propiedad intelectual sobre dicha palabra
- Los nombres compuestos por dos palabras no podrán utilizar un articulo definido o indefinido, ni una preposición como una de dichas palabras

En los ejemplos realizados hasta ahora como invocation name hemos utilizado "uvainformer", aunque siguiendo las reglas podemos descomponerlo en "uva informer".

5.2.4. Intents

Una *skill* no reconoce únicamente un solo tipo de pregunta o petición, al contrario, es capaz de reconocer tantos tipos de preguntas distintas como nosotros programemos.

Cada una de esas preguntas o peticiones recibe el nombre de Intent.

Cada intent que creemos se encargará de entender un tipo de petición distinta que haga el usuario, pongamos un ejemplo para que quede más claro:

Tenemos una *skill* de una agencia de viajes podrá ofrecer información sobre los paquetes vacacionales que tienes o sobre rutas simples de viajes. El usuario cuando vaya a solicitar información sobre los paquetes vacacionales no preguntará de la misma manera que sobre las rutas simples, ya que los paquetes vacacionales solo necesitan la información del destino deseado

- Alexa, pide a uva viajes las ofertas en Italia

Mientras que las rutas simples necesitarán la información de origen y destino.

- Alexa, inicia uva viajes y pide información para viajar de Valladolid a Palencia.

Como hemos podido observar las dos peticiones no comparten nada más que la wake word y el invocation name.

5.2.5. Slots

Cuando realizamos una petición de todas las palabras que componen la frase nos interesan únicamente unas pocas, esas pocas palabras que nos interesan son las que nos ofrecen la información relevante sobre la que el usuario está preguntando, y para poderlas tratar las encapsulamos en slots.

Un slot recoge los posibles valores en una posición concreta de las frases y nos permite comprobar si esa palabra está dentro de las que esperábamos e incluso asociarle un ID.

Sigamos con el ejemplo de la agencia de viajes.

En la frase: Alexa, pide a uva viajes las **ofertas** en **Italia**, podemos pensar que la agencia no solo tiene ofertas, también tendrá paquetes normales, hoteles, espectáculos, etc... Así mismo no trabajará únicamente con Italia, habrá una lista de países disponibles.

En este caso Italia corresponderá al slot país y Ofertas al slot producto, el slot de país estará compuesto por un listado de países de los cuales la agencia tiene información.

5.2.6. Utterances

A cada forma distinta de invocar un mismo intent se le da el nombre de utterance.

Cuando inicializamos un intent podemos utilizar distintos verbos, y no todos utilizan el mismo tipo formulación para referirse a la misma idea, es decir, si nosotros decimos:

> Alexa, **pide a** uva viajes **las** {producto} **de** {país}

El intent estará compuesto por la frase: las {producto} de {país}, pero también podemos solicitar la misma información de la siguiente manera:

> Alexa, inicia uva viajes y pregúntale **en** {país} **que** {producto} **hay**.

De manera que dentro del mismo intent también se recogerá la frase: en {país} que {producto} hay.

Cada una de estas frases es una utterance.

Es recomendable definir múltiples utterances para que concuerden con las palabras de inicialización o que encajen con distintas formas de preguntar la información, de esta manera el usuario podrá utilizar la *skill* de manera más sencilla y fluida.

5.3. Tipos de *skill*

A la hora de crear una *skill* existen distintas maneras de hacerlo, Alexa nos ofrece por defecto una serie de plantillas que vamos a definir a continuación.

5.3.1. Custom

Este tipo de *skill* es la más amplia de todas, permite el desarrollo de todas las demás.

Dentro de una custom *skill* podremos definir nuestros propios intents, así como las utterances de los mismos.
También nos permitirá programar todo el funcionamiento en la propia plataforma de desarroolo de Alexa

5.3.2. Flash Briefing

Una flash briefing *skill* sirve para devolver noticias o contenido al usuario, esta *skill* solo nos permitirá definir las fuentes de información y las utterances.

5.3.3. Smart Home

Las Smart Home *skills* están enfocadas a la dómotica, a través de esta *skill* podremos interactuar con distintos dispositivos externos, pero hasta ahí llega su alcance.

Deberemos proporcionarle nuestro propio endpoint y nos permitirá definir las peticiones que reciba (esto no son intents, ya que están acotadas a funcionalidades de dispositivos existentes) así como sus utterances.

5.3.4. Video

Las *skills* de Video sirven para controlar el funcionamiento de aplicaciones de streaming es decir, esta *skill* podrá conectarse con aplicaciones dentro de fire TV o dispositivos echo con pantalla.

También permite desarrollar *skills* para controlar dispositivos de entretenimiento, pero deberá ser el fabricante del dispositivo quien desarrolle la *skill*.

Capítulo 6

Estudio de alternativas

En este capítulo se analizarán las diversas posibilidades de desarrollo.

Dado que el objetivo de este trabajo es desarrollar una aplicación vocal que permita el acceso a datos del entorno académico, comenzaremos enumerando que datos existentes pueden considerarse de interés, para posteriormente estudiar la viabilidad de dichas ideas. Este estudio no solo servirá para el proyecto actual, también se pretende establecer futuras vías de desarrollo para la aplicación de manera que el producto final que se genere sea de utilidad para los integrantes de la comunidad educativa (desde alumnos a profesorado).

6.1. Datos de interés

Fijándonos en un estudiante recién llegado a la universidad podemos detectar una serie de datos que puedan ser necesarios, y que es probable que no sepa como encontrar.

1. Existencia y ubicación de libros disponibles en los distintos emplazamientos de la Universidad de Valladolid
2. Horario de tutorías del profesorado
3. Información básica del profesorado
4. Disponibilidad horaria real del profesorado
5. Guías docentes de las asignaturas

6.2. Estudio de viabilidad

Para cada una de estas ideas se ha analizado la extracción, tratamiento y presentación de los datos propuestos.

6.2.1. Existencia y ubicación de libros

Ubicación

La información solicitada se encuentra en el directorio de Almena UVA.

Obtención de datos

No se ha encontrado una manera automatizable de obtención de datos, no cuenta con funcionalidades REST ni se han conseguido obtener los datos mediante otros procesos automáticos que se puedan implementar en javascript.

Manipulación

Los datos se pueden adaptar a un formato JSON, lo que permitiría realizar operaciones básicas de búsqueda y ordenación sobre ellos sin mayor problema.

Presentación

Los datos se pueden adaptar a un formato JSON, por lo que sería sencillo presentar la información requerida a través de Alexa.

6.2.2. Horario de tutorías

Ubicación

La información solicitada se encuentra en la página web `uva.es` indexada de manera alfabética, por campus, y por rama por lo que se puede establecer una tabla de búsqueda.

Obtención de datos

Los datos se encuentran en la página web `uva.es` en alfabética y es posible acceder a ellos.

Manipulación

Los datos se encuentran en formato embebido como tablas tipo html, por lo que la manipulación de estos es costosa.

Presentación

Los datos de ser accesibles se podrían presentar en un formato tanto vocal como visual.

6.2.3. Información básica

Ubicación

La información solicitada se encuentra en `directorio.uva.es`.

Obtención de datos

Los datos se pueden obtener mediante ordenes POST y GET.

Manipulación

Los datos están en formato JSON, por lo que su manipulación es sencilla

Presentación

Los datos se pueden adaptar a un formato JSON, por lo que sería sencillo presentar la información requerida a través de Alexa.

6.2.4. Disponibilidad horaria

Ubicación

No se han podido encontrar los datos, se baraja la opción de solicitarlos al profesor correspondiente vía email

Obtención de datos

La opción de obtención de los datos vía email a tenido que ser descartada por imposibilidad de desarrollo en la plataforma de Alexa

Manipulación

No contamos con los datos en ningún formato por lo que no se puede especular

Presentación

No contamos con los datos en ningún formato por lo que no se puede especular

6.2.5. Guías docentes

Ubicación

Los datos se encuentran en la página web `uva.es` en alfabética y es posible acceder a ellos.

Obtención de datos

Los datos se encuentran en formato PDF por lo que se podrían obtener mediante descarga

Manipulación

Al estar en formato PDF no se puede acceder a los datos.

Presentación

No se puede presentar un PDF directamente, y al no poder acceder a los datos es imposible su presentación

6.3. Decisión final

6.3.1. Datos

Se ha optado por la opción más sencilla, la información básica del profesorado, ya que es la que mejor se adapta al proyecto, tanto por la complejidad de desarrollo como por el tiempo necesario para su explotación.

6.3.2. Skill

Se ha optado por desarrollar una Custom Skill, ya que es la única que nos brinda la libertad suficiente para llevar a cabo el desarrollo planteado.

Capítulo 7

Desarrollo

En este último capítulo se describe todo el proceso de desarrollo seguido en la creación de la custom *skill* “uva informer”

Para ello vamos a empezar con un análisis previo de los componentes que intervienen en la propia aplicación, en base a los estudios previos realizados. Después seguiremos con el planteamiento de la parte vocal, desde las sentencias esperadas a las propuestas de respuesta. Y por último acabaremos exponiendo el núcleo del código que ejecutará la aplicación.

Dado que tenemos 2 propuestas distintas de implementación cada sección contará con su propio apartado sobre la misma, esto se debe a que en lugar de casos de uso para la aplicación, aquí contamos con los distintos intents que se han desarrollado, ya que fuera de estos no se pueden realizar más interacciones.

7.1. Análisis y arquitectura

El sistema de Alexa *skills* cae dentro de la arquitectura de agente de diálogo orientado a tareas, ya que el funcionamiento principal de Alexa consiste en obtener una petición por parte del usuario e intentar completarla,, pudiendo solicitar más información de ser necesaria mediante un diálogo no muy desarrollado, ya que todas las sentencias que Alexa emite están predefinidas por el desarrollador de la *skill*.

Este modelo de funcionamiento se puede ver tanto en la Figura 7.1, donde se muestran los componentes que intervienen en la operación, como en la Figura 7.2, donde podemos ver más en detalle las interacciones entre los componentes cuando se ejecuta un intent cualquiera, ya que como se explica en Sección A.2, hay una estructura común de funcionamiento para todas las *skills*.

7.1.1. Intent 1: Información de profesores

La primera de las propuestas de aplicación consiste en obtener cierta información del profesorado a través de la página web `http://directorio.uva.es/search`. Como se puede observar en la Figura 7.3

el directorio UVA cuenta con un panel de búsqueda que contiene distintos campos para rellenar, este sistema se asimila a las plantillas con huecos de la arquitectura, por lo que vamos a intentar realizar un enlace entre los mismos.

Indagando en el funcionamiento del sistema de búsquedas del directorio UVA, utilizando la consola de desarrollo del navegador web chrome, podemos observar que cuando realizamos

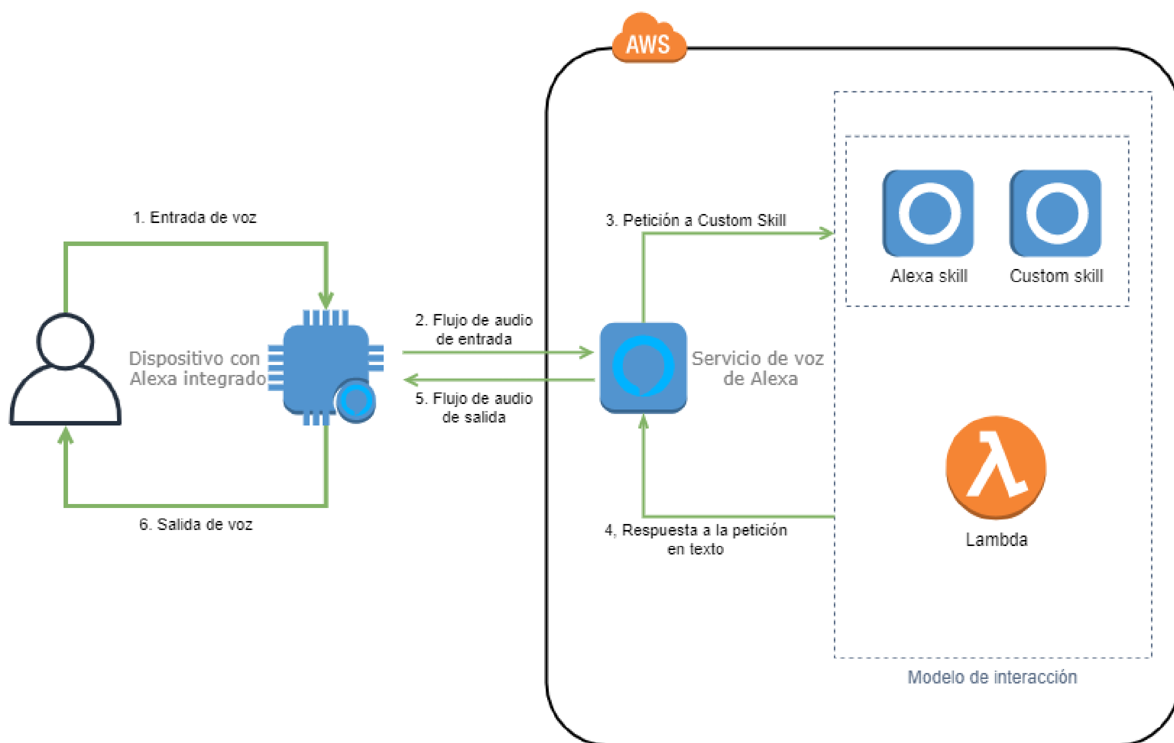

Figura 7.1: Diagrama conceptual de funcionamiento

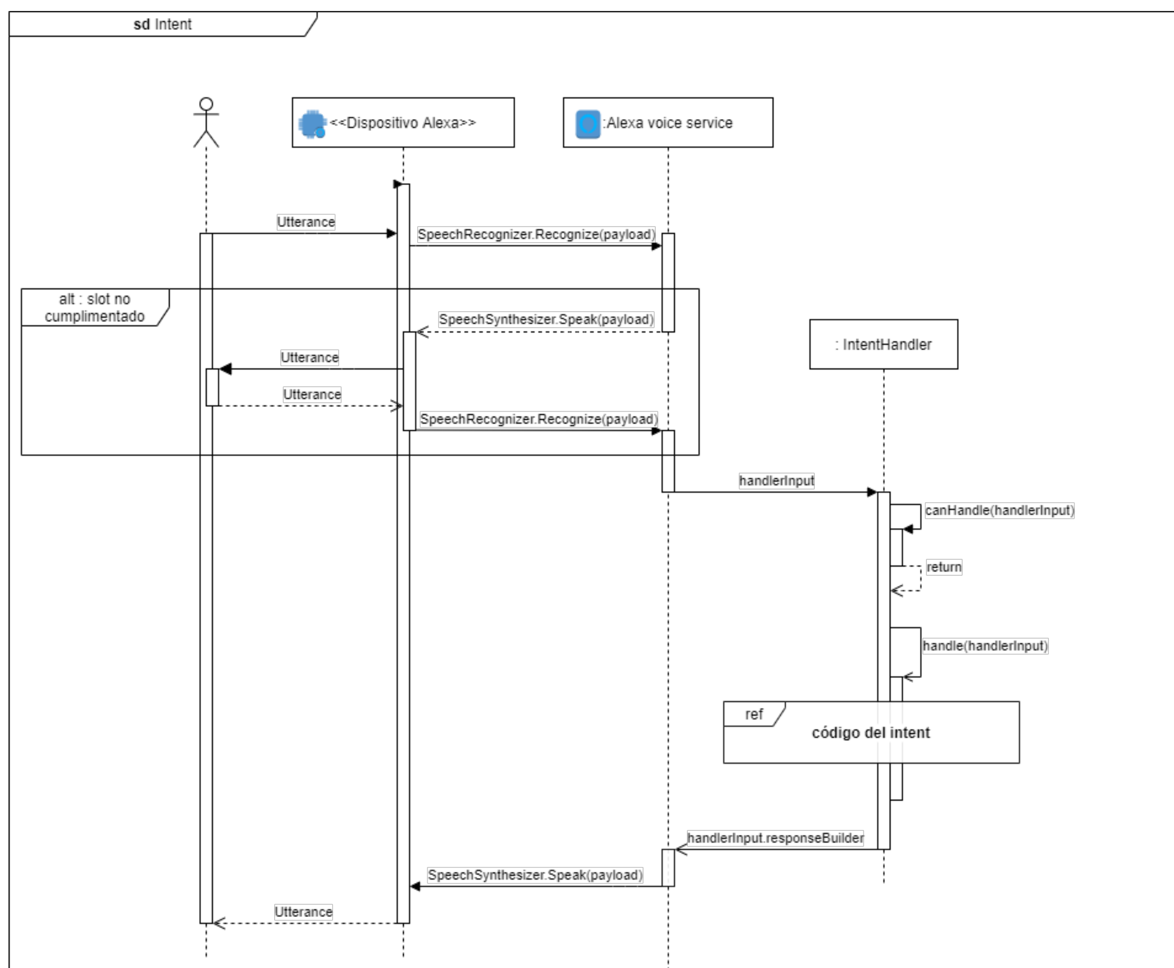

Figura 7.2: Diagrama de secuencia de un intent cualquiera

Figura 7.3: Panel de búsqueda

una búsqueda por nombre, el funcionamiento interno de la página consiste en una petición de tipo GET a una url en la que se incluyen los tags de los parámetros de búsqueda (Figura 7.4), así como el texto introducido codificado para html (se puede apreciar en las dos primeras líneas del apartado General de la Figura 7.4)

Con la información que proporciona la consola, podemos extraer los parámetros necesarios para ejecutar nuestras propias ordenes GET y POST, así que procedemos a comprobarlo mediante una orden CURL/Fetch ejecutada desde un terminal (Figura 7.5).

Si nos fijamos en el resultado que nos ofrece la interfaz web de directorio UVA a la búsqueda Figura 7.6, podemos observar que hemos obtenido varios resultados, pero de esos resultados solo nos muestra el nombre junto con el cargo, mientras que el resultado ofrecido tanto por la consola de desarrollo de chrome, como el obtenido mediante la orden curl en terminal, nos muestra toda la información disponible Figura 7.7.

Con esta información que podemos obtener en formato JSON, ya tenemos todo lo necesario de momento para poder desarrollar el intent de nuestra *skill*.

7.1.2. Intent 2: Horarios de tutorías

En esta segunda propuesta de aplicación queremos obtener los horarios de tutorías de cada profesor, dicha información está disponible de forma abierta y en formato html en la página web de la Universidad de Valladolid, aunque no de manera directa.

Dado que nuestra *skill* esta planteada como un sistema de marcos y huecos, tenemos que plantear que datos necesitamos para poder generar una respuesta adecuada. Lo primero sería el nombre del profesor en cuestión del que queremos conocer los horarios. Pero no nos basta solo con esto, ya que un mismo profesor puede dar clases en distintos grados, y tener asociadas distintas horas para cada grado. Esta estructura de profesor en grado es la que sigue la página web de la Universidad de Valladolid para organizar la información, por lo que a parte del nombre del profesor, necesitamos también el grado en el que imparte clase.

Con esta información vamos a poder realizar una búsqueda en la página web de la Universidad de Valladolid, donde en el apartado de Docencia > Grado > Oferta Formativa Grados > Alfabética se encuentran listados todos los grados que se imparten.

Similar a la sección anterior vamos a realizar una petición a una página web (`https://www.uva.es/export/sites/uva/2.docencia/2.01.grados/2.01.02.ofertaformativagrados/`

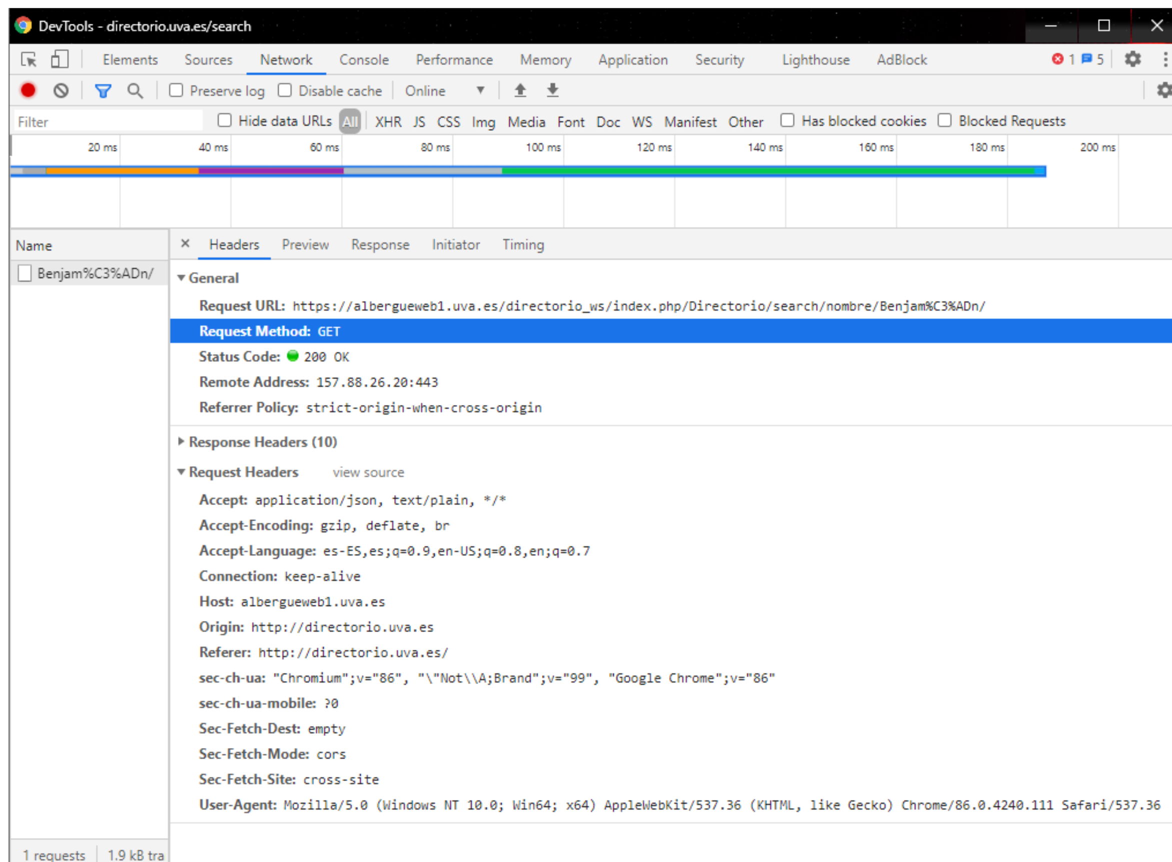

Figura 7.4: Consola de desarrollo

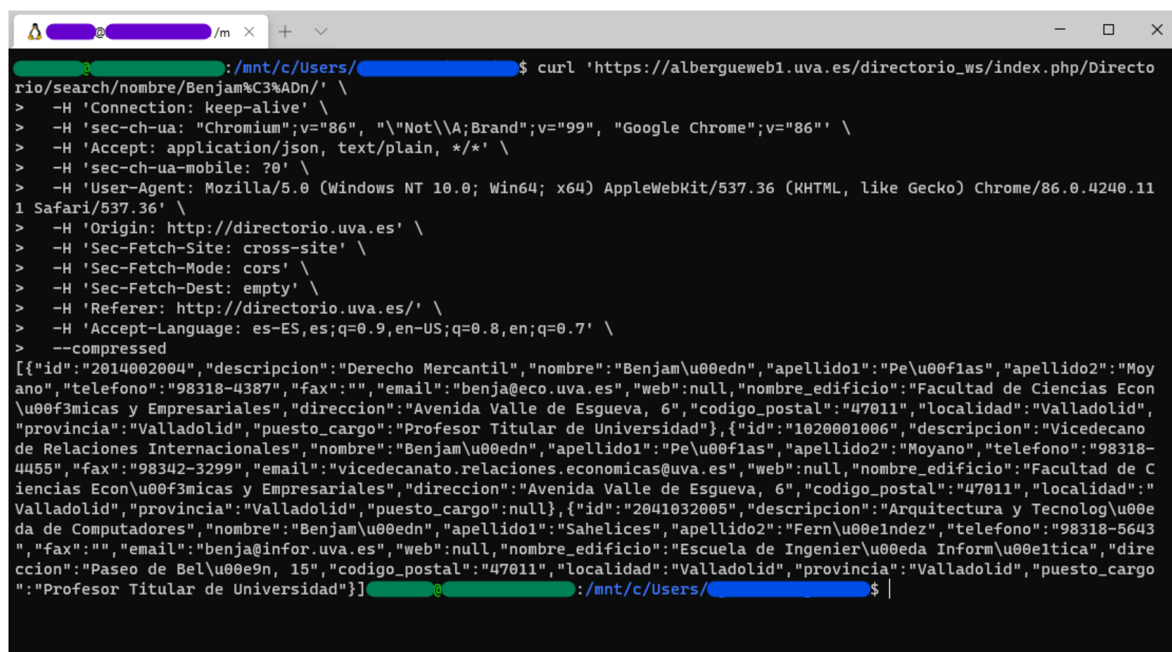

Figura 7.5: Orden CURL sobre directorio UVA

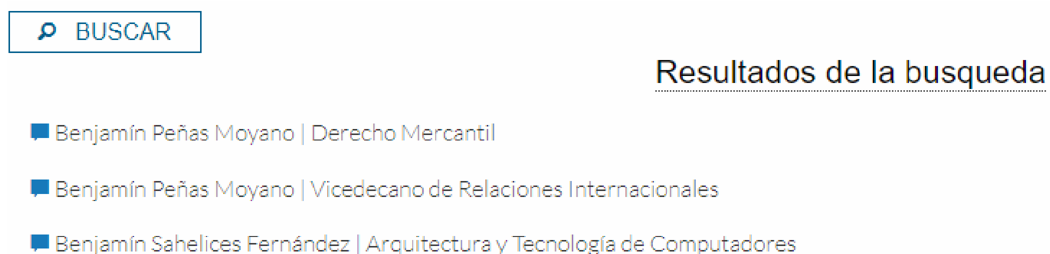

Figura 7.6: Resultado web de una búsqueda en directorio UVA

Name | Headers | Preview | Response | Initiator | Timing

Benjam%C3%ADn/


```
▼[{id: "2014002004", descripcion: "Derecho Mercantil", nombre: "Benjamín", apellido1: "Peñas",…},…]
 ▶0: {id: "2014002004", descripcion: "Derecho Mercantil", nombre: "Benjamín", apellido1: "Peñas",…}
 ▶1: {id: "1020001006", descripcion: "Vicedecano de Relaciones Internacionales", nombre: "Benjamín",…}
 ▶2: {id: "2041032005", descripcion: "Arquitectura y Tecnología de Computadores", nombre: "Benjamín",…}
```


Figura 7.7: Respuesta a la orden GET realizada sobre directorio UVA

`2.01.02.01.alfabetica/index.html`), para poder obtener el listado de los grados. La respuesta a esta petición podemos verla tanto en la consola de desarrollo de chrome como en un terminal (mediante su correspondiente orden CURL/Fetch),y viene en la forma de una página web en formato html como se ve en la Figura 7.8

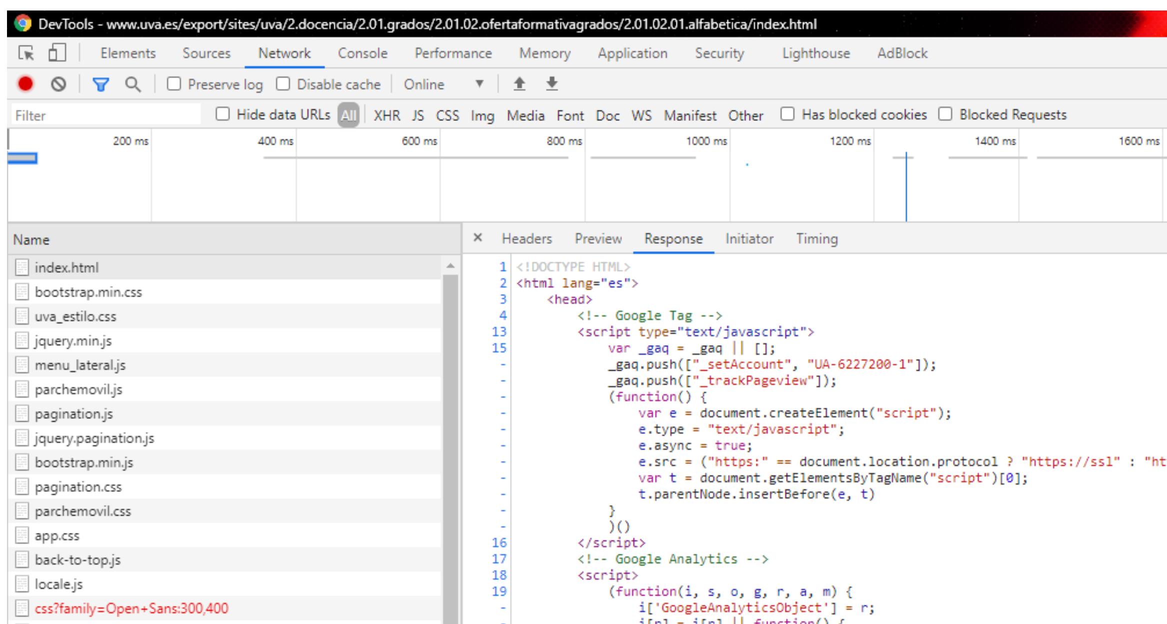

Figura 7.8: Página web alfabética de la UVA como respuesta a petición GET

Al igual que dentro de la página web (Figura 7.9), dentro del archivo html vienen listados todos los grados disponibles utilizando la estructura que se muestra en la Figura 7.10

Dado que la estructura es común a todos los enlaces, pero los nombres de los grados son únicos, podemos utilizar estos, para obtener el enlace a la información concreta del grado.

Ahora que tenemos el enlace vamos a repetir de nuevo el proceso, realizando una solicitud y obteniendo una página web en formato html de vuelta. Lo que ahora queremos obtener es el equivalente a la tabla de horarios que se ve en Figura 7.11. Pero esa tabla no se encuentra en la página web actual, si no que tiene su propia ruta, si buscamos la ruta de donde la

Figura 7.9: Página web de la Universidad de Valladolid con el listado de grados

```html
<div class="elemento_elementos_lista">
    <div>
        <div id="titulo_grado_new">
            <p>
                <a href="/export/sites/uva/2.docencia/2.01.grados/2.01.02.ofertaformativagrados/detalle/Grado-en-Ingenieria-Informatica/">Grado en Ingeniería Informática</a>
            </p>
        </div>
    </div>
</div>
```

Figura 7.10: Segmento en html que define el enlace al grado

página web carga la tabla, veremos que el archivo da igual en que grado entremos, se llama **tutoria.htm**, y que dicho archivo se puede obtener mediante una petición GET. Esa petición nos devolverá la tabla de horarios de tutorías en formato html, y ahora solo es cuestión de extraer la información que necesitamos.

Figura 7.11: Página web de la Universidad de Valladolid con el listado de horarios de tutorías

7.2. Diseño vocal

En esta sección vamos a definir los intents y slots que se han creado para cada intent, ya que el proceso de creación de los mismos se ha definido en el Apéndice A. Todo el código de la *skill* que se está presentando aquí, se puede encontrar en `https://github.com/lvareo/UVaInformer.git`

7.2.1. Intent 1: Información de profesores

Slots

Siguiendo el análisis realizado en la sección anterior, todo lo que necesitamos saber es el nombre del profesor, por lo que el único slot que definamos deberá recoger el nombre. Dado

que no tenemos un listado de todos los profesores de la universidad, vamos a hacer uso del slot predefinido de alexa: AMAZON.SearchQuery.

Este slot recoge cualquier tipo de entrada, lo que nos permitirá realizar las búsquedas. Otra opción que vamos a marcar es que es un slot necesario, y que por lo tanto en caso de no suministrar el usuario un contenido para el mismo, tendremos que preguntar específicamente por este.

En cuanto a los utterances vamos a tener de dos tipos, los propios del intent, y los asociados al slot, y comenzaremos definiendo estos últimos.

Lo primero será definir las preguntas que haremos cuando el slot no haya sido rellenado con la información ofrecida, podemos para ello realizar preguntas del estilo:

- ¿Qué profesor estás buscando?
- ¿De qué profesor quieres saber la información?

A lo que el usuario como respuesta pueda dar:

- <Nombre >
- A <Nombre >
- Estoy buscando a <Nombre >
- Quiero la información de <Nombre >

Es obvio que cuantas más variaciones aportemos mayor fluidez podrá tener la interacción.

Utterances

Ahora pasaremos a los utterances propios del intent, para estos hay que tener en cuenta que no solo vamos a acceder directamente al mismo, si no que hay varios caminos para activarlo, por ello vamos a ir generando los intents en función de los modos de acceso.

Llamada directa Cuando activamos el intent de manera directa desde Alexa, el usuario realizará peticiones como "Alexa pide a Uva Informer **la información de <Nombre >**", de donde extraemos que el utterance será: la información de <Nombre >.

Otras utterances asociadas a la llamada directa podrían ser:

- datos sobre <Nombre >
- datos de <Nombre >
- la información sobre <Nombre >

Arranque de *skill* Al iniciar la *skill* sin ejecutar ningún intent concreto por defecto se ejecuta el LaunchRequest (que veremos más adelante en la Sección 7.3), esta petición corresponde con el texto de bienvenida a la *skill*, y dicho texto ofrece información sobre las capacidades de la misma con la siguiente frase:

Hola, bienvenido a Uva Informer. Puedo ofrecerte información sobre profesores u horarios de tutoría. ¿Cuál te gustaría probar?

En base a esta sentencia el usuario podría responder de las siguientes maneras:

- información sobre profesores
- profesor
- profesores

Interacción con *skill* iniciada La última manera de acceder al intent es cuando la *skill* ya se ha iniciado, y el usuario realiza una petición directa sin la necesidad de preguntar primero por el nombre de la *skill*, esto lleva a que emita comandos como:

- quiero la información de <Nombre >
- dame la información de <Nombre >

Y con esto quedaría cubierto para nuestro prototipo las utterances del primer intent, de la misma manera vamos a ver las del segundo.

7.2.2. Intent 2: Horarios de tutorías

Para este intent hemos visto en la sección anterior que vamos a necesitar dos tipos de datos, por un lado el nombre del grado, y por el otro el nombre del profesor. Como ambos slots admiten gran cantidad de respuestas y ajustándonos al tiempo disponible para el desarrollo del proyecto, se ha decidido que ambos slots sean del tipo AMAZON.SearchQuery.

Dado que ambos slots son imprescindibles para el correcto funcionamiento de la *skill* vamos a tener que definir utterances para los mismos como en el intent previo.

Para el slot asociado a los grados, realizaremos preguntas como:

- ¿De qué grado necesitas la información?
- ¿Para qué grado estás buscando información?

y las respuestas esperadas por parte del usuario serán:

- para el grado en <grado >
- <grado >
- de <grado >
- para <grado >

Por otro lado para preguntar por el nombre del profesor realizaremos preguntas como:

- ¿De qué profesor quieres saber el horario?
- ¿Para qué profesor quieres saber la información?

y de nuevo las respuestas del usuario tomarán la forma:

- De <Profesor >
- Para <Profesor >
- <Profesor >

Una vez solucionada la parte correspondiente a los slots vamos a proceder con el intent en sí.

Dado que necesitamos que ambos intents sean completados, y que hemos marcado la opción de slot filling en ambos, no podemos utilizarlos a la vez en las utterances (a parte de que no quedarían muy naturales), por lo que la forma más natural de preguntar se ha reducido a usar de manera directa solo el slot de profesor.

Llamada directa

- el horario de tutorías
- qué horario de tutorías tiene <Profesor >
- el horario de tutorías de <Profesor >

7.2.3. Arranque de *skill*

Recordemos que al iniciar la *skill* sin ejecutar ningún intent Alexa nos contestará con:

Hola, bienvenido a Uva Informer. Puedo ofrecerte información sobre profesores u horarios de tutoría. ¿Cuál te gustaría probar?

Por lo que las utterances asociadas a esta pregunta serán:

- tutorías
- tutoría
- horarios
- horario

7.2.4. Interacción con *skill* iniciada

- dame el horario de tutorías de <Profesor >
- dame el horario de tutorías
- quiero el horario de tutorías de <Profesor >
- quiero el horario de tutorías

7.3. Implementación

En esta sección se describirá el proceso de implementación del código *backend* que hará funcionar nuestra *skill*. El código *backend* es el encargado de realizar las acciones asociadas a cada intento para poder proporcionar los datos o acciones solicitados por el usuario en dicho intento. Para ello, podrá recuperar también los datos del dominio proporcionados en el intento a través de los slots. La estructura de este código está predefinida por Alexa. Esencialmente, a cada intento se asocia un *handler* y otros al tratamiento de errores y de inicio y terminación ordenada (ver Sección A.2 para más detalles en el caso de la aplicación de aprendizaje).

En el caso de la skill que estamos desarrollando se han creado una serie de archivos para agrupar distintas funciones en base a su funcionamiento aportando mayor modularidad al desarrollo.

7.3.1. Archivos auxiliares

textUtils.js

En este archivo se han agrupado todas las funciones que tiene que ver con la manipulación de cadenas de texto y una función para extraer los valores de los slots de el paquete que se recibe del servicio de voz de Alexa (ya que como se puede ver en el Apéndice B la estructura del paquete es compleja.)

restApi.js

En la primera sección de este capítulo se ha realizado un análisis del comportamiento que tendrá cada intent, y el paso que más se ha repetido ha sido la realización de llamadas GET a distintas dirección web, por lo que se ha recogido en un solo archivo (este) todas las funciones que realizan dichas llamadas.

subIndex.js

Como la programación que estamos realizando es asíncrona, podemos generar errores en las respuestas de Alexa si cuando llamamos al servicio de voz no hemos obtenido todavía los datos esperados, esto se puede dar ya que en una ejecución asíncrona no se espera a que finalice la última función ejecutada antes de continuar con la siguiente.

Para solucionar esto se ha optado por el uso de las instrucciones async/await y la ejecución secuencial con promesas, de esta manera podremos esperar a tener los resultados antes de realizar la llamada al servicio de voz de Alexa.

Dado que cada intent tiene su propia ejecución, para mantener un poco de orden, estructura y limpieza en el código, se han separado las funciones secuenciales de cada intent en el archivo subIndex.js, que por lo tanto contiene el núcleo de ejecución de los intents que hemos desarrollado.

7.3.2. Intent 1: Información de profesores

Comenzamos explicando el funcionamiento de este primer intent por el manejador que se encuentra en fichero index.js

```
27 const TeacherHandler = {
28     canHandle(handlerInput) {
29         return handlerInput.requestEnvelope.request.type === 'IntentRequest'
30             && handlerInput.requestEnvelope.request.intent.name === 'teacher';
31     },
32     async handle(handlerInput) {
33         var me = handlerInput.requestEnvelope;
34 
35         //slots check and extraction
36         var slots = {};
37         var sourceSlots = me.request.intent.slots;
38         slots = textUtils.getSlots(sourceSlots);
39         if (!slots) {
40             throw new Error('no se han podido detectar los slots');
41         }
42         var nombre = slots.nombre;
43         var bundle = {
44             'speechWaiting': 'Voy a mirar en los archivos.',
45             'name': nombre,
46         }
47 
48         try {
49             callDirectiveService(bundle.speechWaiting, handlerInput);
50         } catch (err) {
51             // if it failed we can continue, just the user will wait longer for first response
52             console.log(err);
53         }
54         try {
55             const responseBundle = await subIndex.executeTeacher(bundle);
56             return handlerInput.responseBuilder
57                 .speak(responseBundle.text)
58                 .withSimpleCard('Resultados', responseBundle.card)
59                 .getResponse()
60         } catch (err) {
61             console.log(err);
62         }
63     }
64 };
```

Figura 7.12: TeacherHandler: Código del manejador del intent Información de profesores

Como se puede ver en la Figura 7.12 primero se comprueba si el intento que el usuario a activado coincide.

Una vez se ha comprobado dicha coincidencia pasamos al manejador (linea 32) y empezamos el proceso de extracción de la información asociada a los slots (líneas de la 33 a la 42), el proceso de extracción como tal está contenido en una función del fichero *textUtils.js*, más concretamente en la función *getSlots*[Figura 7.13], dicha función recibe la parte del paquete que contiene toda la información de los slots, y extrae únicamente el valor o ID de los mismos.

```
//============================================================================
//Slots data extraction function
//============================================================================

/**
 * @description Extract and check the slots data.
 * @param {JSON} obj The JSON with the raw slots data.
 * @returns {object}
 */
function getSlots(obj) {
    var extracted = {}
    //Object iteration
    Object.keys(obj).forEach(function (d) {
        //Check if default slot type
        if (!Object.keys(obj[d]).includes('resolutions')) {
            extracted[d] = obj[d].value;
        }
        //Check if custom slot type and if succesfull id
        else if (obj[d].resolutions.resolutionsPerAuthority[0].status.code === 'ER_SUCCESS_MATCH') {
            extracted[d] = obj[d].resolutions.resolutionsPerAuthority[0].values[0].value.id
        }
        //Error on slot identification
        else {
            return null;
        }
    });
    return extracted;
}
```

Figura 7.13: setSlots: Código de la función encargada de extraer los slots

Con los slots ya extraídos, lo siguiente que se realiza es la creación de una variable *bundle*, que va a contener la información que enviemos a sucesivas funciones. Dentro de *bundle* también se va a generar un texto que se enviará al servicio de voz de Alexa para que emita mientras se ejecuta el resto de la función (líneas de la 43 a la 53 de *index.js*), de esta manera el usuario sabrá que la función se está ejecutando correctamente y además podrá escuchar como se ha entendido la información que ha aportado para los slots.

A partir de aquí se realiza una llamada a la función *executeTeacher* [Figura 7.14] alojada en *subIndex.js* (línea 55) que se encargará de realizar las peticiones web necesarias (siguiendo el proceso que se describe en la sección 1 de este capítulo), y devolverá el texto de respuesta para que sea emitido por el servicio de voz de Alexa (líneas de la 54 a la 62).

En la función de *executeTeacher* que hemos mencionado anteriormente podemos ver como se encarga de generar la dirección web (línea 6 de *subIndex.js*) a través de la función *checkName* de *textUtils.js* [Figura 7.15]. Dentro de la función *checkName* se comprueba si el nombre que el usuario ha emitido es solamente el nombre o por el contrario ha añadido también algún apellido, ya que el formato de búsqueda hace distinción entre ambos. Una vez ha realizado la comprobación, compone junto a la dirección web que deseamos conectar la cadena completa de llamada, y devolviendo esta como resultado.

```
4  exports.executeTeacher = function (bundle) {
5      return new Promise(((resolve, reject) => {
6          let path = textUtils.checkName(bundle.name);
7          resolve(rest.getName(bundle.name, path, function (error, data){
8              if(error){
9                  console.log(error);
10                 return error;
11             }
12             console.log("Name info retrieved succesfully.");
13             let text = '';
14             let card = '';
15             switch(data.length){
16                 case 0:
17                     text = 'No se ha encontrado a nadie por ese nombre.'
18                     card = text;
19                     break;
20                 case 1:
21                     text = 'Esta es la información encontrada sobre '+data[0].nombre+' '+data[0].apellido1+' '+data[0].apellido2
22                     card = data[0].nombre+' '+data[0].apellido1+' '+data[0].apellido2+'\n'
23                     if(data[0].telefono!==''){
24                         text+='. Su número de teléfono es <say-as interpret-as="telephone">'+data[0].telefono.replace('-','')+'</say-as>'
25                         card+='Telefono: '+data[0].telefono.replace('-','')+'\n'
26                     }
27                     if(data[0].email!==''){
28                         text+='. Su correo es '+data[0].email
29                         card+='email: '+data[0].email+'\n'
30                     }
31                     break;
32                 default:
33                     text = 'Se han encontrado '+data.length+' resultados con ese nombre, por favor añada su apellido en la búsqueda.'
34                     for(let i=0;i<data.length;i++){
35                         card += data[i].nombre+' '+data[i].apellido1+' '+data[i].apellido2+'\n '
36                     }
37                     break;
38             }
39             return {'text':text, 'card':card};
40
41         }))
42     }))
43 }
```

Figura 7.14: executeTeacher: Código de la función principal del intent de Información de profesores

```
16 function checkName(rawName){
17     var output = '';
18     var completeName = encodeURIComponent(rawName).split('%20');
19     switch(completeName.length){
20         case 0:
21             break;
22         case 1:
23             output = '/directorio_ws/index.php/Directorio/search/nombre/'+completeName[0]+'/' ;
24             break;
25         case 2:
26             output = '/directorio_ws/index.php/Directorio/search/nombre/'+completeName[0]+'/apellido1/'+completeName[1];
27             break;
28         case 3:
29             output = '/directorio_ws/index.php/Directorio/search/nombre/'+completeName[0]+'/apellido1/'+completeName[1]+'/apellido2/'+completeName[2];
30             break;
31         case 4:
32             if(completeName[1]==='de'){
33                 completeName[0]+='%20'+completeName[1];
34                 if(completeName[2]==='la' || completeName[2]==='los'){
35                     completeName[0]+='%20'+completeName[2];
36                     completeName.splice(2,1);
37                     output = '/directorio_ws/index.php/Directorio/search/nombre/'+completeName[0]+'/apellido1/'+completeName[1]
38                     break;
39                 }
40                 else{
41                     completeName.splice(1,1);
42                 }
43             }
44             else if(completeName[1]==='del'){
45                 completeName[0]+='%20'+completeName[1];
46                 completeName.splice(1,1);
47             }
48             else if(completeName[2]==='de' || completeName[2]==='del'){
49                 completeName[1]+='%20'+completeName[2];
50                 completeName.splice(2,1);
51             }
52             output = '/directorio_ws/index.php/Directorio/search/nombre/'+completeName[0]+'/apellido1/'+completeName[1]+'/apellido2/'+completeName[2];
53             break;
54         case 5:
55             if(completeName[1]==='de'){
56                 completeName[0]+='%20'+completeName[1];
57                 completeName[0]+='%20'+completeName[2];
58                 completeName.splice(2,1);
59                 completeName.splice(1,1);
60             }
61             else if(completeName[2]==='de'){
62                 completeName[1]+='%20'+completeName[2];
63                 completeName[1]+='%20'+completeName[3];
64                 completeName.splice(3,1);
65                 completeName.splice(2,1);
66             }
67             output = '/directorio_ws/index.php/Directorio/search/nombre/'+completeName[0]+'/apellido1/'+completeName[1]+'/apellido2/'+completeName[2];
68             break;
69         default:
70             break;
71     }
72     return output;
73 }
```

Figura 7.15: CheckName: Código de la función encargar de generar el path para el intent de Información de profesores

Después de generar la url, se realiza la llamada utilizando la función *getName* de *restApi.js* [Figura 7.16] donde se configura y ejecuta la petición de tipo GET en base a la url previamente generada, devolviendo la respuesta de la petición.

```
4  function getName(name, path, callback){
5      return new Promise((resolve, reject) => {
6          var get_data = '';
7
8          var headers = {
9          'Connection': 'keep-alive',
10         'Accept': 'application/json, text/plain, */*',
11         'Origin': 'http://directorio.uva.es',
12         'Sec-Fetch-Site': 'cross-site',
13         'Sec-Fetch-Mode': 'cors',
14         'Sec-Fetch-Dest': 'empty',
15         'Referer': 'http://directorio.uva.es/search',
16         'Accept-Language': 'es-ES,es;q=0.9,en-US;q=0.8,en;q=0.7'
17         };
18
19         var get_options = {
20             hostname: 'alberguewebl.uva.es',
21             path: path,
22             method: 'GET',
23             agent: new https.Agent({ rejectUnauthorized: false }),
24             headers: headers
25         };
26
27         var responseString = '';
28
29          // Request set up
30         var get_req = https.request(get_options, function (res) {
31             res.setEncoding('utf8');
32             res.on('data', function (chunk) {
33                 responseString += chunk;
34                 console.log(chunk);
35             });
36             res.on('end', function () {
37                 // Correct authentication
38                 if (res.statusCode === 200) {
39                     resolve(callback(null, JSON.parse(responseString)));
40                 }
41                 // Unknown error
42                 else {
43                     console.log(responseString);
44                     let error = "Unknown problem while reaching the cloud server." +
45                         "Please check the address configuration or contact the server administrator.";
46                     reject(callback(error, null));
47                 }
48             });
49         });
50         get_req.write(get_data);
51         get_req.end();
52     }));
53 }
```

Figura 7.16: getName: Código de la función encargar de realizar la petición GET a directorio uva.

Ahora que ya tenemos los datos solicitados por el usuario, lo único que queda es componer el texto de respuesta, para ello se analiza que tipo de información hemos obtenido y en función de ello se compone una cadena de salida u otra (líneas de la 12 a la 39 de *subIndex.js*), también se ha querido ampliar el funcionamiento de la *skill* aportando información por pantalla para que se muestre en los dispositivos compatibles.

7.3.3. Intent 2: Horarios de tutorías

Si miramos el código del manejador en la de este segundo intent, podemos observar que el comportamiento es prácticamente idéntico a al del manejador del intent sobre Información de profesores, las únicas diferencias las encontramos en el hecho de que hay dos slots en lugar de uno, y que la función a la que se llama para encargarse del procesamiento es *executeHorarios*.

```
66  const HorariosHandler = {
67      canHandle(handlerInput) {
68          return handlerInput.requestEnvelope.request.type === 'IntentRequest'
69              && handlerInput.requestEnvelope.request.intent.name === 'horarios';
70      },
71      async handle(handlerInput) {
72          var me = handlerInput.requestEnvelope;
73
74          //slots check and extraction
75          var slots = {};
76          var sourceSlots = me.request.intent.slots;
77          slots = textUtils.getSlots(sourceSlots);
78          if (!slots) {
79              throw new Error('no se han podido detectar los slots');
80          }
81          var grado = slots.grado;
82          var profesor = slots.profesor;
83          var bundle = {
84              'speechWaiting': 'Un momento que consulto las tablas de horarios del profesor '+profesor+' del grado en '+grado+'.',
85              'grado': grado,
86              'profesor': profesor,
87          }
88          try {
89              callDirectiveService(bundle.speechWaiting, handlerInput);
90          } catch (err) {
91              // if it failed we can continue, just the user will wait longer for first response
92              console.log(err);
93          }
94          try {
95              const responseText = await subIndex.executeHorarios(bundle);
96              return handlerInput.responseBuilder
97                  .speak(responseText)
98                  .withSimpleCard('Resultados', responseText)
99                  .getResponse()
100         } catch (err) {
101             console.log(err);
102         }
103     }
104 };
```

Figura 7.17: HorariosHandler: Código del manejador del intent Horarios de tutorías.

Dentro de *executeHorarios* [Figura 7.18] el primer paso que da es obtener la página web con el listado de los grados de la Universidad de Valladolid y esto lo realiza mediante la función *getWebPage* de *restApi.js*.

El funcionamiento de *getWebPage* [Figura 7.19] es similar a la función *getName*, solo que aquí esperamos obtener la página web completa como respuesta.

Una vez que hemos obtenido la página web donde están listados los grados, pasamos a identificar la línea donde está el grado que el usuario nos ha indicado en el slot grado (líneas 53 a la 61) , esto lo conseguimos gracias a la función *searchStringInArray* [Figura 7.20], que nos devuelve la línea de la página web en la que se encuentra el texto que hace referencia al grado que buscamos, con la línea identificada extraemos la dirección url sobre el grado (línea 61) ya que se encuentra en la misma línea que el nombre de este.

Con esta dirección volvemos a ejecutar la función *getWebPage* para obtener esta nueva página web, donde volveremos a realizar una búsqueda de texto mediante la función *searchStringInArray*, en este caso buscaremos el texto "tutoria.htm", que es el nombre del archivo que deseamos consultar.

De nuevo extraemos la dirección web que se encuentra en la misma línea que la cadena de texto que hemos buscado y realizamos una tercera petición GET con *getWebPage*, ahora obteniendo como resultado una tabla con la información de los horarios de tutorías del grado solicitado en formato html.

Para obtener la información ejecutamos la función *extractTimeTables* de *textUtils.js* [Figura 7.21], esta función toma como entrada la tabla completa y el nombre del profesor que se está buscando. Ya que en esta tabla tenemos la información que realmente queremos obtener,primero debemos aislar la sección donde se encuentra, dado que la tabla está formateada

```
45 exports.executeHorarios = function (bundle) {
46     return new Promise(((resolve, reject) => {
47         let mainPath = '/export/sites/uva/2.docencia/2.01.grados/2.01.02.ofertaformativagrados/2.01.02.01.alfabetica/index.html';
48         resolve(rest.getWebPage(mainPath, function (error, data){
49             if(error){
50                 console.log(error);
51                 return error;
52             }
53             let dataSplit = data.split("\n");
54             let gradoLine = textUtils.searchStringInArray(bundle.grado, dataSplit);
55             let text = '';
56             if(gradoLine === -1){
57                 text = 'No he podido encontrar el grado '+bundle.grado+'. Por favor inténtelo de nuevo.';
58                 return text;
59             }
60             console.log("Career path retrieved succesfully.");
61             let pathGrado = dataSplit[gradoLine].split("\"")[1]
62             return rest.getWebPage(pathGrado, function(error, data){
63                 let dataSplit = data.split("\n");
64                 let tablaLine = textUtils.searchStringInArray("tutorias.htm", dataSplit);
65                 let tablaPath = dataSplit[tablaLine].split("\"")[3]
66                 return rest.getWebPage(tablaPath, function(error, data){
67                     if(error){
68                         console.log(error);
69                         return error;
70                     }
71                     let timeTableArray = textUtils.extractTimeTables(data, bundle.profesor);
72                     let text = '';
73                     if (timeTableArray === -1){
74                         text = 'No he podido encontrar a '+bundle.profesor+' del grado en '+bundle.grado+'. Por favor inténtelo de nuevo.';
75                         return text;
76                     }
77                     let timeTable = textUtils.timeTableJSON(timeTableArray);
78                     text = 'El horario de tutorías para '+bundle.profesor+' es ';
79                     var today = new Date();
80                     var limit = new Date('2021-02-15');
81                     for (let j=timeTable.length-1;j>=0;j--){
82                         if(today < limit){
83                             if(timeTable[j]['formato'] !== 'Segundo Cuatrimestre'){
84                                 text = text.concat('los '+timeTable[j]['dia']+
85                                                    ' de '+timeTable[j]['desde']+
86                                                    ' a '+timeTable[j]['hasta']+
87                                                    ' en '+timeTable[j]['facultad']+
88                                                    ' despacho '+timeTable[j]['despacho'].split(" ")[1]+
89                                                    '. ')
90                             }
91                         }
92                         else{
93                             if(timeTable[j]['formato'] !== 'Primer Cuatrimestre'){
94                                 text = text.concat('los '+timeTable[j]['dia']+
95                                                    ' de '+timeTable[j]['desde']+
96                                                    ' a '+timeTable[j]['hasta']+
97                                                    ' en '+timeTable[j]['facultad']+
98                                                    ' despacho '+timeTable[j]['despacho'].split(" ")[1]+
99                                                    '. ')
100                            }
101                        }
102
103                    }
104                    return text;
```

Figura 7.18: executeHorarios: Código de la función principal del intent de Horarios de tutorías.

```
57  function getWebPage(path, callback){
58      //console.log(path);
59      return new Promise(((resolve, reject) => {
60          var get_data = '';
61
62          var headers = {
63              "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
64              "accept-language": "es-ES,es;q=0.9,en-US;q=0.8,en;q=0.7",
65              "sec-fetch-dest": "document",
66              "sec-fetch-mode": "navigate",
67              "sec-fetch-site": "same-origin",
68              "sec-fetch-user": "?1",
69              "upgrade-insecure-requests": "1"
70          };
71
72          var get_options = {
73              hostname: 'www.uva.es',
74              path: path,
75              method: 'GET',
76              agent: new https.Agent({ rejectUnauthorized: false }),
77              headers: headers
78          };
79
80          var responseString = '';
81
82           // Request set up
83          var get_req = https.request(get_options, function (res) {
84              res.setEncoding('utf8');
85              res.on('data', function (chunk) {
86                  responseString += chunk;
87  //                console.log(chunk);
88              });
89              res.on('end', function () {
90                  // Correct authentication
91                  if (res.statusCode === 200) {
92  //                      console.log(responseString);
93                      resolve(callback(null, responseString));
94                  }
95                  // Unknown error
96                  else {
97                      console.log(responseString);
98                      let error = "Unknown problem while reaching the cloud server." +
99                          "Please check the address configuration or contact the server administrator.";
100                     reject(callback(error, null));
101                 }
102             });
103         });
104         get_req.write(get_data);
105         get_req.end();
106     }));
107 }
```

Figura 7.19: getWebPage: Código de la función encargar de realizar la petición GET al dominio uva.es

```
//==============================================================================
//Text data manipulation functions
//==============================================================================


function searchStringInArray (str, strArray) {
    for (var j=0; j<strArray.length; j++) {
        if (strArray[j].toLowerCase().match(str)) return j;
    }
    return -1;
}
```

Figura 7.20: searchStringInArray: Código de la función que busca una cadena de texto dentro de las cadenas de texto de un array de cadenas de texto

en html, podemos identificar estructuras similares de código, y partir el código en bloques de dichas estructuras. Con esta idea en mente, la función *extractTimeTables* primero aísla como hemos dicho la sección de la tabla referente al profesor, y después extrae el contenido de las celdas respetando la estructura de la tabla, y es el contenido de dichas celdas lo que la función devuelve como resultado.

```
75 function extractTimeTables(webPage, profesor){
76     var timeTable = [];
77     var profesorNormalized = profesor.normalize("NFD").replace(/[\u0300-\u036f]/g, "");
78     let textByTeacher = webPage.split('<span style="font-family: Arial; color: #000000; font-size: 14px; line-height: 1; *line-height: normal; font-weight: bold;">');
79     textByTeacher.shift();
80     let timeTableLine = searchStringInArray(profesorNormalized, textByTeacher);
81     if (timeTableLine === -1){
82         return -1;
83     }
84     let timeTableText = textByTeacher[timeTableLine].split("\n");
85     for(var i in timeTableText){
86         if(timeTableText[i].match('<span style="font-family: \'DejaVu Sans\', Arial, Helvetica, sans-serif; color: #000000; font-size: 10px; line-height: 1.215332;">')){
87             timeTable.push(timeTableText[i].split(">")[1].split("<")[0]);
88         }
89     }
90     return timeTable;
91 }
```

Figura 7.21: extractTimeTables: Código de la función que extrae la información de horarios de un profesor de la tabla general en formato html

Ahora que tenemos la información de los horarios de tutorías, así como de la ubicación de las mismas, es momento de darle formato para devolver la información al usuario (líneas 77 a 104). Para ello nos valemos primero de la función *timteTableJSON* de *textUtils.js* [Figura 7.22] que nos devuelve en formato json la información extraída de la tabla en formato html, y después en función de la fecha actual (para comprobar si nos encontramos en el primer o segundo cuatrimestre) componemos la sentencia de respuesta para el usuario.

```
93  function timeTableJSON(array){
94      var timeTableFormated = []
95      let template = {};
96      for (let i=0; i < array.length/6; i++){
97          template =  {'formato':'','dia':'','desde':'','hasta':'','facultad':'','despacho':''};
98          for (let j=0;j<6;j++){
99              template[Object.keys(template)[j]]=array[(i*6)+j];
100         }
101         timeTableFormated.push(template);
102
103     }
104     return timeTableFormated;
105 }
```

Figura 7.22: timeTableJSON: Código de la función que da formato JSON a la información de horarios de profesores extraída de la tabla html

7.4. Pruebas

La propia plataforma de desarrollo nos permite realizar simulaciones de nuestra *skill* sin necesidad de contar con un dispositivo con el sistema Alexa. Gracias a esto podemos realizar las pruebas con relativa facilidad ya que no solo admite una entrada de voz, también nos permite introducir los utterances del usuario por texto, y nos devuelve por el mismo medio su respuesta.

Como con la licencia de desarrollo gratuita y basada en lambda no se puede configurar un proceso automático para realizar una batería de pruebas, estas se han realizado individualmente, comprobando el correcto funcionamiento de ambos intents y la correcta identificación de todos los utterances. Estas pruebas se han realizado con múltiples variaciones en la entrada de información para certificar el correcto funcionamiento del prototipo desarrollado.

Una demostración de las pruebas realizadas se puede encontrar en la siguiente carpeta compartida en OneDrive: `https://bit.ly/2K8jPWK`

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

Respecto a los objetivos planteados en un inicio, se puede concluir que todos han sido alcanzados. Se ha estudiado, comprendido y descrito las bases del funcionamiento de los sistemas conversacionales, sirviendo esta memoria como introducción a la materia de una manera sencilla. Gracias a los conocimientos adquiridos se ha podido realizar un primer prototipo de sistema conversacional sobre la plataforma Alexa de Amazon, y se ha podido implementar funcionalidad útil de cara al alumnado de la Universidad de Valladolid, permitiendo consultar información sobre el profesorado y sobre los horarios de tutorías, de manera que los alumnos puedan acceder fácilmente a la información y de una manera intuitiva.

8.2. Líneas de trabajo futuro

Durante el desarrollo del proyecto se han planteado distintas posibilidades para dotar de funcionalidad al prototipo, por lo que cualquiera de dichas posibilidades descartadas podría ser una nueva vía de desarrollo. A continuación se van a plantear distintas posibilidades para ampliar la funcionalidad del prototipo, así como anotaciones que se consideran relevantes sobre dichas posibilidades nacidas del estudio ya realizado.

- Consulta de disponibilidad de libros en las bibliotecas de la Universidad de Valladolid. [Sería muy conveniente desarrollar un servicio RESTful para poder acceder de manera más sencilla a la información.]

- Crear en un servidor propio de la Universidad para alojar y ejecutar el backend de la *skill*.[Las capacidades gratuitas que ofrece Amazon por defecto limitan en gran medida las posibilidades de desarrollo.]

- Publicar la *skill* de manera abierta en la tienda de *skills* de Alexa de manera que cualquiera pueda utilizarla sin necesidad de importarla a su repositorio personal.

- Ampliar las capacidades de detección de nombre del intent sobre horarios de tutorías, de manera que permita utilizar nombres y apellidos simultáneamente en la búsqueda

Anexo A

Tutorial: Creación de una Alexa Skill

Lo primero de todo vamos a necesitar una cuenta de desarrollado para Amazon Developer (`https://developer.amazon.com/es/`). Con la cuenta ya creada accederemos a la consola de desarrollo de Alexa (`https://developer.amazon.com/alexa/console/ask`).

Dentro encontraremos un listado con las *skills* que hemos desarrollado, detallando el idioma en el que está disponible, la última fecha de modificación, su estado, y una caja de opciones.

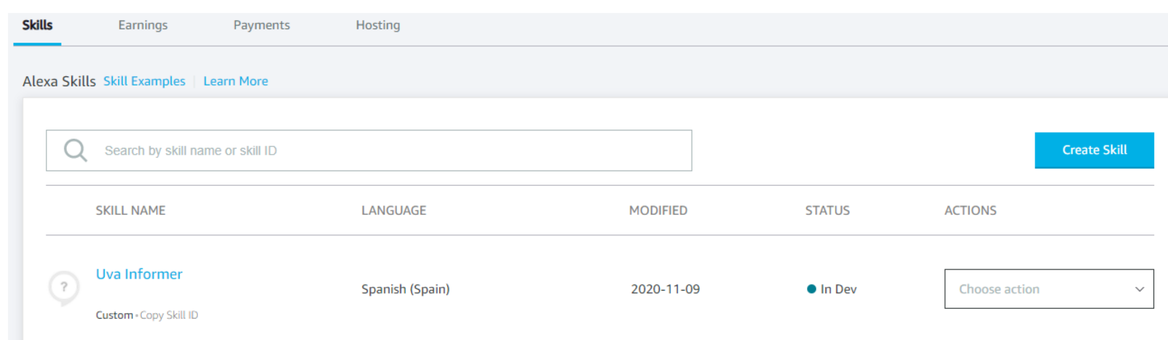

Figura A.1: Interfaz principal de la consola de desarrollo de Alexa *skills*

Para crear nuestra primera *skill* le daremos al botón azul de Create Skill situado en la esquina superior derecha del recuadro principal (como se ve en la Figura A.1).

Lo primero que debemos decidir es el nombre que tendrá nuestra *skill*, que no es lo mismo que el nombre por el que la invocaremos (invocation name). Así mismo también deberemos elegir en que idiomas vamos a desarrollar nuestra *skill* (ya que dependiendo del idioma podremos realizar unas acciones u otras).

El siguiente paso es elegir que tipo de *skill* queremos desarrollar. En función de nuestro objetivo elegiremos una opción u otra, pero ahora mismo vamos a seguir con la opción de custom *skill*, ya que es la menos restrictiva y nos permitirá explorar más.

Por último vamos a elegir la manera en la que aportaremos el código backend de nuestra *skill*, así como el lenguaje de programación en el que la vamos a desarrollar. Amazon nos ofrece por defecto utilizar sus servicios de AWS Lambda, AWS CloudWatch y AWS S3 (código, log y almacenamiento respectivamente) incluidos en la capa gratuita de AWS, donde podemos elegir realizar el desarrollo en Node.js o Python. Por otro lado podemos aportar nosotros mismos una conexión a un servidor distinto a lo ofrecido por Amazon donde tengamos toda la parte de backend, pero para simplificar y agilizar vamos a elegir la opción de Node.js

Create a new skill

Skill name

Enter skill name

0/50 characters

Default language

Spanish (ES)

More languages can be added to your skill after creation

Figura A.2: Selección de nombre e idioma

1. Choose a model to add to your skill

There are many ways to start building a skill. You can design your own custom model or start with a pre-built model. Pre-built models are interaction models that contain a package of intents and utterances that you can add to your skill.

Custom

SELECTED

Design a unique experience for your users. A custom model enables you to create all of your skill's interactions.

Flash Briefing

Give users control of their news feed. This pre-built model lets users control what updates they listen to.

"Alexa, pon el resumen de noticias."

Smart Home

Give users control of their smart home devices. This pre-built model lets users turn off the lights and other devices without getting up.

"Alexa, enciende las luces de la cocina"

Video

Let users find and consume video content. This pre-built model supports content searches and content suggestions.

"Alexa, pon Interstellar"

Figura A.3: Selección de tipo de *skill*

ofertada por defecto.

2. Choose a method to host your skill's backend resources

You can provision your own backend resources or you can have Alexa host them for you. If you decide to have Alexa host your skill, you'll get access to our code editor, which will allow you to deploy code directly to AWS Lambda from the developer console.

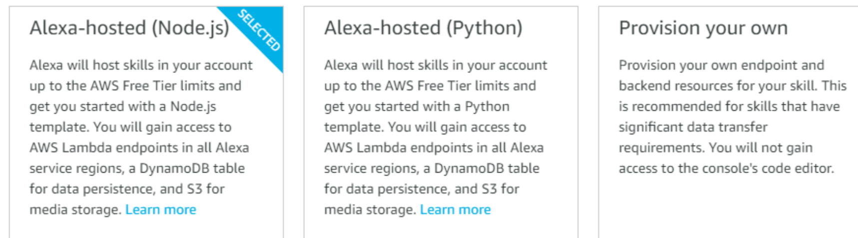

Figura A.4: Selección de aprovisionamiento de backend

Continuamos dando al botón azul de Create *skill* situado en la parte superior derecha de la página.

En la siguiente ventana tendremos que elegir si queremos desarrollar nuestra *skill* desde cero, a partir de una plantilla

Choose a template to add to your skill

Select a skill template from the list below or import a skill shared by the Alexa community as a public Git repository.

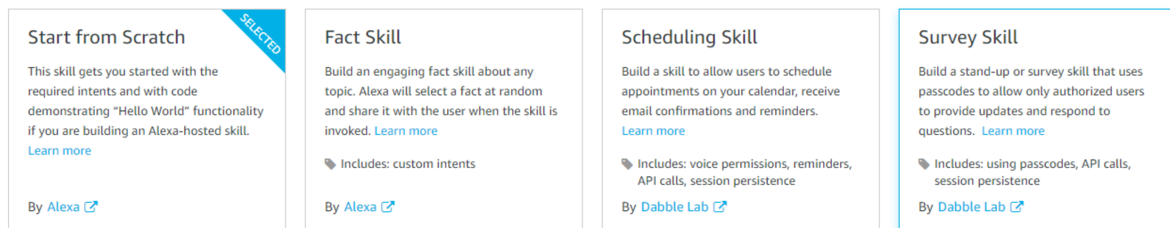

Figura A.5: Selección de tipo de plantilla para el desarrollo de la *skill*

O si por lo contrario queremos importar una *skill* desde un repositorio Git, desde la opción Import *skill* que se encuentra en la esquina superior derecha.

Vamos a elegir la opción empezar desde cero para que no nos genere código innecesario. Continuamos pulsando el botón azul de Continue with template que se encuentra en la esquina superior derecha.

Esto hará que comience el desarrollo de la *skill*, ya que la plataforma de desarrollo comenzará a configurar automáticamente los componentes de la plantilla que hayamos seleccionado.

A partir de aquí vamos a explicar brevemente que podemos encontrar en las secciones más destacadas, ya que la manera de completarlas depende de la *skill* que se quiera desarrollar.

A.1. Build

Antes de empezar debemos saber que para aplicar cualquier cambio que hagamos deberemos darle al botón de Build Model, de lo contrario aunque hayamos guardado los cambios, estos no tendrán efecto alguno.

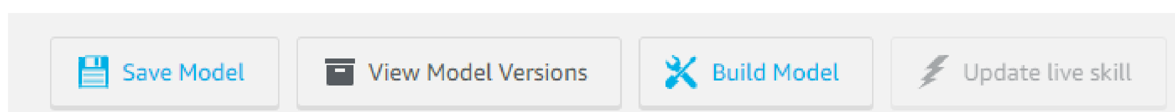

Figura A.6: Barra de herramientas del modelo.

A.1.1. Custom

En la pestaña de Custom encontraremos una serie de guías sobre el desarrollo de *skills* escritas por el equipo de desarrollo de Alexa, también encontramos un check-list de los pasos necesarios para crear una *skill* de Alexa.

A.1.2. Invocation

En este apartado definiremos el nombre por el que la *skill* es invocada, siguiendo las normas descritas en Subsección 5.2.3

A.1.3. Interaction Model

Aquí vamos a interactuar principalmente con el apartado de Intents, ya que el resto de apartados son demasiado avanzados para incluirlos en esta guía rápida.

Dentro de Intents veremos que contamos con cinco intents predefinidos, de los cuales cuatro son propios del sistema, ya que se encargan de la navegación, cancelación y parada de la *skill*. El quinto intent es un intent muy simple de ejemplo al más puro estilo hola mundo, que es el primer ejemplo que se pone a la hora de aprender un lenguaje de programación.

Si entramos dentro del intent HelloWorldIntent (Figura A.7) veremos que tenemos una serie de utterances ya definidas, estas frases son las que la *skill* espera que el usuario vocalice para ejecutar el intent.

Figura A.7: Pantalla de desarrollo de HelloWorldIntent

También podemos observar que para este intent en concreto no contamos con ningún slot definido, ya que no esperamos que ninguna de las utterances pueda aportar un dato relevante más allá de la propia sentencia.

A.1.4. Assets

Dentro de esta pestaña es donde veremos los slots que hemos definido, como hablábamos en la Subsección 5.2.5, vamos a crear a modo de ejemplo un slot para los países de una agencia de viajes.

Para ello tenemos dos posibilidades, como podemos ver en la Figura A.8, podemos crear nuestro slot desde cero, o elegir uno predefinido de Alexa.

Figura A.8: Pantalla de creación de un slot

Vamos a crear nuestro propio slot y lo llamaremos países, como se puede ver en la Figura A.9 hemos definido dentro de nuestro slot dos países, Francia e Italia, a los que también hemos asociado un ID, de esta manera cuando utilicemos nuestro slot en un intent, a nivel de código podremos utilizar el ID asociado.

Figura A.9: Pantalla de definición de un slot

A.2. Code

Al igual que pasaba con Build, en code tenemos que comentar las opciones de guardado y despliegue antes de comenzar, para dejar muy claro que aunque guardemos los cambios en el código, si no lo desplegamos (botón deploy), estos no surtirán efecto en la *skill*.

Para comenzar vamos a hablar sobre los archivos que vienen por defecto con la *skill*

Figura A.10: Opciones de guardado y despliegue.

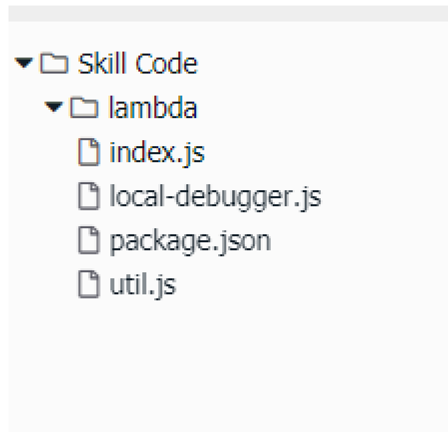

Figura A.11: Archivos predefinidos en la *skill*.

A.2.1. util.js

El archivo de util.js contiene información relevante respecto a los sistemas de almacenamiento que ofrece aws por defecto. No entran dentro del alcance del proyecto.

A.2.2. package.json

El archivo de package.json es el que define el paquete que estamos desarrollando (recordemos que estamos desarrollando en node.js), es aquí donde daremos nombre, marcaremos la versión actual de desarrollo, y lo más importante, donde se define que archivo funciona como main. Es aquí también donde podremos cargar las dependencias, podemos ver que vienen tres paquetes configurados por defecto, con los paquetes de ask-sdk-model y ask-sdk-core deberíamos tener suficiente, pero para evitar errores podemos cargar ask-sdk.

A.2.3. local-debugger.js

Esta funcionalidad está obsoleta como se indica dentro del propio archivo.

A.2.4. index.js

El archivo más importante de todos, en el se recoge las peticiones enviadas por la parte vocal de Alexa, se define el procesamiento que recibe cada petición, y se configura y envía el mensaje de salida que Alexa deberá emitir, vamos a explicar el contenido por partes.

Lo primero de todo es definir que librería se va a encargar de gestionar la construcción de la *skill*, en este caso es la librería *ask-sdk-core* .

Lo siguiente que vemos en el archivo es que existen distintos bloques, estos bloques son los manejadores de cada petición, como podemos ver tenemos más manejadores que intents definidos en build, esto se debe a que existen peticiones que no tienen un intent asociado, como son las peticiones de arranque de la *skill*, y de terminación de esta.

Todos los bloques comparten una misma estructura, en la primera parte de esta estructura (función *canHandle*) se comprueba el tipo de petición que se ha recibido, y cada manejador devuelve si acepta ese tipo de petición o no.

Si el manejador es capaz de aceptar el tipo de petición y de intent entonces ejecutará la segunda parte de la estructura (función *handle*).

El *handlerInput* que se recibe en ambas funciones es el paquete recibido desde el componente vocal de Alexa con toda la información asociada, dentro podremos encontrar desde la información del dispositivo con el que se ha realizado la escucha, así como toda la información sobre el intent que se ha detectado, los slots que lo componen y la información que se ha obtenido sobre ellos. Es a partir de este paquete de donde nosotros vamos a sacar las variables necesarias para el funcionamiento de nuestra *skill*.

Cuando hayamos acabado el procesamiento de los datos es hora de devolver una respuesta al usuario, para ello utilizaremos el constructor de respuestas que viene incluido en el paquete recibido desde el componente vocal.

Con *handlerInput.responseBuilder* vamos a poder definir el mensaje de respuesta en .speak, el mensaje está codificado en ssml, y es una variable de texto string.

También podemos mostrar información en una pantalla si el dispositivo es compatible con .withSimpleCard .

Por último debemos cerrar siempre con .getResponse()

Para que la función que maneja un intent funcione debemos recordad que hay que exportarla, y para conseguirlo solo tenemos que añadir el nombre de la función en el manejador de exportación que se encuentra al final del archivo, incluimos el nombre de nuestra función igual que están el resto de funciones.

A.3. test

En este apartado podemos probar el correcto funcionamiento de nuestra *skill* desde el navegador, habilitando las opciones de desarrollo, marcando la opción development como se ve en la Figura A.12

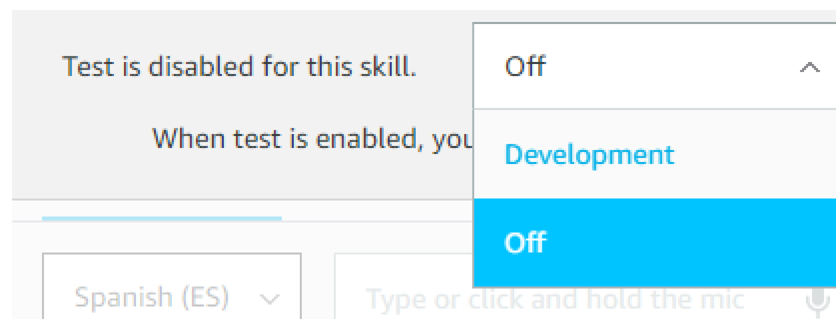

Figura A.12: Activación de testeo

Podremos escribir las sentencias simulando un usuario tipo, o incluso emitirlas de viva voz pulsando en el micrófono. El sistema nos responderá como lo haría en una interacción normal,

y a la derecha podremos ver una simulación de pantalla en el caso de que se devuelva algo por esta. Si activamos la opción de Device Log, podremos ver más abajo todos los paquetes que genera Alexa, y el contenido de los mismos, lo que nos puede ayudar a la hora de debuggear nuestra *skill*, o directamente en el desarrollo de la misma.

Anexo B

Tutorial: Ampliación de la skill

Para entender un poco mejor el proceso de desarrollo de una *skill*, vamos a crear nuestro propio intent dentro de la *skill* explicada en el Apéndice A, como forma de ampliación sencilla.

B.1. Diseño vocal

Utilizando la idea sobre una agencia de viajes, vamos a plantear un intent que nos devuelva información sobre los viajes disponibles, comenzaremos creando el intent, para ello nos situamos en la sección de Build, y dentro de esta vamos a la pestaña de Interaction Model. Dentro de la misma, si pulsamos en Intents (opción que es a la vez menú desplegable y sección) veremos que a la derecha nos carga el listado de los intents que tenemos generados, vamos a darle al botón azul de *+Add Intent*

En la siguiente ventana (Figura B.1) veremos que podemos elegir entre crear un intent personalizado (custom intent) o usar uno existente de los predefinidos, nosotros vamos a elegir crear uno personalizado con el nombre Viajes.

Figura B.1: Selección del tipo de intent

Ahora llega el momento de plantearse que queremos ofrecer exactamente al usuario, que

información necesitamos de este para conseguirlo y como va a interactuar el usuario con nuestro intent. La respuesta a la primera pregunta ya la tenemos, vamos a ofrecer al usuario información sobre rutas de viajes.

Respecto a la segunda pregunta: ¿Qué necesitamos para conseguirlo? Dado que vamos a ofrecer información de rutas, lo primero será conocer el origen y el destino, por lo que ya tenemos nuestros dos primero slots, que llamaremos respectivamente origen y destino (el tipo del slot lo escogeremos un poco más tarde). Lo siguiente que necesitamos saber es cuando desea viajar el usuario, en este ejemplo vamos a reducir la ventana de tiempo a los días de la semana, por lo que tenemos un tercer slot que será fecha.

Ahora que tenemos los slots necesarios tenemos que definir de que tipo son, como somos una agencia de viajes pequeña, solo tenemos rutas desde Francia a Italia y viceversa, y ese tipo de slot ya lo hemos creado en Apéndice A, con el nombre de países, por lo que se lo asignamos a Origen y Destino. Para el slot de fecha vamos a utilizar uno de los ya existentes y seleccionaremos AMAZON.DayOfWeek, que agrupa los días de la semana representados por su nombre.

De los tres slots que hemos definido vamos a considerar que solo Origen y Destino son imprescindibles, ya que al usuario puede no importarle viajar un día u otro, o quiere conocer los trayectos que existen antes de decidir. Para conseguir que Origen y Destino siempre se completen vamos a marcarlos como necesarios de confirmar, para ello vamos a pulsar sobre *Edit Dialog* que se encuentra en la columna ACTIONS de los slots, como se ve en Figura B.2.

NAME	SLOT TYPE	ACTIONS
origen	paises	Edit Dialo
destino	paises	Edit Dialo

Figura B.2: Slots que hemos generado para el intent de Viajes

En la nueva ventana veremos que tenemos dos activadores apagados, uno para Slot filling, que es el que indicará que nuestro slot debe ser completado para continuar, y Slot confirmation. que sirve para repetir al usuario la información que hemos recogido para el slot para que este confirme si es correcta o no.

Encendemos Slot filling, lo que hará que aparezcan dos campos nuevos, el primero es para las utterances de pregunta hacia el usuario (para completar el slot), y el segundo son las utterances que esperamos del usuario como respuesta. Como estamos en Origen, dentro del primer campo realizaremos preguntas del estilo:

- ¿Desde qué país desea comenzar el viaje?
- ¿Cuál es el país de partida?

Cuantas más sentencias distintas se planteen más "real" será para el usuario la experiencia en caso de que repita el uso de la *skill*, ya que no estaremos anclados a un único tipo de pregunta (como pasa en los sistemas visuales, ya que el texto que se define para un label o

un botón, por lo general es estático).

Pero hay que tener en cuenta que cuantas más formas tengamos de preguntar, más formas tendrá el usuario de contestar, y es en el siguiente campo donde debemos recogerlas. Para las dos preguntas de ejemplo planteadas hemos planteado que las respuestas del usuario puedan ser las siguientes:

- {origen}
- Desde {origen}
- Saldré desde {origen}
- Saldremos desde {origen}
- Partiré desde {origen}
- Partiremos desde {origen}
- Comenzaré en {origen}
- Comenzaremos en {origen}

Muy importante tener en cuenta la ortografía, ya que en el idioma español hay palabras que pueden tener distinto significado dependiendo de la puntuación, y que Alexa transcribe todos los signos de acentuación, por lo que puede afectar a la hora de recoger la información.

Realizaremos un trabajo similar con el slot de Destino.

Con esto quedaría resuelta nuestra segunda pregunta, vamos a ver como contestamos a la tercera: ¿Cómo va a interactuar el usuario con nuestro intent?

Para contestar a esta pregunta tenemos que saber como puede llegar el usuario hasta el intent. En el Capítulo 5 vimos que existen dos maneras de inicializar una *skill*, y cada una de estas maneras implica que el usuario emitirá unas sentencias u otras.

Si tomamos por ejemplo la inicialización sin petición, una vez el usuario a arrancado la *skill*, lo siguiente que puede hacer es emitir una orden en lugar de una pregunta o petición, por lo que las utterances que podemos esperar serán del tipo:

- Dame las rutas entre {origen} y {destino} para el {fecha}
- Dame las rutas entre {origen} e {destino} para el {fecha}
- Dame las rutas para el {fecha} entre {origen} y {destino}
- Dame las rutas para el {fecha} entre {origen} e {destino}
- Dame las rutas para {origen}
- Dame las rutas para {fechas}

Aunque existen muchísimas más formas nosotros nos vamos a quedar con estas de momento.

Ahora vamos a ver como se llamaría al intent si hacemos una inicialización con petición:

- qué rutas hay entre {origen} y {destino}

- qué rutas hay desde {origen} a {destino} el {fecha}

De nuevo existen más variaciones de pregunta, pero no nos vamos a extender aquí en ello.
Con esto quedaría cerrada la parte vocal de nuestro nuevo intent, así que lo guardamos y lo compilamos.

B.2. Código

A continuación vamos a generar el código que lo hará funcionar, pero no vamos a contemplar todo el funcionamiento, desarrollaremos a modo de ejemplo la respuesta para algunas utterances del intent y se deja el desarrollo del resto como trabajo personal para comprobar si se ha entendido la explicación.
Como el funcionamiento del código que Alexa nos ofrece por defecto en lambda ya se ha explicado en Apéndice A, por lo que avanzaremos relativamente rápido por esta sección.

Lo primero que debemos hacer es crear el manejador de nuestro intent en el archivo index.js, para ello podemos tomar como ejemplo el manejador el intent Hello World, y sustituir el nombre del manejador y del intent que espera lo que resultaría en un código similar al de la Figura B.4
Lo siguiente será extraer los slots del paquete que recibimos de Alexa *handlerInput*, Este paquete contiene mucha información que para nosotros ahora mismo no es relevante, por lo que vamos a acotar el contenido del mismo creando una nueva variable llamada *me* y almacenando en esta el objeto *handlerInput.requestEnvelope.request*, el contenido de este objeto tiene una estructura similar a la de la Figura B.3.

Esta estructura de ejemplo corresponde a un utterance que ha aportado información para los tres slots de los que disponemos, y para los slots de origen y destino se confirma que la información coincide con alguno de los valores que hemos definido anteriormente.
Suponiendo que este esta es la petición que recibimos lo siguiente que deberíamos hacer es extraer el país de origen y el de destino, así como la fecha, para ello crearemos nuevas variables (origen, destino y fecha respectivamente) de forma que el código quedaría como se ve en la Figura B.5
Ahora que ya tenemos los datos, podemos manejarlos como queramos, en este ejemplo que hemos desarrollado vamos a realizar un simple if-else que de como resultado que existen viajes de Francia a Italia los lunes y martes, y de Italia a Francia los jueves y viernes, y lo plasmamos en una cadena de texto que será la que posteriormente enviemos a la interfaz de voz. El código resultante se puede ver en la Figura B.6

Por último preparamos el paquete de respuesta y lo enviamos como se puede ver en la Figura B.7
Recordemos que hay que incluir el nombre de la función en el manejador de exportaciones que se encuentra al final del archivo.
Todo el código de la *skill* se puede encontrar en el siguiente repositorio: `https://github.com/lvareo/AmpliacionHolaMundo.git`

```
1  "request": {
2      "type": "IntentRequest",
3      "requestId": "amzn1.echo-api.request.d88337c9-b3c5-4409-b711-6640c69ac7de",
4      "locale": "es-ES",
5      "timestamp": "2020-11-12T18:24:53Z",
6      "intent": {
7          "name": "Viajes",
8          "confirmationStatus": "NONE",
9          "slots": {
10             "fecha": {
11                 "name": "fecha",
12                 "value": "martes",
13                 "confirmationStatus": "NONE",
14                 "source": "USER"
15             },
16             "origen": {
17                 "name": "origen",
18                 "value": "francia",
19                 "resolutions": {
20                     "resolutionsPerAuthority": [
21                         {
22                             "authority": "amzn1.er-authority.echo-sdk.amzn1.ask.skill.e17f3b8b-5b74-40fe-bd51-a3989f6c65ed.paises",
23                             "status": {
24                                 "code": "ER_SUCCESS_MATCH"
25                             },
26                             "values": [
27                                 {
28                                     "value": {
29                                         "name": "Francia",
30                                         "id": "FR"
31                                     }
32                                 }
33                             ]
34                         }
35                     ]
36                 },
37                 "confirmationStatus": "NONE",
38                 "source": "USER"
39             },
40             "destino": {
41                 "name": "destino",
42                 "value": "italia",
43                 "resolutions": {
44                     "resolutionsPerAuthority": [
45                         {
46                             "authority": "amzn1.er-authority.echo-sdk.amzn1.ask.skill.e17f3b8b-5b74-40fe-bd51-a3989f6c65ed.paises",
47                             "status": {
48                                 "code": "ER_SUCCESS_MATCH"
49                             },
50                             "values": [
51                                 {
52                                     "value": {
53                                         "name": "Italia",
54                                         "id": "IT"
55                                     }
56                                 }
57                             ]
58                         }
59                     ]
60                 },
61                 "confirmationStatus": "NONE",
62                 "source": "USER"
63             }
64         }
65     },
66     "dialogState": "COMPLETED"
67 }
```

Figura B.3: Estructura de las peticiones recibidas desde Alexa

```
const ViajesIntentHandler = {
    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) === 'IntentRequest'
            && Alexa.getIntentName(handlerInput.requestEnvelope) === 'Viajes';
    },
```

Figura B.4: Primera parte del código del manejador del intent Viajes

```
var me = handlerInput.requestEnvelope.request;
var origen = me.intent.slots['origen'].resolutions.resolutionsPerAuthority[0].values[0].value.id;
var destino = me.intent.slots['destino'].resolutions.resolutionsPerAuthority[0].values[0].value.id;
var fecha = me.intent.slots['fecha'].value;
```

Figura B.5: Segunda parte del código del manejador del intent Viajes

```
var speakOutput = '';

if(origen==='FR'){
    if(destino==='IT'){
        if(fecha==='lunes' || fecha==='martes'){
            speakOutput = 'Contamos con dos rutas disponibles desde Francia a Italia el ' + fecha;
        }
        else{
            speakOutput = 'No contamos con ninguna ruta disponible entre Francia e Italia para el' + fecha;
        }
    }
}else if(origen==='IT'){
    if(destino==='FR'){
        if(fecha==='jueves' || fecha==='viernes'){
            speakOutput = 'Contamos con tres rutas disponibles desde Italia a Francia el ' + fecha;
        }
        else{
            speakOutput = 'No contamos con ninguna ruta disponible entre Italia y Francia para el' + fecha;
        }
    }
}
```

Figura B.6: Tercera parte del código del manejador del intent Viajes

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

Figura B.7: Cuarta parte del código del manejador del intent Viajes

Bibliografía

[1] Michael F McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys (CSUR)*, 34(1):90–169, 2002.

[2] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966.

[3] Jeremy Peckham. Speech understanding and dialogue over the telephone: an overview of the esprit sundial project. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*, 1991.

[4] Stephanie Seneff. Robust parsing for spoken language systems. In *icassp*, pages 189–192. IEEE, 1992.

[5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing.* Draft available at `https://web.stanford.edu/~jurafsky/slp3/`, 2019.

[6] Alan Ritter, Sam Clark, Oren Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 1524–1534, 2011.

[7] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364*, 2015.

[8] Amazon alexa. `https://developer.amazon.com/es-ES/alexa`, Visitado por última vez el 12/11/2020.

[9] Asistente de google. `https://assistant.google.com/intl/es_es/`, Visitado por última vez el 12/11/2020.

[10] Microsoft cortana. `https://www.microsoft.com/en-us/cortana`, Visitado por última vez el 12/11/2020.

[11] Siri. `https://www.apple.com/siri/`, Visitado por última vez el 12/11/2020.

[12] Google duplex llega a españa. `https://bit.ly/2GWFS1c`, Visitado por última vez el 12/11/2020.

[13] Windows 10 mobile wikipedia. `https://en.wikipedia.org/wiki/Windows_10_Mobile`, Visitado por última vez el 12/11/2020.

[14] Daniel G Bobrow, Ronald M Kaplan, Martin Kay, Donald A Norman, Henry Thompson, and Terry Winograd. Gus, a frame-driven dialog system. *Artificial intelligence*, 8(2):155–173, 1977.

[15] D. Gibbon, R. Moore, and R. Winski. *Handbook of Standards and Resources for Spoken Language Systems.* Mouton de Gruyter, 1997.

[16] Alan Ritter, Colin Cherry, and William B Dolan. Data-driven response generation in social media. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 583–593, 2011.

[17] Ramón López-Cózar, Zoraida Callejas, David Griol, and Jose Quesada. Review of spoken dialogue systems. *Loquens*, 1:e012, 02 2015.

[18] Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan. A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*, 2016.

[19] Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.

[20] Sending email using amazon ses. `https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/ses-examples-sending-email.html`,Visitado por última vez el 12/11/2020.

[21] Sam Agnew. 5 ways to make http requests in node.js. `https://www.twilio.com/blog/2017/08/http-requests-in-node-js.html`,Visitado por última vez el 12/11/2020.

[22] Nick Carneiro. Curl converter. `https://curl.trillworks.com/`,Visitado por última vez el 12/11/2020.

[23] Matthew B. Hoy. Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical Reference Services Quarterly*, 37(1):81–88, 2018. PMID: 29327988.

[24] Cortana wikipedia. `https://en.wikipedia.org/wiki/Cortana`, Visitado por última vez el 12/11/2020.