Universitat Oberta de Catalunya

uoc.edu

APP DE LA UOC PARA LA IA ALEXA

Álvaro Rodríguez-Moroy Porcel
Grado de Ingeniería Informática
Desarrollo Web

Gregorio Robles Martínez
Santi Caballe Llobet

12/06/2019

Reconocimiento-NoComercial-CompartirIgual 3.0 España (CC BY-NC-SA 3.0 ES)

FITXA DEL TREBALL FINAL

Título del trabajo:	*APP DE LA UOC PARA LA IA ALEXA*
Nombre del autor:	*Álvaro Rodríguez-Moroy Porcel*
Nombre del consultor/a:	Gregorio Robles Martínez
Nombre del profesor:	Santi Caballe Llobet
Fecha de Entrega (mm/aaaa):	*06/2019*
Titulación o programa:	Grado de Ingeniería Informática
Área del Trabajo Final:	Desarrollo Web
Idioma del trabajo:	*Castellano*
palabras clave	*IA, Alexa, Aplicación*

Resumen del Trabajo (máximo 250 palabras): Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo

Este trabajo presenta la necesidad de evolución de la UOC para adaptarse a las nuevas tecnologías. Esta necesidad nace de la inclusión de inteligencias artificiales (IA) capaces de interactuar con los usuarios mediante la voz.

Estas IAs pretenden ayudar a los usuarios a tener información o realizar acciones simplemente pidiéndolas. Por ejemplo, un usuario podría saber sus notas o las fechas de entrega de sus actividades a través de ellas.

En el caso de este trabajo, la IA elegida es Alexa, de Amazon. Y se ha realizado una aplicación que consta de dos partes, la creación de la aplicación de Alexa por un lado y los servicios de backend por otro lado.

El objetivo es mostrar cómo debe crearse una aplicación de este tipo y la mejora que ofrecería a los alumnos. Todos los servicios de backend son una simulación de conexión con los servicios de la UOC.

La metodología escogida para conseguir un producto de calidad y cumplir los plazos de entrega ha sido Kanban y para el desarrollo ha sido TDD, así como la utilización del IDE Eclipse y el repositorio de GitHub para albergar el código.

El resultado ha sido un producto de calidad y funcional al 100% donde

se muestran buenas prácticas, diferentes opciones para “jugar” con la IA de Alexa y herramientas de apoyo para la mejora del desarrollo.

Como conclusión se puede destacar la necesidad de integrarse con esta nueva tecnología y abrir un abanico de posibilidades que aportarían más caché aún a una universidad instaurada en las nuevas tecnologías.

Abstract (in English, 250 words or less):

This paper presents the need for an evolution of the UOC to adapt to new technologies. This need arises from the inclusion of artificial intelligences (AI) capable of interacting with users through voice.

These IAs are intended to help users to get information or perform actions simply by asking for them. For example, a user could know their grades or the delivery dates of their activities through them.

In the case of this work, the chosen IA is Alexa, from Amazon. And an application has been made that consist of two parts, the creation of the Alexa application on the one hand and backend services on the other.

The objective is to show how an application of this type should be created and the improvement it would offer to students. All the backend services are a simulation of connection with the services of the UOC.

The methodology chosen to achieve a quality product and meet the delivery deadlines was Kanban and for the development was TDD, as well as the use of the Eclipse IDE and the GitHub repository to host the code.

The result has been a 100% quality and functional product where good practices are shown, different options to "play" with the Alexa AI and support tools to improve development.

In conclusion, we can highlight the need to integrate with this new technology and open a range of possibilities that would bring more cache even to a university established in new technologies.

Índex

1. INTRODUCCIÓN 8

1.1. CONTEXTO Y JUSTIFICACIÓN DEL TRABAJO 8
1.2. OBJETIVOS DEL TRABAJO 9
1.2.1. Objetivo general 9
1.2.2. Objetivos específicos 9
1.3. ENFOQUE Y MÉTODO SEGUIDO 10
1.3.1. Producto 10
1.3.2. Kanban 10
1.3.3. TDD (Test-driven development) [3] 11
1.4. PLANIFICACIÓN DEL TRABAJO 11
1.5. BREVE SUMARIO DE PRODUCTOS OBTENIDOS 13
1.6. BREVE DESCRIPCIÓN DE LOS OTROS CAPÍTULOS DE LA MEMORIA 13

2. CASOS DE USO 15

2.1. CASO DE USO DE FLASH BRIEFING 15
2.1.1. Objetivo 15
2.1.2. Diagrama de flujo 15
2.1.3. Prerrequisitos 16
2.1.4. Flujo de éxito 16
2.1.5. Flujos alternativos de error 16
2.1.6. Información adicional 16
2.2. CASO DE USO DE "LOGIN" 17
2.2.1. Objetivo 17
2.2.2. Diagrama de flujo 18
2.2.3. Prerrequisitos 18
2.2.4. Flujo de éxito 18
2.2.5. Flujos alternativos de error 19
2.2.6. Información adicional 19
2.2.7. Utterances (frases) 19
2.3. CASO DE USO DE "PEDIR NOTAS" 20
2.3.1. Objetivo 20
2.3.2. Diagrama de flujo 20
2.3.3. Prerrequisitos 20
2.3.4. Flujo de éxito 20
2.3.5. Flujos alternativos de error 21
2.3.6. Información adicional 21
2.3.7. Utterances (frases) 22
2.4. CASO DE USO DE "CALENDARIO" 22
2.4.1. Objetivo 22
2.4.2. Diagrama de flujo 22
2.4.3. Prerrequisitos 22
2.4.4. Flujo de éxito 23
2.4.5. Flujos alternativos 23
2.4.6. Flujos alternativos de error 23
2.4.7. Información adicional 24
2.4.8. Utterances (frases) 24

3. CREACIÓN DEL PROYECTO REST 24

3.1. CONECTAR EL PROYECTO AL GITHUB 26

4. CREACIÓN DE LA PRIMERA SKILL 32

5. CREACIÓN DEL SERVICIO REST DEL FLASH BRIEFING 34

6. CREACIÓN DE LA CUSTOM SKILL 37

6.1. INVOCATION NAME 38
6.2. INTENTS (FUNCIONALIDADES) EN ALEXA 39
6.3. SLOTS (VARIABLES) Y SLOT TYPES 42
6.4. EJEMPLO 43
6.5. INTENTS Y LOS SERVICIOS REST 44
6.6. ACCESO AL SERVICIO REST DESDE ALEXA 45
6.7. ERRORES Y SOLUCIONES 46

7. CREACIÓN DEL BACKEND DE LA CUSTOM SKILL 47

7.1. PRIMERA PRUEBA 47
7.2. HANDLERS DE ALEXA 50
7.3. SIGUIENDO TDD 51

8. TEST DESDE LA CONSOLA DE ALEXA 53

9. INTENTS (FUNCIONALIDADES) DE LA APLICACIÓN 55

9.1. CONFIGURACIONES DE RESPUESTA DE ALEXA 56
9.1.1. Efectos 56
9.1.2. Sonidos 57
9.1.3. Comportamiento de ALEXA 59
9.2. LOGININTENT 60
9.3. NOTASINTENT 63
9.4. CALENDARIOINTENT 66
9.5. NOINTENTHANDLER 68

10. CONECTAR SKILL EN DESARROLLO CON EL DISPOSITIVO ECHO 69

11. PROYECTO JAVA EN ECLIPSE 72

11.1.1. Estructura del proyecto 73
11.1.2. Buenas prácticas 78

12. COSTE DEL PROYECTO 80

13. ARQUITECTURA DE REFERENCIA 81

14. CONCLUSIONES 83

15. GLOSARIO 84

16. BIBLIOGRAFÍA 86

17. ANEXOS 87

17.1. CREACIÓN DE UNA CUENTA DE ESTUDIANTE DE GITHUB SDP 87
17.2. PIVOTAL 87
17.3. INSTALACIÓN DE LOMBOK 87

Lista de figuras

Ilustración 1 - Tablero trello 1 8
Ilustración 2 - Tablero de Trello 2 10
Ilustración 3 - Diagrama Gantt - TFG-Alexa – Completo [8] 11
Ilustración 4 - Diagrama Gantt - TFG-Alexa – PEC1 [8] 11
Ilustración 5 - Diagrama Gantt - TFG-Alexa – PEC2 [8] 12
Ilustración 6 - Diagrama Gantt - TFG-Alexa – PEC3 [8] 12
Ilustración 7 - Diagrama Gantt - TFG-Alexa – PEC4 [8] 12
Ilustración 8 - Caso de uso Flash Briefing [4] 15
Ilustración 9 - Noticias UOC 1 17
Ilustración 10 - Noticias UOC 2 17
Ilustración 11 - Caso de uso de Login [4] 18
Ilustración 12 - Caso de uso de Pedir Notas [4] 20
Ilustración 13 - Caso de uso de Pedir Notas [4] 22
Ilustración 14 - Instalación plugin STS 24
Ilustración 15 - Creación proyecto Spring Boot 25
Ilustración 16 - Propiedades del proyecto rest 25
Ilustración 17 - Repositorio Github 26
Ilustración 18 - Conexión Eclipse - Github 26
Ilustración 19 - Eclipse con proyecto de Github 27
Ilustración 20 - Eclipse sincronizando proyecto 27
Ilustración 21 - Eclipse mostrando cambios 28
Ilustración 22 - Eclipse proyecto en Git Bash 28
Ilustración 23 - Github login 29
Ilustración 24 - Eclipse comandos git 29
Ilustración 25 - Eclipse comando git 3 30
Ilustración 26 - Eclipse comandos git 4 30
Ilustración 27 - Eclipse comandos git 2 30
Ilustración 28 - Eclipse comandos git 5 31
Ilustración 29 - Eclipse comandos git 6 31
Ilustración 30 - Github proyecto 32
Ilustración 31 - Github proyecto 2 32
Ilustración 32 - Primera Skill 32

Ilustración 33 - Selección tipo de skill 33
Ilustración 34 - Dashboard Flash Briefing 33
Ilustración 35 - Dashboard Flash Briefing 2 34
Ilustración 36 - Mensaje de error 34
Ilustración 37 - Mensaje de advertencia 34
Ilustración 38 - Primera prueba con TDD 35
Ilustración 39 - Primer controlador 35
Ilustración 40 - Clase de literales 36
Ilustración 41 - Controlador refactorizado 36
Ilustración 42 - Controlador adecuado para Alexa 37
Ilustración 43 - Respuesta controlador 37
Ilustración 44 - Consola Alexa - Tipos de Skill 38
Ilustración 45 - Consola Alexa - Configuracion Custom Skill 38
Ilustración 46 - Consola Alexa - Invocation Name 39
Ilustración 47 - Consola Alexa - Intents 39
Ilustración 48 - Consola Alexa - Add Intent 40
Ilustración 49 - Consola Alexa - Utterances 41
Ilustración 50 - Consola Alexa - Account Linking 41
Ilustración 51 - Consola Alexa - LoginIntent 42
Ilustración 52 - Consola Alexa – LoginIntent - utterances 42
Ilustración 53 - Consola Alexa - Slots 43
Ilustración 54 - Consola Alexa - Slot Types 43
Ilustración 55 - Consola Alexa - Success Message 44
Ilustración 56 - Consola Alexa - Endpoint 45
Ilustración 57 - Consola Alexa - Test 46
Ilustración 58 - Consola Alexa - Error Message 46
Ilustración 59 - Consola Alexa - Invocation Name Solution 46
Ilustración 60 - Consola Alexa - Slot Values 47
Ilustración 61 - Eclipse - TDD 48
Ilustración 62 - Eclipse - TDD - Result 48
Ilustración 63 - Eclipse - TDD - Refactoring 49
Ilustración 64 - Eclipse - TDD - Result 2 49
Ilustración 65 - Eclipse - TDD - Refactoring 2 50
Ilustración 66 - Eclipse - TDD - Result 3 50
Ilustración 67 - Eclipse - TDD - Refactoring 3 51

Ilustración 68 - Eclipse - TDD - Refactoring 4 52
Ilustración 69 - Eclipse - TDD - Refactoring 5 53
Ilustración 70- Consola Alexa - Test 2 53
Ilustración 71- Consola Alexa - Test 3 54
Ilustración 72- Consola Alexa - Test 4 54
Ilustración 73- Consola Alexa - Test 5 55
Ilustración 74 - Alexa Documentation - Effects 57
Ilustración 75 - Alexa Documentation - Sounds 58
Ilustración 76 - Alexa Documentation - Sounds 2 59
Ilustración 77- Eclipse - Code 61
Ilustración 78 - Consola Alexa - Test Login 62
Ilustración 79 - Consola Alexa - Test Login 2 63
Ilustración 80 - Consola Alexa - NotasIntent 63
Ilustración 81 - Consola Alexa - Slot Asignatura 64
Ilustración 82 - Eclipse - Code 2 65
Ilustración 83 - Consola Alexa - Test Notas 65
Ilustración 84 - Consola Alexa - Test Notas 2 66
Ilustración 85 - Consola Alexa - CalendarioIntent 67
Ilustración 86 - Consola Alexa - Test Calendario 68
Ilustración 87 - Consola Alexa - NoIntent 68
Ilustración 88 - Consola Alexa - Test NoIntent 69
Ilustración 89 - Consola Alexa, pestaña "Distribution" [6] 70
Ilustración 90 - Consola Alexa - Distribution Save 70
Ilustración 91 - Consola Alexa - Privacy 71
Ilustración 92 - Consola Alexa - Availability 71
Ilustración 93 - App Alexa - UOC_Amoroy 72
Ilustración 94 - Consola Alexa - BetaTest 72
Ilustración 95 - Eclipse - Project 73
Ilustración 96 - Eclipse - Project 2 74
Ilustración 97 - Eclipse - Project 3 77
Ilustración 98 - Eclipse - Coverage 78
Ilustración 99 - Eclipse - Coverage 2 79
Ilustración 100 - Eclipse - Coverage 3 79
Ilustración 101 - Eclipse - JavaDoc 79
Ilustración 102 - Arquitectura de referencia (Microservicios) [4] 82

1. Introducción

NOTA: El documento está vivo hasta su entrega final como memoria del proyecto, por lo que pueden realizarse cambios de una PEC a otra en la estructura y contenido del mismo. Así como contener apartados inacabados.

Los contenidos acabados son los que están dentro del recuadro rojo, en la columna Done:

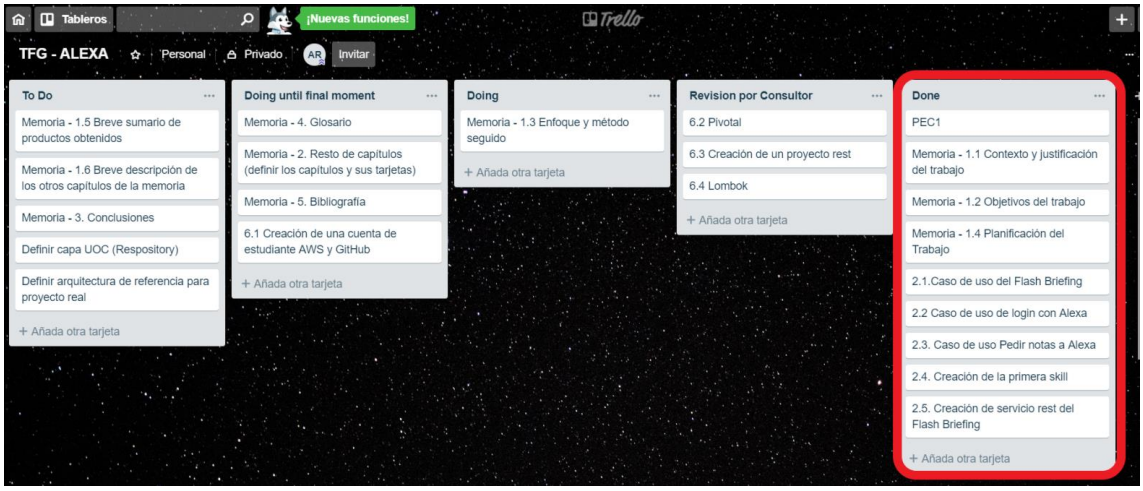


Ilustración 1 - Tablero trello 1

1.1. Contexto y justificación del trabajo

Habla para que yo pueda conocerte (Sócrates)

La sociedad –dejando a un lado la salud y la igualdad– evoluciona hacia un objetivo muy claro; la comodidad. Los seres humanos que tienen cubiertas sus necesidades básicas lo que buscan es vivir de la manera más cómoda posible.

Se podría decir que la comodidad es la “cache” de la sociedad. Así, los inventos más utilizados siempre son aquellos que facilitan realizar las acciones más cotidianas, o incluso realizan estas tareas de manera automática.

Hay ejemplos que van desde las lanzas de la prehistoria (evolución de las piedras) hasta los robots para todo tipo de finalidades (lavar, limpiar, cocinar,…). O incluso objetos tan cotidianos como la fregona que ha evitado que se tenga que limpiar los suelos estando agachados.

Este trabajo quiere acercar un elemento que pronto será tan importante en nuestras vidas como lo es un móvil o una lavadora. De lo que se está hablando es de las inteligencias artificiales. Estas son creadas para facilitar la vida.

En este caso se trata de la inteligencia artificial de Amazon Alexa (a partir de ahora IA o Alexa). Alexa es capaz de ofrecer información de todo tipo en tiempo real de manera auditiva (y visual, dependiendo el dispositivo).

Alexa consigue de esta manera dos hitos, por un lado permite realizar acciones que antes se hacían de manera manual haciendo clics o utilizando las manos, y ahora se permiten hacer con menos esfuerzo físico, es decir, hablando.

Por otro lado, las personas con problemas de movilidad u otro tipo de discapacidad pero que puedan hablar se les abrirá un mundo al que quizás antes no tenían acceso.

Este trabajo final mostrará el proceso de desarrollo de una aplicación para esta inteligencia artificial en la que se obtendrá una aplicación usable con cualquier dispositivo de Alexa (Echo).

1.2. Objetivos del trabajo

1.2.1. Objetivo general

El objetivo general es desarrollar y enseñar una aplicación de Alexa para la UOC en la que un estudiante pueda solicitar información personalizada.

1.2.2. Objetivos específicos

- Crear una documentación que explique cómo se desarrolla una aplicación de Alexa en el lado de Alexa
- Crear una documentación que explique cómo se desarrolla una aplicación de Alexa en el lado del servidor
- Crear un servicio rest al que pueda conectarse Alexa para ofrecer la información del Flash Briefing Skill
- Crear una Skill funcional del tipo *Flash Briefing Skill* en el lado de Alexa
- Crear un servicio rest que permita a Alexa ofrecer información personalizada según la demanda del usuario
- Crear una Skill funcional del tipo *Custom Skill* en el lado de Alexa
    - Crear un Intent que devuelva las notas de una asignatura
    - Crear un Intent que devuelva los eventos del calendario

1.3. Enfoque y método seguido

Al final del documento se puede acceder a un resumen de las buenas prácticas más importantes que se han dado en este proyecto [Buenas prácticas].

1.3.1. Producto

Se ha decidido presentar un producto funcional, es decir, una aplicación orientada a la inteligencia artificial Alexa y una memoria donde se refleje toda la teoría de cómo funciona y por qué se ha hecho así.

También se presentará una arquitectura de referencia en caso de que este aplicativo pudiera llegar a ser un producto real. El producto se irá creando de manera iterativa cumpliendo la planificación gracias a la metodología Kanban.

1.3.2. Kanban

Es importante seguir una metodología de trabajo para conseguir los objetivos de forma iterativa y sin retrasos. Para conseguirlo se ha elegido la metodología Kanban que permitirá ver el estado actual y la evolución del proyecto con un solo vistazo.

Los puntos más importantes de esta metodología son que se pongan tarjetas con las tareas y así poder visualizar que hay que hacer y que está hecho, y que no se tengan en desarrollo más de un número concreto de tareas (en este caso 3).

Una manera de ayudar a seguir con la esta metodología es utilizar la herramienta Trello que permite listar las tareas y establecer los pasos por los que debe pasar cada tarea hasta que se haya finalizado. Así se puede ver el estado del proyecto y de las tareas:

https://trello.com/b/WfYjKr6M/tfg-alexa

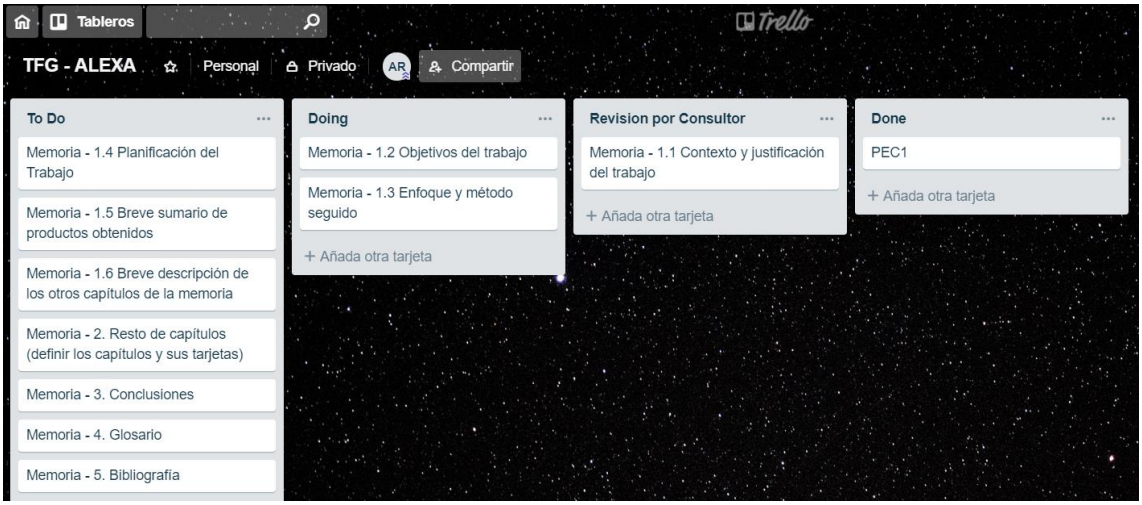


Ilustración 2 - Tablero de Trello 2

1.3.3. TDD (Test-driven development) [3]

Una de las características de la metodología Kanban es producir estrictamente lo necesario y no añadir funcionalidades que no estén en los requisitos, para eso se ha seguido el principio del TDD.

Este principio TDD –desarrollo orientado a pruebas– se ha seguido para el desarrollo de los servicios rest. Mediante esta estrategia de desarrollo se consigue tener una cobertura de código muy alta y se desarrolla únicamente lo que los requisitos necesitan para cumplirse.

1.4. Planificación del Trabajo

El proyecto necesitará utilizar algunas herramientas para optimizar y asegurar la consecución del proyecto. Por un lado habrá que definir una metodología y utilizar las herramientas que mejor se adapten a dicha metodología (punto anterior).

En este caso se ha decantado por la metodología ágil Kanban ya que Scrum está más orientado a equipos de trabajo y constantes contactos con el cliente. Y para la visualización de las tareas del proyecto la herramienta Trello.

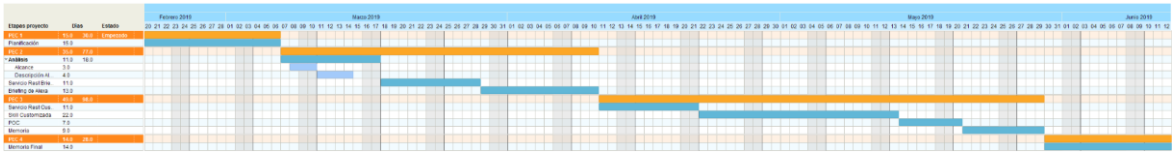


Ilustración 3 - Diagrama Gantt - TFG-Alexa – Completo [8]

Para mostrar mejor la planificación se va a poner la planificación para cada una de las entregas:

Gantt PEC 1

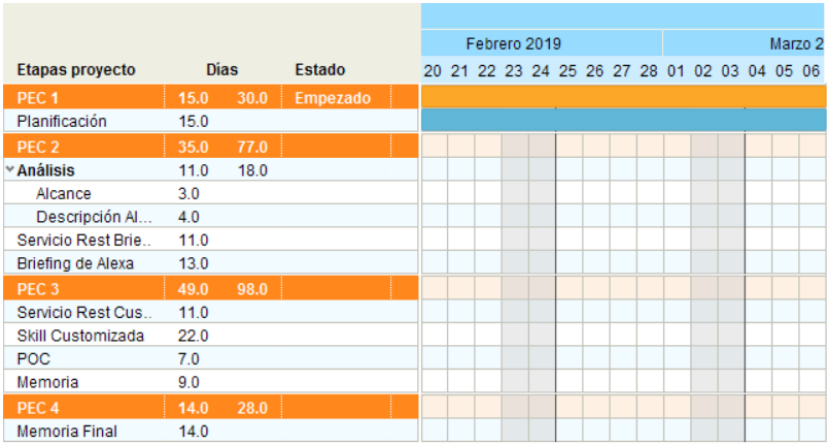


Ilustración 4 - Diagrama Gantt - TFG-Alexa – PEC1 [8]

Gantt PEC 2

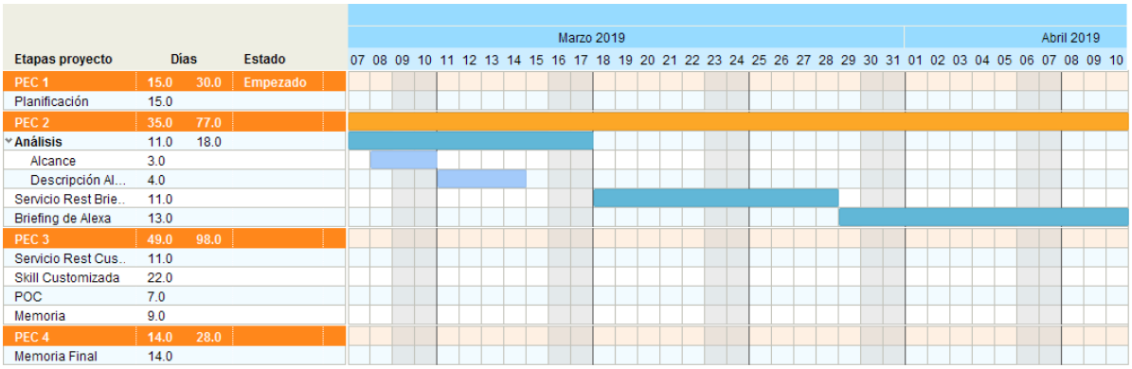


Ilustración 5 - Diagrama Gantt - TFG-Alexa – PEC2 [8]

Gantt PEC 3

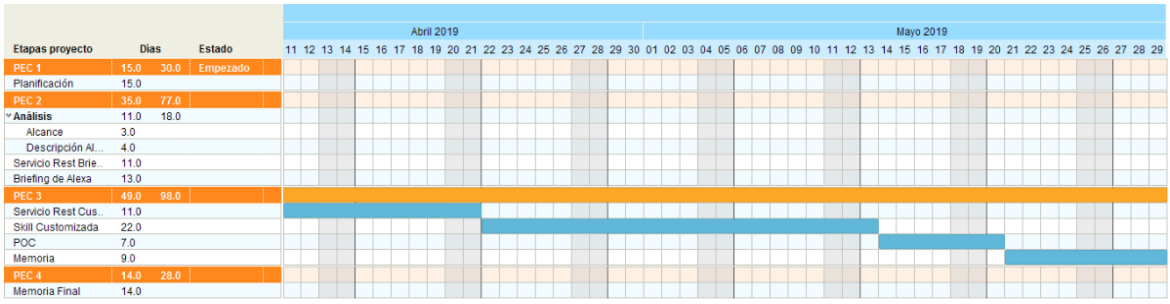


Ilustración 6 - Diagrama Gantt - TFG-Alexa – PEC3 [8]

Gantt PEC 4

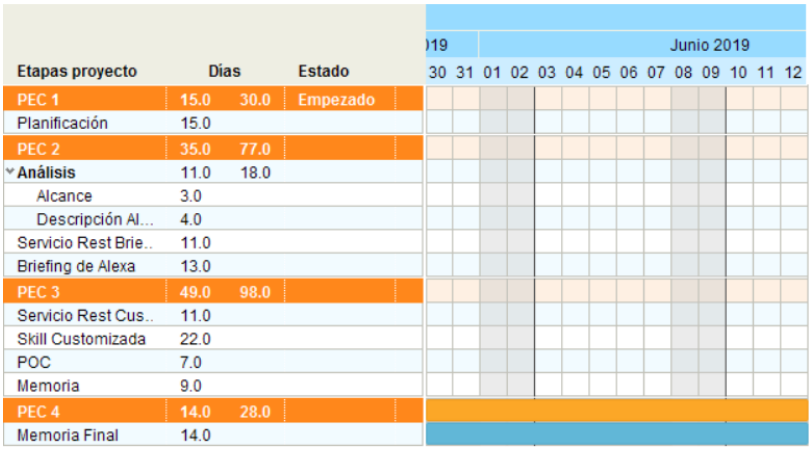


Ilustración 7 - Diagrama Gantt - TFG-Alexa – PEC4 [8]

Así, el primer paso es un documento donde se expone de manera breve la idea y la planificación para llevarla a cabo. En segundo lugar se comienza con el análisis del proyecto, describiendo los puntos que hay que hacer y el alcance del mismo.

Esta segunda fase también llevará el primer servicio rest, en donde habrá que utilizar el servidor Cloud que se haya definido en el análisis, así como la creación del proyecto en el repositorio Github y el IDE Eclipse.

De la misma manera se creará la primera Skill (Flash Briefing) y todo lo relacionado con la creación de la cuenta de Alexa y del propio proyecto. También se iniciará la documentación sobre cómo se desarrolla una aplicación en Alexa, aunque en esta fase no se entrará en mucho detalle.

En tercer lugar se crea el segundo servicio rest así como la Skill personalizada (Custom Skill) para poder interactuar con el usuario y con Alexa. En esta fase se mostrará el modo de prueba que tiene la web de Alexa y se hará una prueba de concepto (POC).

Por último se completará y se perfeccionará la memoria. En esta fase se entrará más a fondo sobre la documentación de las fases anteriores así como añadir todo lo que se considere necesario.

1.5. Breve sumario de productos obtenidos

El producto obtenido es una aplicación 100% funcional para los dispositivos que puedan conectarse con la inteligencia artificial Alexa de Amazon dividido en dos partes.

Estas dos partes son; por un lado una aplicación que deberá desplegarse en un servidor de aplicaciones (en este caso el servidor cloud Pivotal) y hará las funciones de conector entre Alexa y la UOC (esta parte simulada).

Por otro lado, la propia aplicación de Alexa, que es la que los usuarios se instalarían en sus dispositivos Echo, y que se corresponde con toda la configuración propia de la Inteligencia Artificial Alexa.

1.6. Breve descripción de los otros capítulos de la memoria

2. Casos de Uso

Son las definiciones de las funcionalidades. El cómo y el qué tiene que hacer cada funcionalidad. Las funcionalidades serán; “Flash Briefing”, “Login”, “Pedir Notas” y “Calendario”

3. Creación del proyecto rest

Es la creación del proyecto con Spring Boot y la conexión del proyecto con el repositorio de código Github y el control de versiones Git.

4. Creación de la primera Skill

Se explica cómo se crea en la consola de Alexa una Skill (aplicación de Alexa) del tipo Flash Briefing y algunos errores y advertencias que se pueden dar durante la creación.

5. Creación del servicio rest del Flash Briefing

Este capítulo será la implementación de la parte backend para el Flash Briefing, que aunque no tiene mucho peso, sí que se aprovecha para explicar la creación del proyecto en Eclipse y de despliegue en el servidor cloud Pivotal.

6. Creación de la Custom Skill

Esta Skill es la aplicación de Alexa que contendrá todas las funcionalidades que no sean las del Flash Briefing. En este capítulo se aprovecha para entrar en detalle y explicar el funcionamiento de la consola de Alexa, así como las funciones y posibilidades que ofrece.

7. Creación del backend de la Custom Skill

Aqui se explicará cómo se crea la parte backend de las Custom Skill, ya que es algo diferente del Briefing Skill, y se muestra cómo se va creando la primera funcionalidad mediante TDD.

8. Test desde la consola de Alexa

Habrá otro apartado dedicado a las pruebas que permite realizar Alexa desde la consola, ya que son muy potentes y permite simular las mismas pruebas que con el dispositivo Echo.

9. Intents (funcionalidades)

Una vez explicada toda la funcionalidad de la consola de Alexa se explicará las posibilidades que ofrece Alexa para las respuestas, como son los sonidos, los cambios de tono, las pausas,…

De aquí se continúa con los Intents (funcionalidades) y se explica cada uno de ellos con los diferentes resultados y una muestra de la utilización de la parte de test para probar el funcionamiento.

10. Conectar Skill en desarrollo con el dispositivo Echo

Este siguiente paso explica cómo se conecta una skill en desarrollo a un dispositivo Echo real mediante la opción de Beta Test y así probar en dispositivos reales antes de publicarse la aplicación.

11. Proyecto Java en Eclipse

En este capítulo se explicará la estructura del proyecto Java y se hará un breve resumen de las buenas prácticas que se han seguido en este proyecto y en esta memoria.

12. Coste del proyecto

Se hará una valoración de los recursos necesarios y sus costes para llevar a cabo el proyecto real. Aunque solo hablando de recursos como personas y contando con que la UOC ya pondría sus propios servidores.

13. Arquitectura de referencia

Por último, se expondrá una arquitectura de referencia basada en microservicios y Spring Cloud para solventar los problemas de escalabilidad tan importantes en una universidad online.

2. Casos de Uso

2.1. Caso de uso de Flash Briefing

2.1.1. Objetivo

El Flash Briefing debe ofrecer una información al usuario cuando este pida las noticias del día. En este caso ofrecerá la información general sobre la universidad o algún titular de las noticias.

2.1.2. Diagrama de flujo

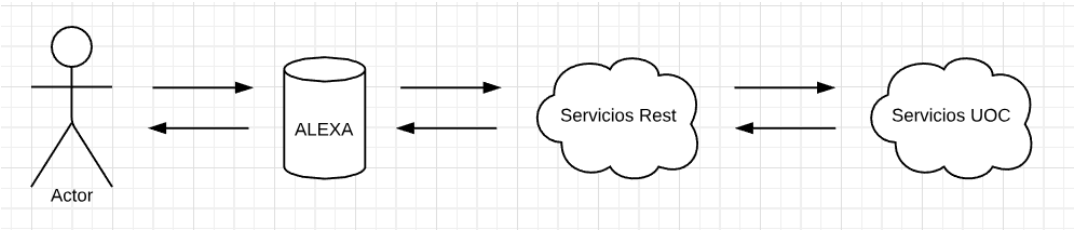


Ilustración 8 - Caso de uso Flash Briefing [4]

- Actor: representa al usuario que se comunicará con la IA ALEXA.
- ALEXA: Alexa es la parte de la IA donde se creará todo lo relacionado con la interacción con el usuario como las frases que entenderá del usuario y a qué servicio llamar.
- Servicios Rest: son la parte de backend encargada de conectar con la UOC para obtener la información que pide el usuario. Estos son los servicios que se van a desarrollar en este trabajo.

- Servicios UOC: representa la parte backend de donde se nutre la aplicación para obtener la información. Para la aplicación sería lo más próximo a la base de datos. En este caso es simulada.

2.1.3. Prerrequisitos

- El usuario debe tener descargada la aplicación de la UOC en su dispositivo de Alexa
- El usuario debe tener el dispositivo de Alexa conectado a Internet

2.1.4. Flujo de éxito

El usuario debe pedir a Alexa que abra la aplicación de la UOC:

1) El usuario le dirá a Alexa la frase “Alexa, dame las noticias del día”
2) Alexa se conectará al servicio rest y obtendrá la frase que deberá decir
3) Alexa dirá la frase devuelta por el servicio rest con la información de la UOC
4) Se acaba el caso de uso

2.1.5. Flujos alternativos de error

- Alexa no entiende la petición
    - Alexa no entiende la petición
    - Devuelve una frase indicando que no lo ha entendido y que vuelva a realizar la petición
    - Se vuelve a comenzar el caso de uso en el paso 1)
- Se produce un error de conexión
    - Se produce un error al intentar conectar con los servicios rest
    - Alexa devuelve una frase indicando que se ha producido un error en la petición
    - Se acaba el caso de uso

2.1.6. Información adicional

La frase que deberá devolver Alexa en este primer servicio será la información oficial de la página (puede tener alguna modificación en forma de mejora) o el titular de una noticia:

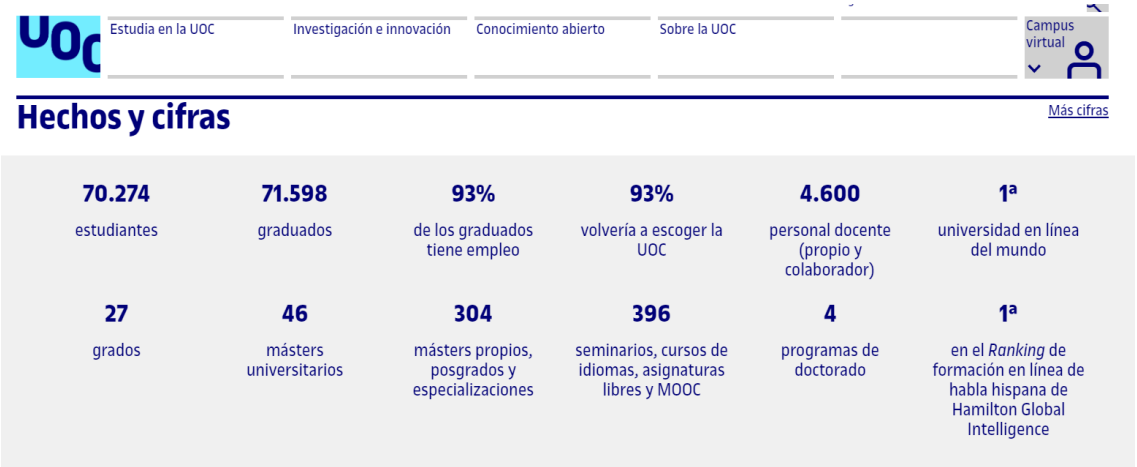


Ilustración 9 - Noticias UOC 1

"*La Universidad Oberta de Cataluña cuenta con 70.274 estudiantes, 71.598 graduados, 93% de los graduados tiene empleo, 93% volvería a escoger la UOC, 4.600 personal docente (propio y colaborador), 1ª universidad en línea del mundo*"

Ilustración 10 - Noticias UOC 2

"Hay que formar profesionales y ciudadanos capaces de afrontar los retos globales de la sociedad. Para descubrir el manifiesto UOC dirigete a www.uoc.edu"

2.2. Caso de uso de "Login"

2.2.1. Objetivo

Alexa permitirá autenticarse contra la UOC (de forma simulada)

2.2.2. Diagrama de flujo

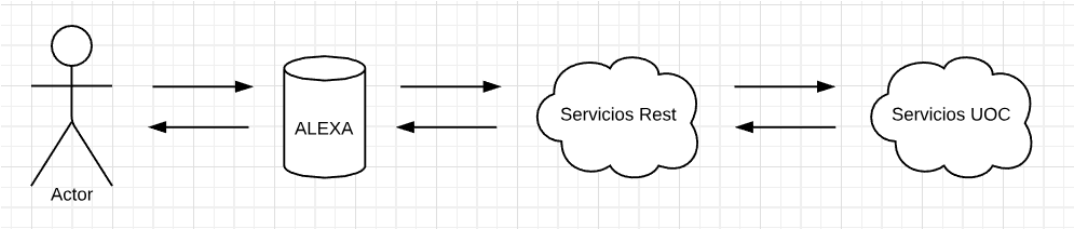


Ilustración 11 - Caso de uso de Login [4]

- Actor: representa al usuario que se comunicará con la IA ALEXA.
- ALEXA: Alexa es la parte de la IA donde se creará todo lo relacionado con la interacción con el usuario como las frases que entenderá del usuario y a qué servicio llamar.
- Servicios Rest: son la parte de backend encargada de conectar con la UOC para obtener la información que pide el usuario. Estos son los servicios que se van a desarrollar en este trabajo.
- Servicios UOC: representa la parte backend de donde se nutre la aplicación para obtener la información. Para la aplicación sería lo más próximo a la base de datos. En este caso es simulada.

2.2.3. Prerrequisitos

- El usuario debe tener descargada la aplicación de la UOC en su dispositivo de Alexa
- El usuario debe tener el dispositivo de Alexa conectado a Internet
- El usuario tiene abierta la aplicación de la UOC (la petición actual se hará después de que el usuario haya dicho “Alexa, abre la UOC”)

2.2.4. Flujo de éxito

El usuario debe pedir a Alexa hacer login:

1) El usuario le dirá a Alexa la frase “Alexa, identifícame como {nombre_usuario} y contraseña {contraseña}”
2) Alexa se conectará al servicio rest y realizará el login con el usuario y contraseña
3) Alexa informará que el usuario se ha logado correctamente y le dará la bienvenida personalizada
4) Alexa dejara en su propia sesión el usuario logado.
5) Se acaba el caso de uso

2.2.5. Flujos alternativos de error

- Alexa no entiende la petición
  - Alexa no entiende la petición
  - Devuelve una frase indicando que no lo ha entendido y que vuelva a realizar la petición
  - Se vuelve a comenzar el caso de uso en el paso 1)
- Se produce un error de conexión
  - Se produce un error al intentar conectar con los servicios rest
  - Alexa devuelve una frase indicando que se ha producido un error en la petición
  - Se acaba el caso de uso
- Se produce un error de usuario o contraseña incorrectos
  - Se produce un error al intentar hacer login con los servicios rest
  - Alexa devuelve una frase indicando que se ha producido un error y que el usuario o la contraseña son incorrectos
  - Se acaba el caso de uso

2.2.6. Información adicional

La frase que deberá devolver Alexa en este servicio será de bienvenida. Se simula la llamada y se devuelve un texto estático del tipo:

“Bienvenido a la UOC, {nombre_usuario}”

Donde {nombre_usuario} es el nombre del usuario.

2.2.7. Utterances (frases)

Las posibles frases que podrá decir el usuario para activar esta funcionalidad son:

- Identifícame como {nombre_usuario}
- Soy {nombre_usuario} con contraseña {contraseña}
- Conectarme como {nombre_usuario} y contraseña {contraseña}

2.3. Caso de uso de “Pedir notas”

2.3.1. Objetivo

Alexa devolverá las notas conocidas hasta el momento del semestre actual.

2.3.2. Diagrama de flujo

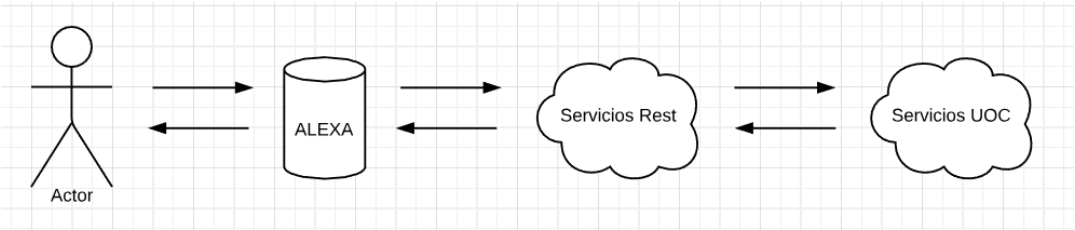


Ilustración 12 - Caso de uso de Pedir Notas [4]

- Actor: representa al usuario que se comunicará con la IA ALEXA.
- ALEXA: Alexa es la parte de la IA donde se creará todo lo relacionado con la interacción con el usuario como las frases que entenderá del usuario y a qué servicio llamar.
- Servicios Rest: son la parte de backend encargada de conectar con la UOC para obtener la información que pide el usuario. Estos son los servicios que se van a desarrollar en este trabajo.
- Servicios UOC: representa la parte backend de donde se nutre la aplicación para obtener la información. Para la aplicación sería lo más próximo a la base de datos. En este caso es simulada.

2.3.3. Prerrequisitos

- El usuario debe tener descargada la aplicación de la UOC en su dispositivo de Alexa
- El usuario debe tener el dispositivo de Alexa conectado a Internet
- El usuario tiene abierta la aplicación de la UOC (la petición actual se hará después de que el usuario haya dicho “Alexa, abre la UOC”)
- El usuario debe estar logado (Caso de uso de Login)

2.3.4. Flujo de éxito

El usuario debe pedir a Alexa que le diga las notas:

1) El usuario le dirá a Alexa la frase “Alexa, dime las notas”

2) Alexa se conectará al servicio rest y obtendrá la frase que deberá decir
3) Alexa dirá la frase devuelta por el servicio rest con la información de las notas
4) Se acaba el caso de uso

2.3.5. Flujos alternativos de error

- Alexa no entiende la petición
  - Alexa no entiende la petición
  - Devuelve una frase indicando que no lo ha entendido y que vuelva a realizar la petición
  - Se vuelve a comenzar el caso de uso en el paso 1)
- Se produce un error de conexión
  - Se produce un error al intentar conectar con los servicios rest
  - Alexa devuelve una frase indicando que se ha producido un error en la petición
  - Se acaba el caso de uso

2.3.6. Información adicional

La frase que deberá devolver Alexa en este servicio serán las notas del usuario. Se simula la llamada y se devuelve un texto estático del tipo:

"Has sacado una {nota} en la actividad {actividad} de la asignatura {asignatura}"

Donde {asignatura} es el nombre de la asignatura, por ejemplo "Trabajo Final de Grado"

{nota} es la nota sacada con los posibles valoras A, B, C+, C-, D, NP

{actividad} es la actividad en cuestión como por ejemplo PEC1 o PRA1

En caso de que no se pida saber las notas de una asignatura en concreto se dirán todas las asignaturas y actividades que lleve el usuario:

"En {asignatura} has sacado una {nota}, en la {actividad} y una {nota}, en la {actividad}. En {asignatura} has sacado una {nota}, en la {actividad}, y una {nota}, en la {actividad}."

2.3.7. Utterances (frases)

Las posibles frases que podrá decir el usuario para activar esta funcionalidad son:

- Qué he sacado en {asignatura}
- Que notas he sacado en {asignatura}
- Que notas he sacado
- Cuáles son mis notas

2.4. Caso de uso de “Calendario”

2.4.1. Objetivo

Alexa devolverá los eventos que importantes en los próximos siete días.

2.4.2. Diagrama de flujo

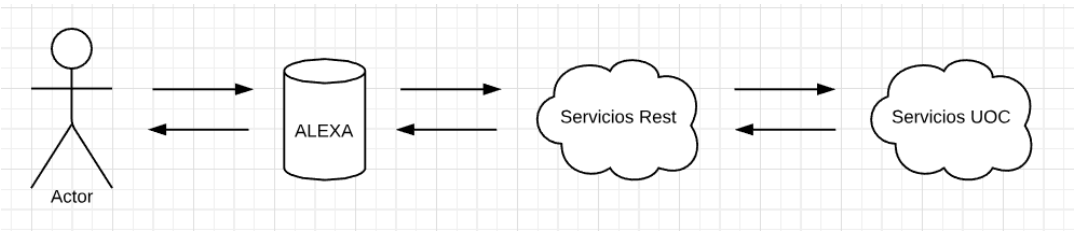


Ilustración 13 - Caso de uso de Pedir Notas [4]

- Actor: representa al usuario que se comunicará con la IA ALEXA.
- ALEXA: Alexa es la parte de la IA donde se creará todo lo relacionado con la interacción con el usuario como las frases que entenderá del usuario y a qué servicio llamar.
- Servicios Rest: son la parte de backend encargada de conectar con la UOC para obtener la información que pide el usuario. Estos son los servicios que se van a desarrollar en este trabajo.
- Servicios UOC: representa la parte backend de donde se nutre la aplicación para obtener la información. Para la aplicación sería lo más próximo a la base de datos. En este caso es simulada.

2.4.3. Prerrequisitos

- El usuario debe tener descargada la aplicación de la UOC en su dispositivo de Alexa
- El usuario debe tener el dispositivo de Alexa conectado a Internet

- El usuario tiene abierta la aplicación de la UOC (la petición actual se hará después de que el usuario haya dicho “Alexa, abre la UOC”)
- El usuario debe estar logado (Caso de uso de Login)

2.4.4. Flujo de éxito

El usuario debe pedir a Alexa que le diga que eventos tiene próximamente:

1) El usuario le dirá a Alexa la frase “Alexa, que eventos tengo”
2) Alexa preguntará si se desea saber los eventos de los próximos siete días
3) Si el usuario dice que sí, Alexa se conectará al servicio rest y obtendrá la frase que deberá decir
4) Alexa dirá la frase devuelta por el servicio rest con la información de los eventos
5) Se acaba el caso de uso

2.4.5. Flujos alternativos

En el punto 3) del flujo de éxito:

3.1) Si el usuario dice que no, Alexa no contestará con los eventos

4) Se acaba el caso de uso

2.4.6. Flujos alternativos de error

- Alexa no entiende la petición
  - Alexa no entiende la petición
  - Devuelve una frase indicando que no lo ha entendido y que vuelva a realizar la petición
  - Se vuelve a comenzar el caso de uso en el paso 1)
- Se produce un error de conexión
  - Se produce un error al intentar conectar con los servicios rest
  - Alexa devuelve una frase indicando que se ha producido un error en la petición
  - Se acaba el caso de uso

2.4.7. Información adicional

La frase que deberá devolver Alexa en este servicio serán los eventos que tenga el usuario en los próximos siete días. Se simula la llamada y se devuelve un texto estático del tipo:

"El miércoles 12 de junio tienes la entrega de la {actividad} de {asignatura}"

Donde {asignatura} es el nombre de la asignatura, por ejemplo "Trabajo Final de Grado"

{actividad} es la actividad en cuestión como por ejemplo PEC1 o PRA1

2.4.8. Utterances (frases)

Las posibles frases que podrá decir el usuario para activar esta funcionalidad son:

- Que eventos tengo
- Qué tengo para la próxima semana
- Dime los eventos

3. Creación del proyecto rest

Para crear un servicio rest lo primero que se va a hacer es instalar un plugin de Eclipse para la creación de proyectos llamado Spring Tool Suite (STS):

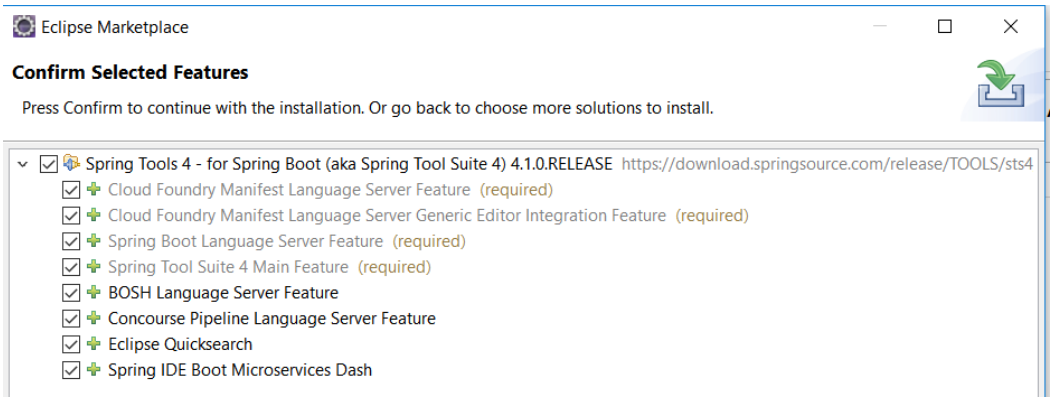


Ilustración 14 - Instalación plugin STS

Una vez instalado ya se puede crear el proyecto de Spring y Maven con este nuevo plugin. Lo que hará internamente es crear el proyecto igual que si se creará con la herramienta de Spring Initializr:

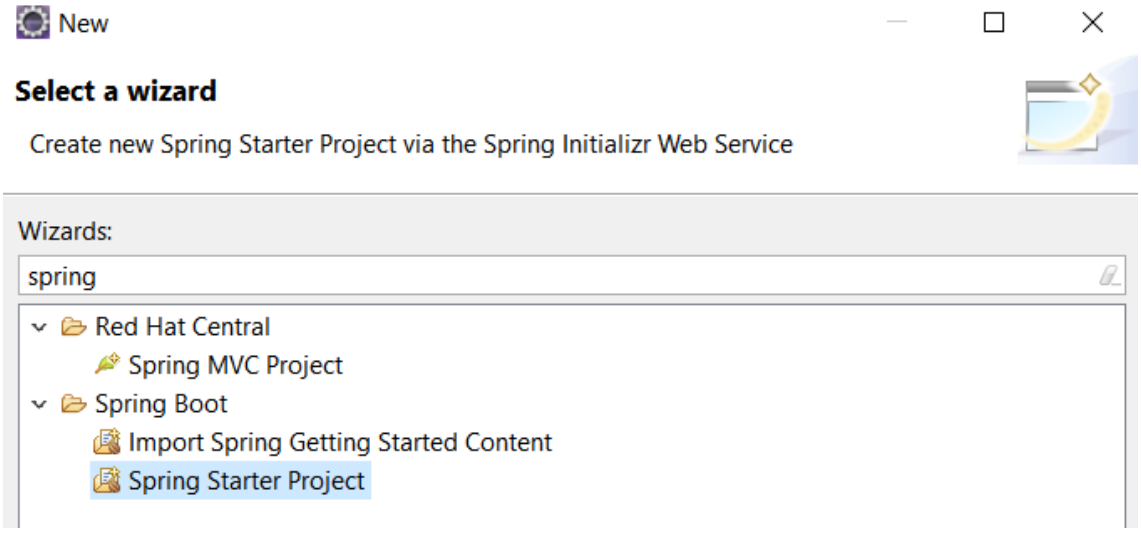


Ilustración 15 - Creación proyecto Spring Boot

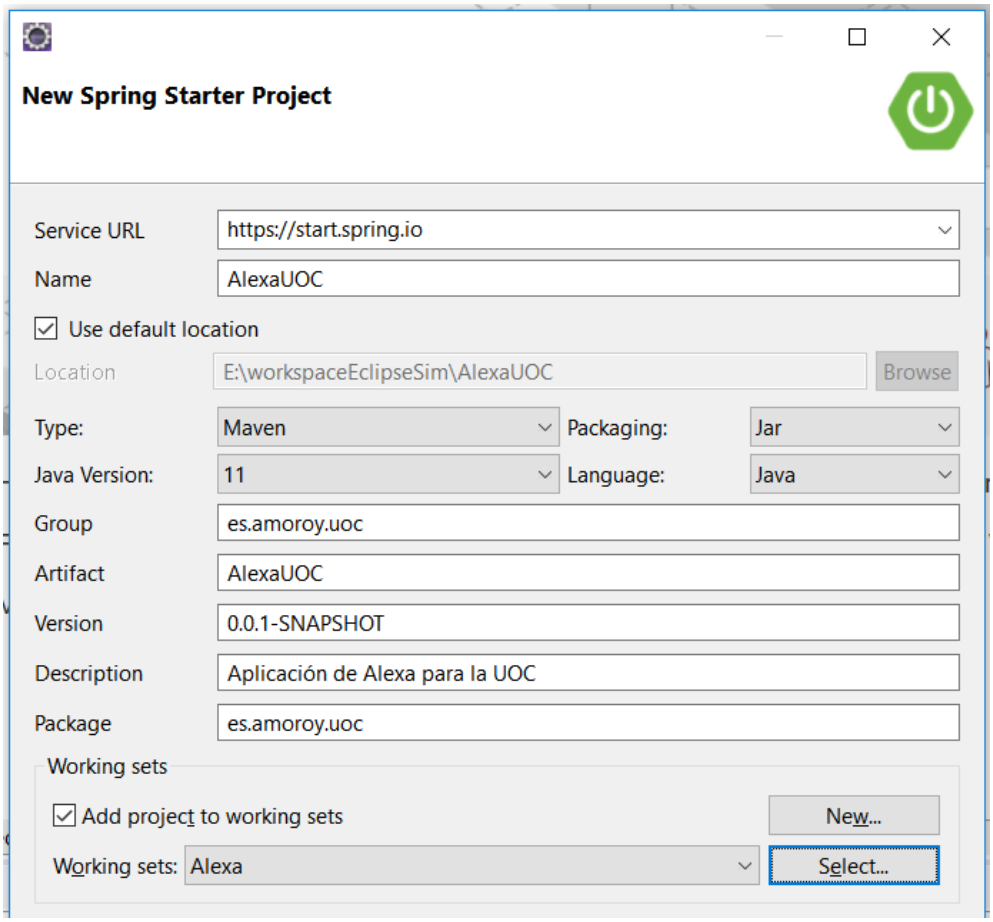


Ilustración 16 - Propiedades del proyecto rest

Así ya se dispondría de la estructura base del proyecto para poder desarrollar la aplicación. La estructura será de tipo maven donde se separan las clases del código fuente de los test.

3.1. Conectar el proyecto al Github

Una vez se ha creado el proyecto rest se debe conectar al repositorio de Github que se ha creado para este TFG. Para conseguirlo hay que ir al repositorio de Github y copiar la url del path del proyecto:

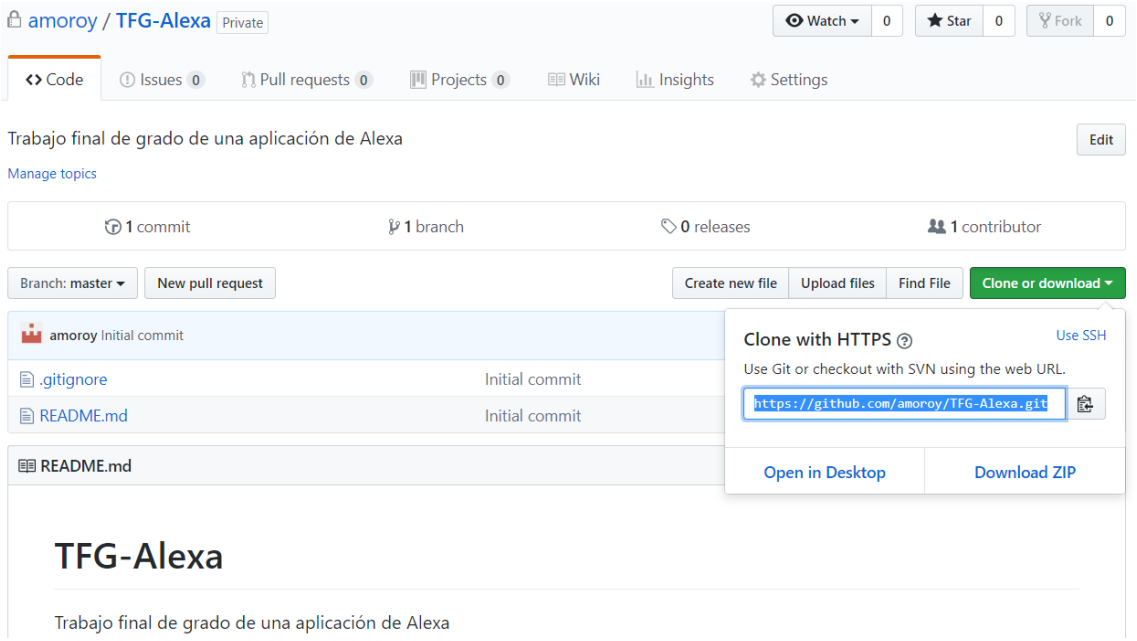


Ilustración 17 - Repositorio Github

El IDE Eclipse te permite crear un repositorio de manera casi automática con esa url. Basta con pegarla en la vista de “Git repositories”:

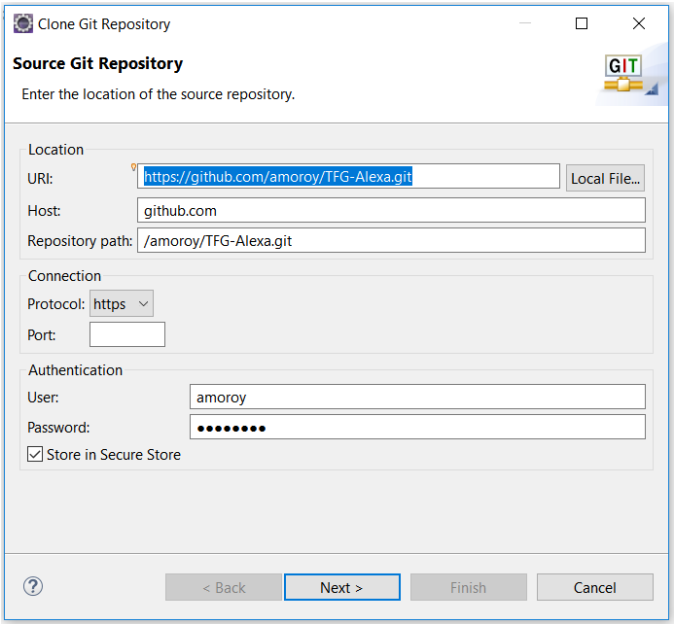


Ilustración 18 - Conexión Eclipse - Github

Una vez se tiene creado el repositorio de git solo se necesita unir nuestro proyecto rest a ese repositorio mediante Team → Share Project y aquí se elige el repositorio recién creado.

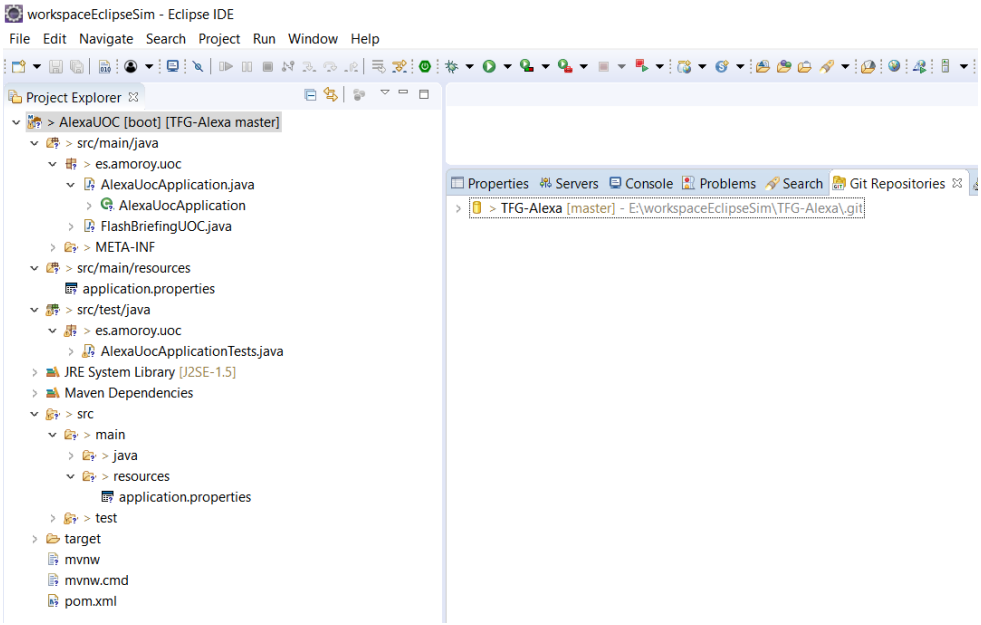


Ilustración 19 - Eclipse con proyecto de Github

El siguiente paso es ver los cambios y empezar con la subida de los ficheros:

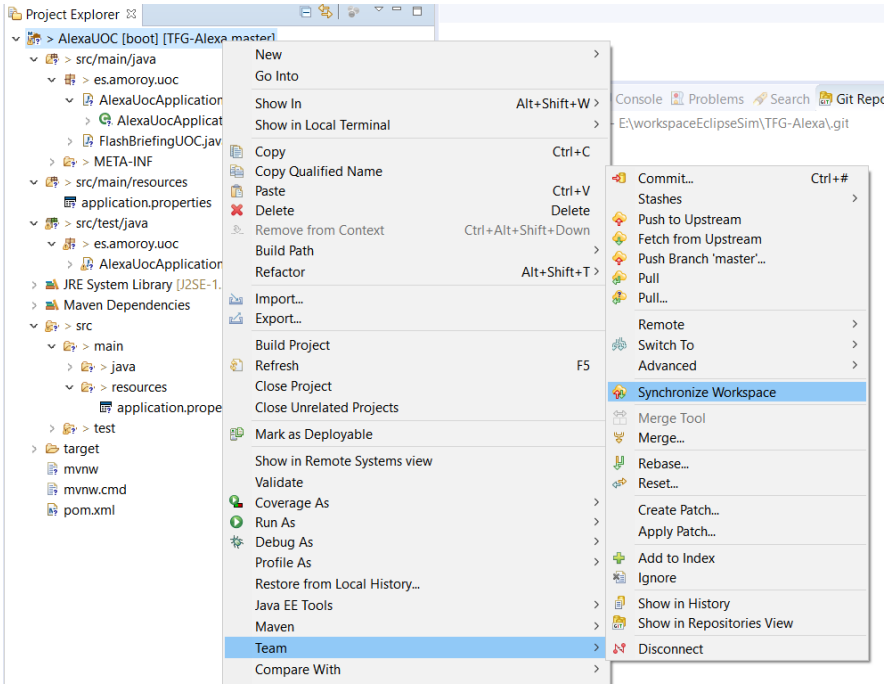


Ilustración 20 - Eclipse sincronizando proyecto

- Se puede hacer mediante el propio Eclipse con la opción Team → Synchronize Workspace.

Donde se puede ver todo lo que se tiene por subir y hacer las acciones necesarias para subir los archivos:

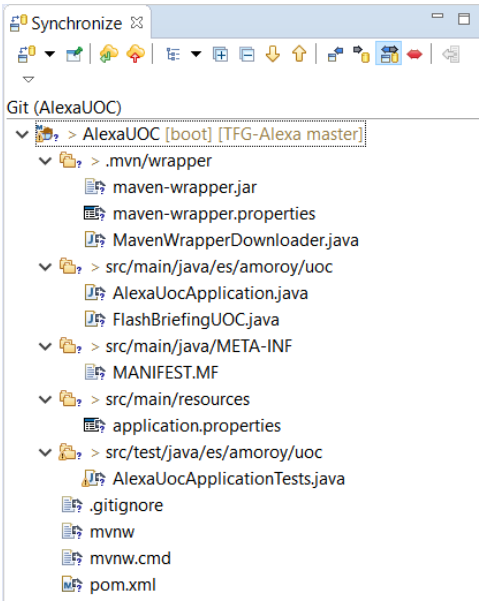


Ilustración 21 - Eclipse mostrando cambios

- O se puede hacer mediante la consola de Git (tanto la nativa de Git bash como la integrada en el propio Eclipse):

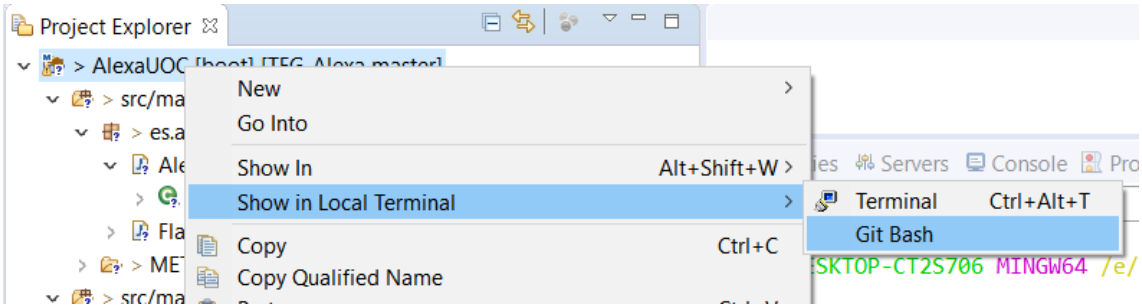


Ilustración 22 - Eclipse proyecto en Git Bash

Cuando se haga el primer pull se pedirá hacer login en GitHub:

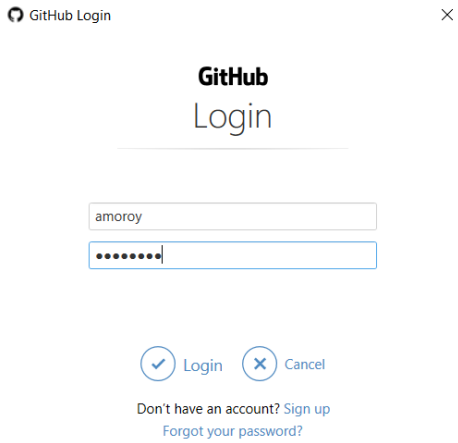


Ilustración 23 - Github login

Una vez hecho ya se puede seguir con el resto de comandos:

```
alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (master)
$ git pull
Already up to date.

alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ./

nothing added to commit but untracked files present (use "git add" to track)
```

Ilustración 24 - Eclipse comandos git

Entre medias, como buena práctica, se va a crear una nueva rama llamada “develop” para no subir los desarrollos directamente a “master”. Ya que la rama master es la que representa lo que iría a producción mientras que la rama develop es en la que se desarrolla y se prueba:

```
alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (master)
$ git checkout -b develop
Switched to a new branch 'develop'
A       AlexaUOC/.gitignore
A       AlexaUOC/.mvn/wrapper/MavenWrapperDownloader.java
A       AlexaUOC/.mvn/wrapper/maven-wrapper.jar
A       AlexaUOC/.mvn/wrapper/maven-wrapper.properties
A       AlexaUOC/mvnw
A       AlexaUOC/mvnw.cmd
A       AlexaUOC/pom.xml
A       AlexaUOC/src/main/java/META-INF/MANIFEST.MF
A       AlexaUOC/src/main/java/es/amoroy/uoc/AlexaUocApplication.java
A       AlexaUOC/src/main/java/es/amoroy/uoc/FlashBriefingUOC.java
A       AlexaUOC/src/main/resources/application.properties
A       AlexaUOC/src/test/java/es/amoroy/uoc/AlexaUocApplicationTests.java

alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (develop)
```

Ilustración 25 - Eclipse comando git 3

```
alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (develop)
$  git push --set-upstream origin develop
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:      https://github.com/amoroy/TFG-Alexa/pull/new/develop
remote:
To https://github.com/amoroy/TFG-Alexa.git
 * [new branch]      develop -> develop
Branch 'develop' set up to track remote branch 'develop' from 'origin'.
```

Ilustración 26 - Eclipse comandos git 4

Así el siguiente paso es realizar una “git add ./”. Este comando empaquetara todos los ficheros:

```
AlexaUOC (DESKTOP-CT2S706)
alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (master)
$ git add ./
warning: LF will be replaced by CRLF in AlexaUOC/.gitignore.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/.mvn/wrapper/MavenWrapperDownloader.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/.mvn/wrapper/maven-wrapper.properties.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/mvnw.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/mvnw.cmd.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/pom.xml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/src/main/java/es/amoroy/uoc/AlexaUocApplication.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/src/main/resources/application.properties.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in AlexaUOC/src/test/java/es/amoroy/uoc/AlexaUocApplicationTests.java.
The file will have its original line endings in your working directory

alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore
        new file:   .mvn/wrapper/MavenWrapperDownloader.java
        new file:   .mvn/wrapper/maven-wrapper.jar
        new file:   .mvn/wrapper/maven-wrapper.properties
        new file:   mvnw
        new file:   mvnw.cmd
        new file:   pom.xml
        new file:   src/main/java/META-INF/MANIFEST.MF
        new file:   src/main/java/es/amoroy/uoc/AlexaUocApplication.java
        new file:   src/main/java/es/amoroy/uoc/FlashBriefingUOC.java
        new file:   src/main/resources/application.properties
        new file:   src/test/java/es/amoroy/uoc/AlexaUocApplicationTests.java
```

Ilustración 27 - Eclipse comandos git 2

En verde ya salen todos los ficheros listos para subir mediante un commit. Por ejemplo se va a hacer el comando “git commit –m ‘Primera subida’”:

```
alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (develop)
$ git commit -m 'Primera subida'
[develop 936e42e] Primera subida
 12 files changed, 723 insertions(+)
 create mode 100644 AlexaUOC/.gitignore
 create mode 100644 AlexaUOC/.mvn/wrapper/MavenWrapperDownloader.java
 create mode 100644 AlexaUOC/.mvn/wrapper/maven-wrapper.jar
 create mode 100644 AlexaUOC/.mvn/wrapper/maven-wrapper.properties
 create mode 100644 AlexaUOC/mvnw
 create mode 100644 AlexaUOC/mvnw.cmd
 create mode 100644 AlexaUOC/pom.xml
 create mode 100644 AlexaUOC/src/main/java/META-INF/MANIFEST.MF
 create mode 100644 AlexaUOC/src/main/java/es/amoroy/uoc/AlexaUocApplication.java
 create mode 100644 AlexaUOC/src/main/java/es/amoroy/uoc/FlashBriefingUOC.java
 create mode 100644 AlexaUOC/src/main/resources/application.properties
 create mode 100644 AlexaUOC/src/test/java/es/amoroy/uoc/AlexaUocApplicationTests.java
```

Ilustración 28 - Eclipse comandos git 5

Con estos pasos se ha creado un “paquete” listo para subir al repositorio de GitHub. De momento todo está en un repositorio en la máquina local. Por lo que ahora hay que subirlo al repositorio mediante un comando push:

```
alvar@DESKTOP-CT2S706 MINGW64 /e/workspaceEclipseSim/TFG-Alexa/AlexaUOC (develop)
$ git push
Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 8 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (30/30), 50.32 KiB | 4.19 MiB/s, done.
Total 30 (delta 0), reused 0 (delta 0)
To https://github.com/amoroy/TFG-Alexa.git
   f1fccc8..936e42e  develop -> develop
```

Ilustración 29 - Eclipse comandos git 6

Y en el GitHub ya está guardado el código y se podrá visualizar:

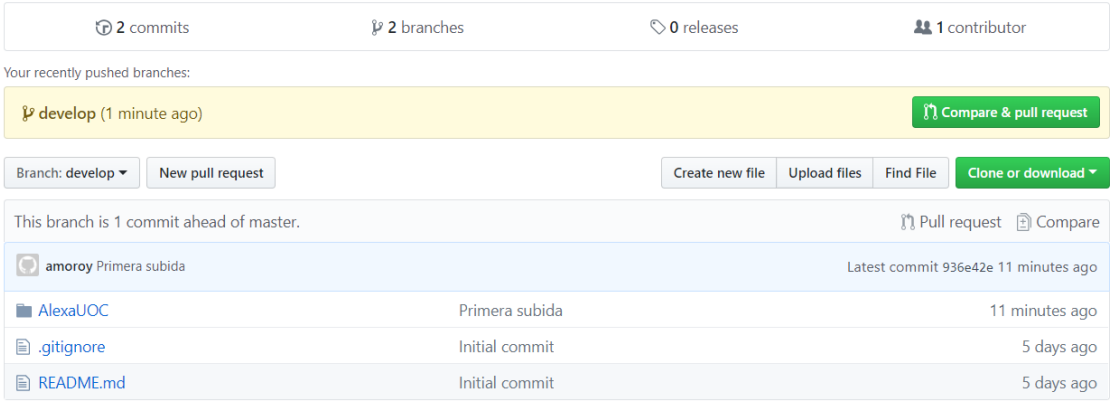


Ilustración 30 - Github proyecto

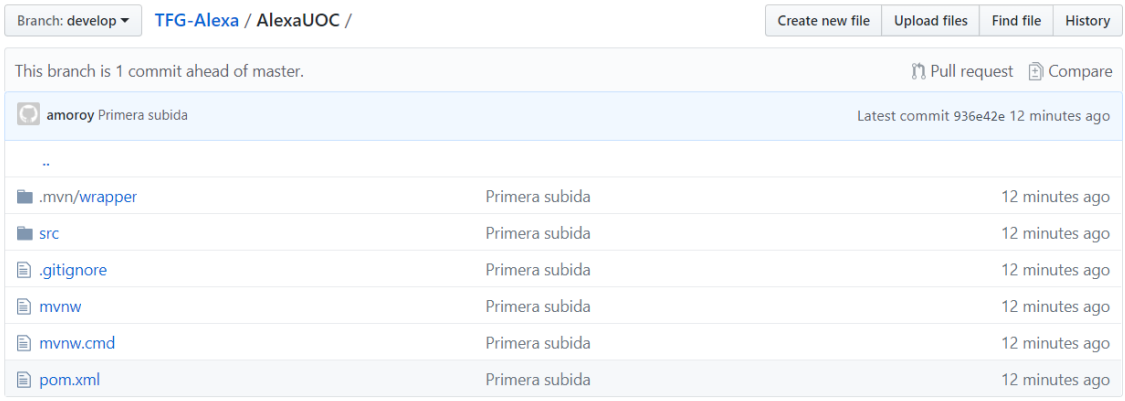


Ilustración 31 - Github proyecto 2

4. Creación de la primera Skill

El primer paso es ir a la consola de Amazon Alexa que es donde se ofrece todo lo relacionado con la creación de las Skills:

https://developer.amazon.com/alexa/console/ask

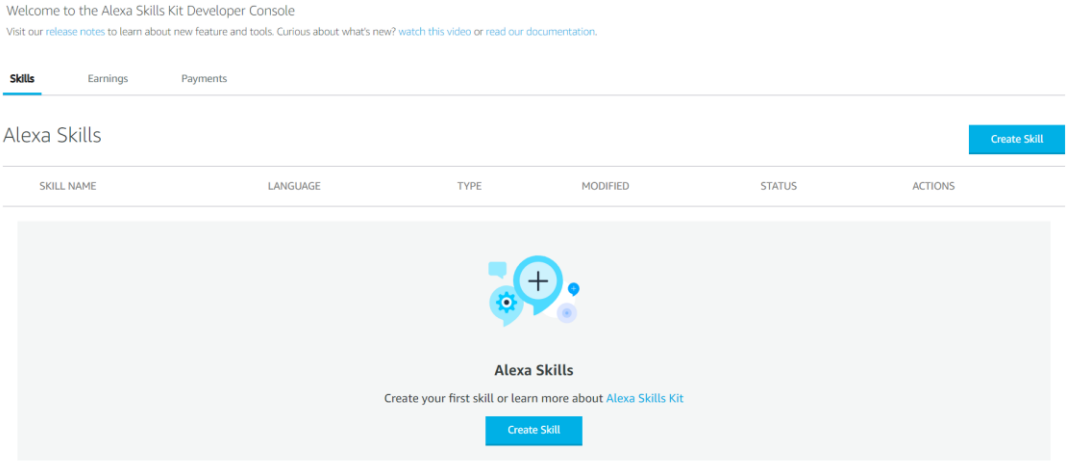


Ilustración 32 - Primera Skill

Así la primera pantalla dará a elegir entre el tipo de skill que se quiere crear. En este caso se debe seleccionar el Flash Briefing que es la primera skill que hay que hacer para una aplicación.

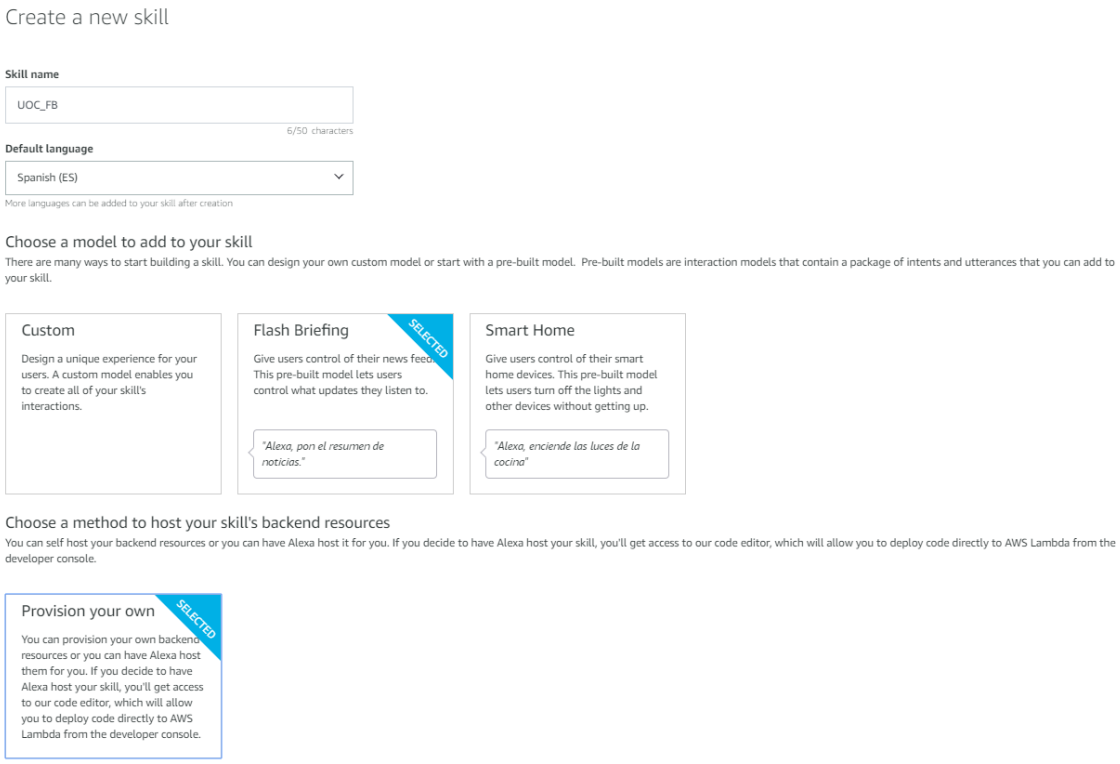


Ilustración 33 - Selección tipo de skill

Y automáticamente ya se tiene la primera skill lista para configurar.

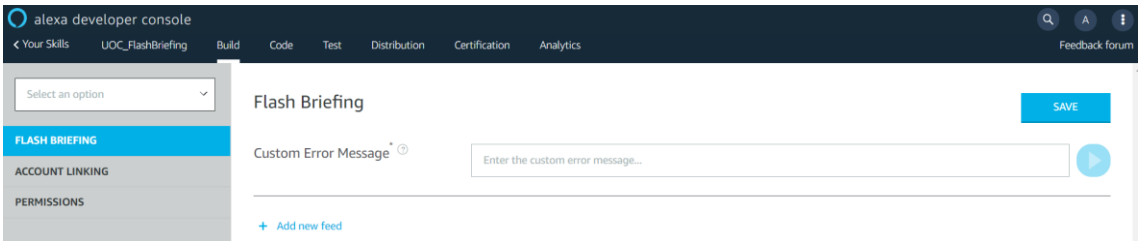


Ilustración 34 - Dashboard Flash Briefing

Aquí se añadirá un nuevo “feed” y se configurará el punto de acceso al servicio rest (apartado “Feed”) así como una frase resumen y tipo de contenido.

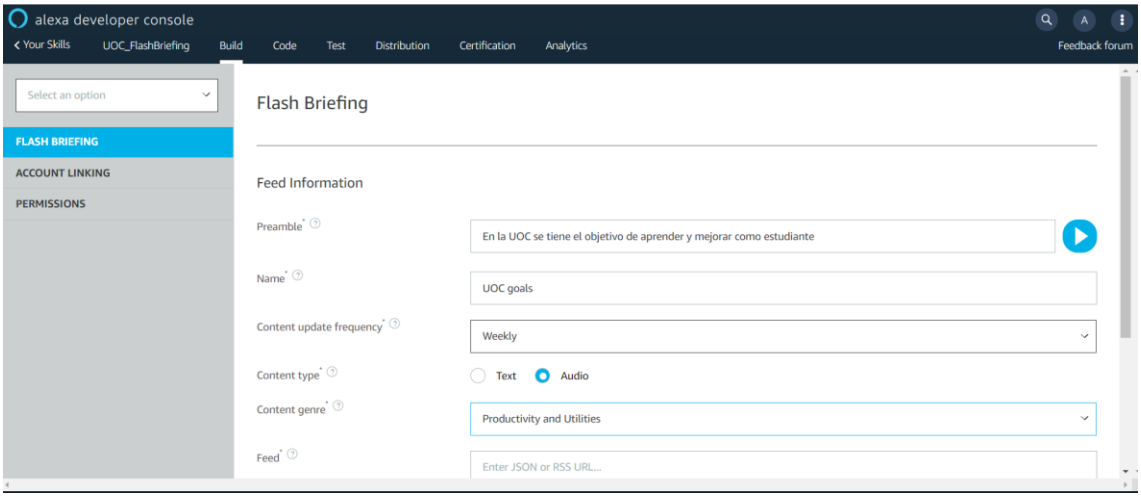


Ilustración 35 - Dashboard Flash Briefing 2

Al poner la url donde se expone el servicio rest y guardarla, pueden surgir errores que Alexa marcará. Por ejemplo que el servicio no está disponible y da un HttpStatus de la familia 4xx:

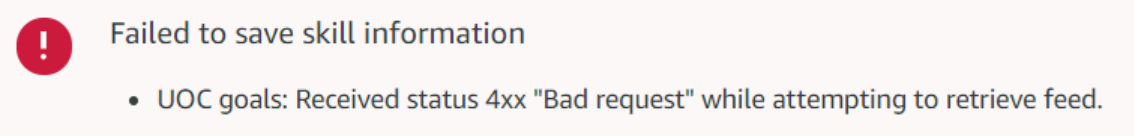


Ilustración 36 - Mensaje de error

Este error se produce porque el servicio rest no está bien expuesto y no se puede llegar a obtener la respuesta. Hay que verificar que el servicio está bien levantado.

Otro error es que el servicio no devuelva una respuesta en el formato adecuado, que en este caso Alexa exige que sea o JSON o RSS. En este proyecto todas las respuestas serán JSON.

También se puede dar que la fecha de creación del mensaje es de hace más de una semana:

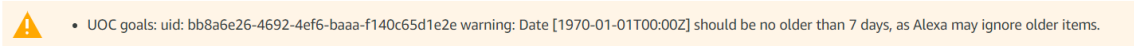


Ilustración 37 - Mensaje de advertencia

Esto se debe a que la respuesta del servicio tiene que llevar la fecha de generación de las respuestas y en caso de que esa fecha sea anterior a 7 días, Alexa ignorará este servicio.

5. Creación del servicio rest del Flash Briefing

Para la creación de este primer servicio rest primero se crea la estructura del proyecto explicada en el apartado [*Creación del proyecto rest*] y a continuación, una vez se tiene un proyecto rest se crea el

primer controlador para ver que funciona y despliega como se explica en el anexo [***¡Error! No se encuentra el origen de la referencia.***].

El siguiente paso es crear un controlador que devuelva lo que se quiere que Alexa responda al invocar las noticias del día. En este caso sería una noticia informativa acerca de la UOC.

Así, siguiendo la metodología TDD (Test Driven Development), se comenzará con un test que espere la frase en cuestión y que falle dicho al no devolver dicha frase. Se refactorizará el controlador para hacer que funcione el test y se volverá a ejecutar el test:

```
10 /**
11  * Clase de test para probar el controlador de la aplicacion
12  *
13  * @author amoroy
14  *
15  */
16 public class FlashBriefingUOCControllerTest {
17
18     private FlashBriefingUOCController controller;
19
20     @Before
21     public void setup() {
22         controller = new FlashBriefingUOCController();
23     }
24
25     @Test
26     public void flashBriefingTest() {
27         String speech = controller.flashBriefing();
28         Assert.assertEquals(speech, "La Universidad Oberta de Cataluña cuenta con 70.274 estudiantes, 71.598 "
29                 + "graduados, 93% de los graduados tiene empleo, 93% volvería a escoger la UOC, 4.600 personal docente "
30                 + "(propio y colaborador), 1ª universidad en línea del mundo");
31     }
32 }
33
```

```
Properties  Servers  Console  Problems  Search  Git Repositories  Terminal  JUnit  History
Finished after 0,045 seconds
Runs: 1/1   Errors: 0   Failures: 1
es.amoroy.uoc.web.FlashBriefingUOCControllerTest [R
  flashBriefingTest (0,001 s)
Failure Trace
org.junit.ComparisonFailure: expected:<[Bienvenido a la UOC]> but was:<[La Universidad Oberta de Cataluña cuenta con 70.274 estudiantes, 71.598
laborador), 1ª universidad en línea del mundo]>
at es.amoroy.uoc.web.FlashBriefingUOCControllerTest.flashBriefingTest(FlashBriefingUOCControllerTest.java:28)
```

Ilustración 38 - Primera prueba con TDD

Se refactoriza y se sigue implementando y mejorando el código:

```
@RestController
public class FlashBriefingUOCController {

    @RequestMapping(value = "/flash")
    public String flashBriefing() {
        return "La Universidad Oberta de Cataluña cuenta con 70.274 estudiantes, 71.598 grac
    }
}
```

Ilustración 39 - Primer controlador

Pero se puede ver que devolver un literal puesto “a pelo” no es lo más conveniente, así que se refactoriza para sacar dicho literal a una clase especifica donde definir todos los literales de la aplicación.

Más tarde se verá que al final se utiliza un fichero yaml para definir todos los literales de la aplicación. También se cambia el contenido de la frase para que sea una noticia y no unos datos fijos de la universidad.

```java
/**
 * Clase que contiene todos los literales de la aplicación
 *
 * @author amoroy
 *
 */
@Getter
@Setter
@Component
@ConfigurationProperties("literals")
public class Literals {

    private String flashBriefingUoc = "Hay que formar profesionales y ciudadanos

}
```

Ilustración 40 - Clase de literales

Y el controlador quedará:

```java
17 @Setter
18 @RestController
19 public class FlashBriefingUOCController {
20
21     @Autowired
22     private Literals literales;
23
24     @RequestMapping(value = "/flash")
25     public String flashBriefing() {
26         return literales.getFlashBriefingUoc();
27     }
28
29 }
```

Ilustración 41 - Controlador refactorizado

También, para que Alexa acepte la respuesta del servicio, se ha tenido que refactorizar el controlador para que el servicio devuelva un objeto respuesta adecuado para Alexa:

```java
/**
 * Controlador de entrada para las Skills de Alexa
 * @author amoroy
 *
 */
@Setter
@RestController
@RequestMapping(value = "/flash")
public class FlashBriefingUOCController {

    @Autowired
    private Literals literales;

    @GetMapping(produces = "application/json")
    public ResponseEntity<RespuestaBriefing> flashBriefing() {
        RespuestaBriefing respuestaBriefing = new RespuestaBriefing();
        respuestaBriefing.setMainText(literales.getFlashBriefingUoc());
        respuestaBriefing.setTitleText("Noticias de la UOC");
        respuestaBriefing.setUid(UUID.randomUUID().toString());
        String time = ZonedDateTime.now(ZoneId.of("UTC")).format(DateTimeFormatter.ISO_INSTANT);
        respuestaBriefing.setUpdateDate(time);
        return new ResponseEntity<>(respuestaBriefing, HttpStatus.OK);
    }

}
```

Ilustración 42 - Controlador adecuado para Alexa

Y una vez que se despliega todo de nuevo se puede ver como se devuelve el objeto que necesita Alexa para interpretar la respuesta:

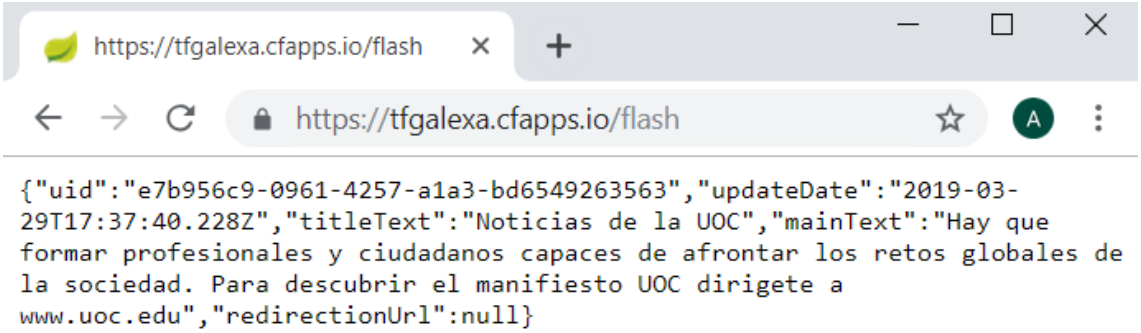


Ilustración 43 - Respuesta controlador

6. Creación de la Custom Skill

La Custom Skill será la aplicación de Alexa que contenga todo lo relacionado con la interacción de Alexa y el usuario. Como se ha comentado anteriormente, esta Skill tendrá todos los Intents (funcionalidades) para que el usuario interaccione.

El primer paso es la creación desde la consola de Alexa de dicha Skill, seleccionando la opción de aprovisionar a la Skill con un servicio rest propio:

https://developer.amazon.com/alexa/console/ask

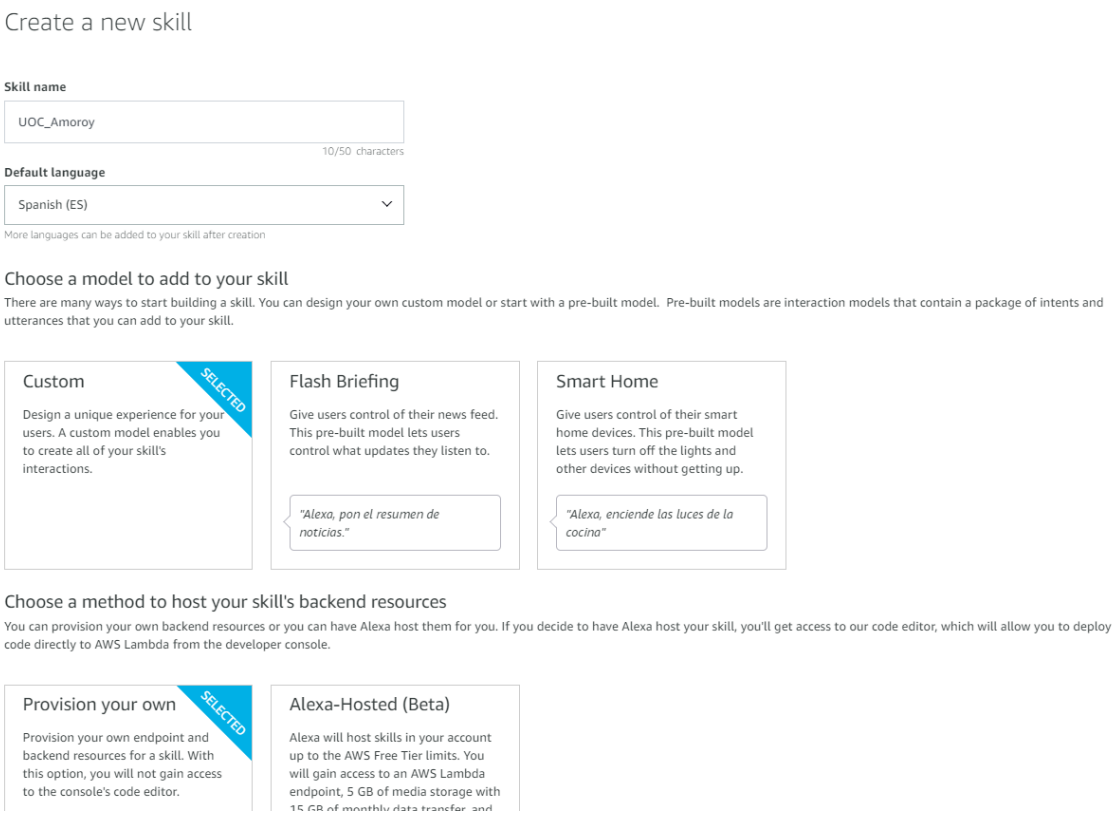


Ilustración 44 - Consola Alexa - Tipos de Skill

Y automáticamente ya se tiene la app creada en Alexa:

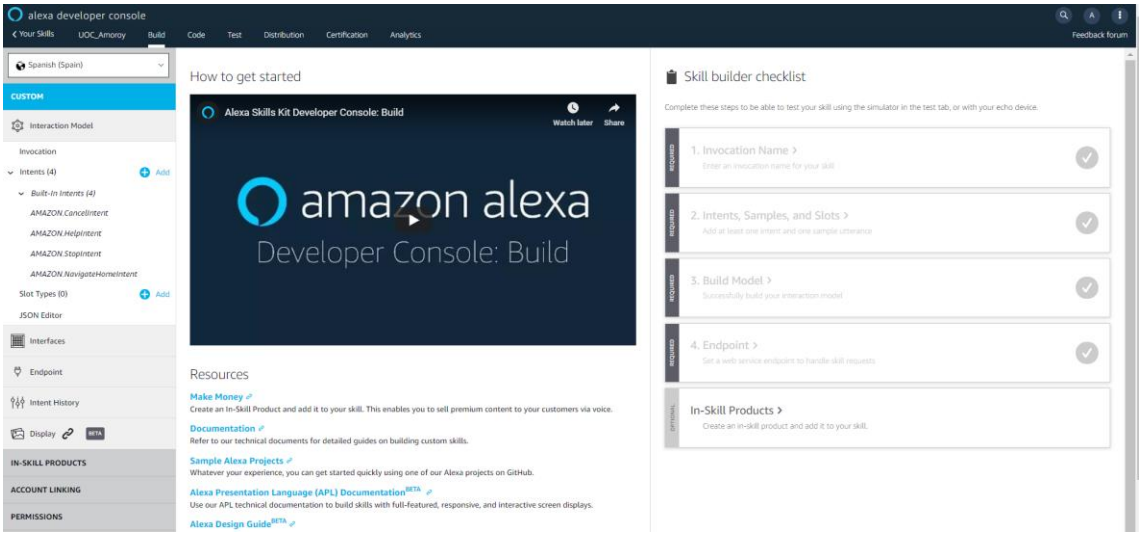


Ilustración 45 - Consola Alexa - Configuracion Custom Skill

6.1. Invocation name

Una de las partes más importantes es la definición del “invocation name” o nombre o palabras con las que se va a abrir la aplicación. Es muy

importante esta parte ya que Amazon no verifica que este nombre o palabras sean únicos entre todos los definidos en la plataforma.

¿Y qué consecuencias puede tener? Pues que si dos aplicaciones definen este “invocation name” de la misma manera, por ejemplo “el tiempo”, Amazon no asegura que vaya a abrirse tu aplicación y no la otra que haya definido la otra app.

Esto hace que sea muy importante definir un nombre o frase única para la aplicación. En este caso se va a definir el nombre de “la uoc”, que es lo suficientemente único como para que nadie más lo utilice.

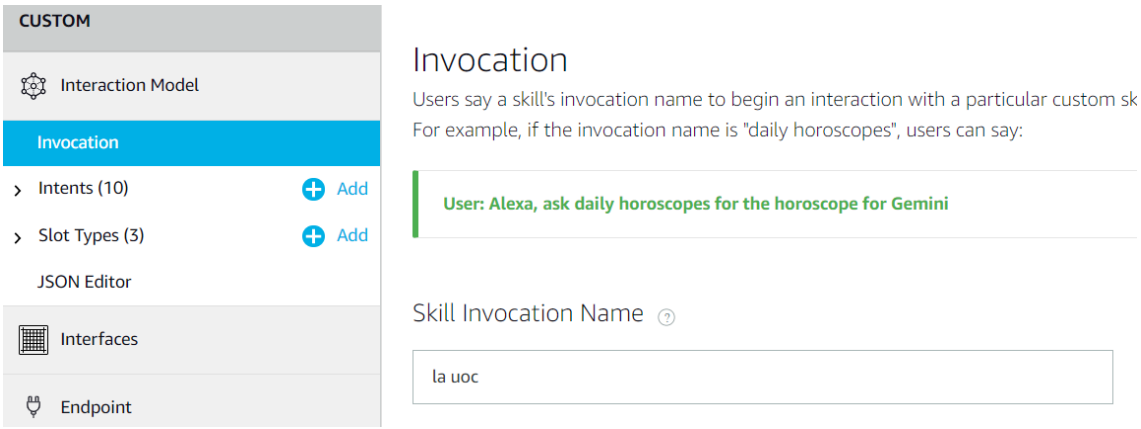


Ilustración 46 - Consola Alexa - Invocation Name

6.2. Intents (funcionalidades) en Alexa

En el menú se puede observar que ya hay algunos Intents por defecto:

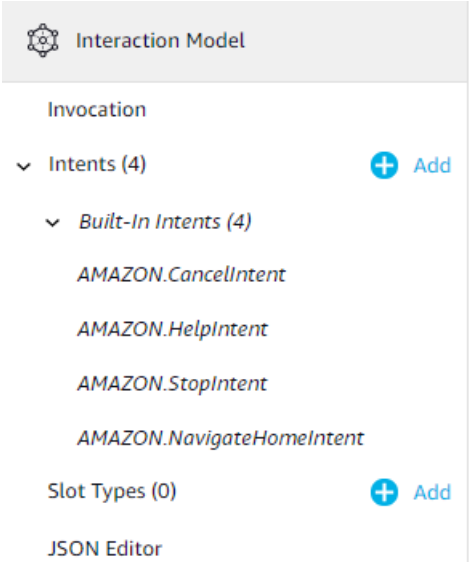


Ilustración 47 - Consola Alexa - Intents

Estos Intents son funcionalidades básicas que vienen pre-configurados con acciones típicas del usuario. Por ejemplo que el usuario necesite ayuda o desee cancelar la interacción actual.

Estas funcionalidades están preparadas para saltar cuando ocurren comportamientos genéricos como puede ser “Alexa, necesito ayuda” o “Alexa, para”.

Pero lo que Alexa conteste debe configurarse en los backend, ya que todo Intent debe estar desarrollado en un backend para que devuelva lo que la aplicación desee.

En este caso, el primer Intent que se desea crear es el del Login, que permita a un usuario identificarse en la UOC:

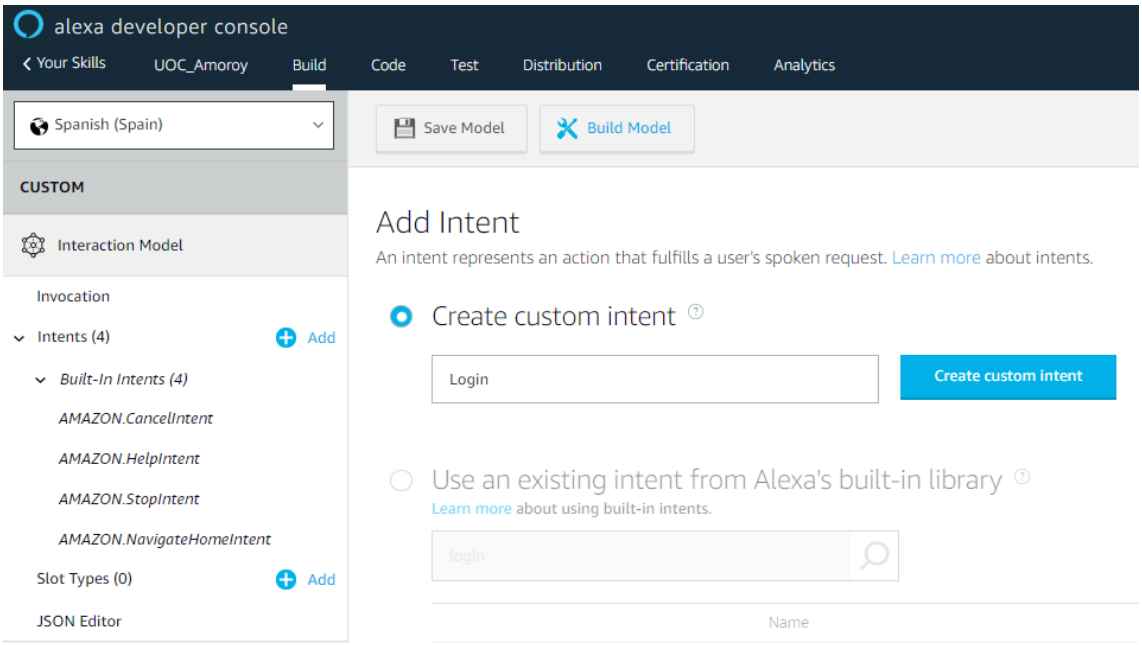


Ilustración 48 - Consola Alexa - Add Intent

Inmediatamente se crea dicho Intent y la primera pantalla que aparece es la de introducir los utterances (frases) que entienda esta funcionalidad:

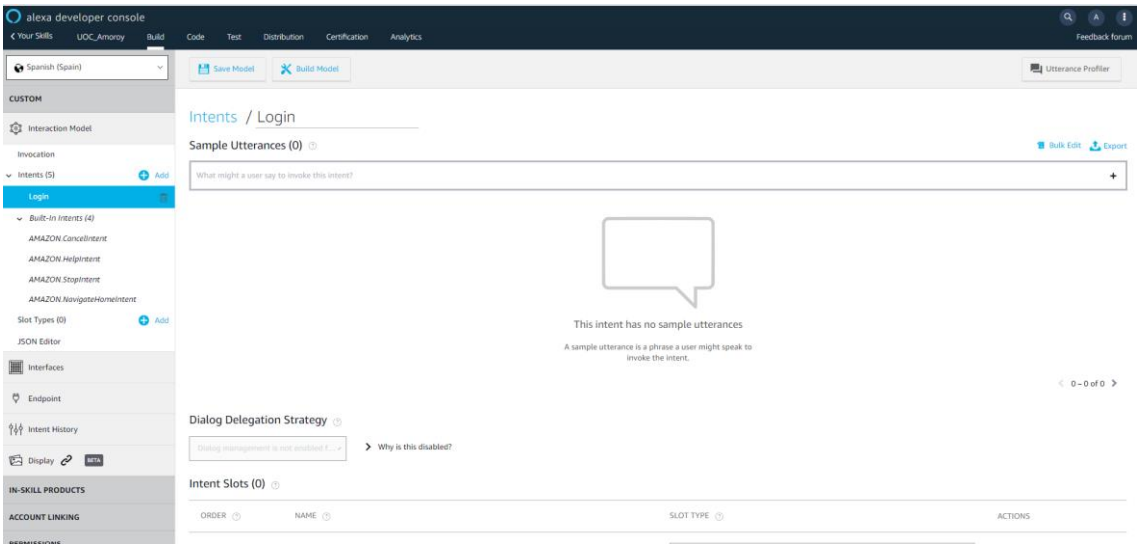


Ilustración 49 - Consola Alexa - Utterances

Amazon Alexa permite servicios de autenticaciones por varias vías. Una de ellas es utilizando al mismo usuario configurado en Alexa. Pero también permite utilizar autenticaciones mediante un sistema que requiere la autenticación en el sistema final una sola vez y la utilización de token y refresh token para mantener dicha conexión.

Si se quiere conocer más sobre la parte de autenticaciones se puede ir a la documentación de Amazon Alexa sobre las Account Linking:

https://developer.amazon.com/es/docs/account-linking/understand-account-linking.html

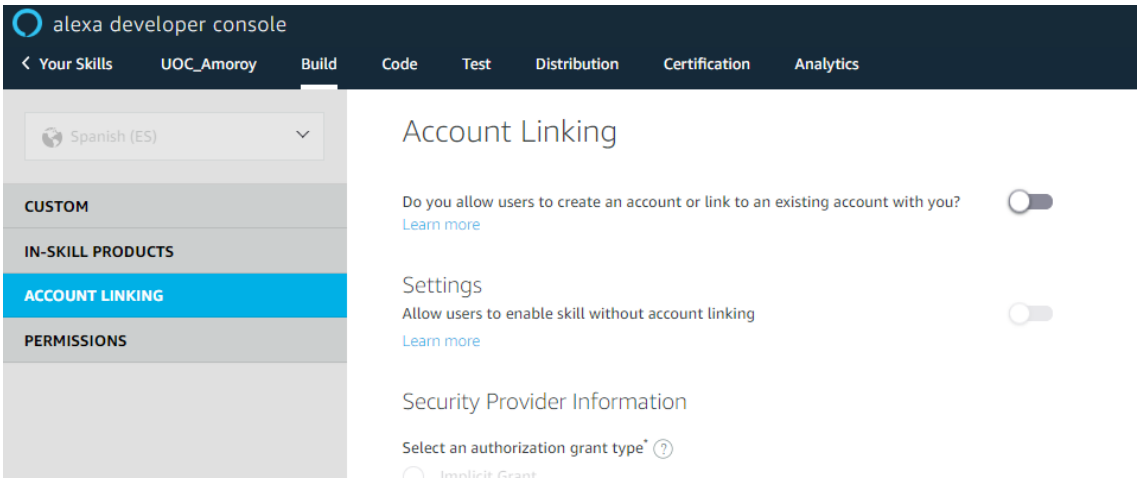


Ilustración 50 - Consola Alexa - Account Linking

Pero esta parte es demasiado compleja y delicada como para poder abordarla en una pequeña aplicación como esta. Por lo que se ha decidido simular la identificación de unas pocas personas en vez de cualquiera –se explicará a continuación con las variables–.

Así, las primeras frases (utterances) que se le dirá que entienda esta funcionalidad (Intent) serán las que aparecen en el caso de uso. Aunque

las frases podrían sufrir alguna modificación por la estandarización de caracteres, ya que no se pueden utilizar caracteres fuera del ISO del lenguaje inglés (como son las eñes):

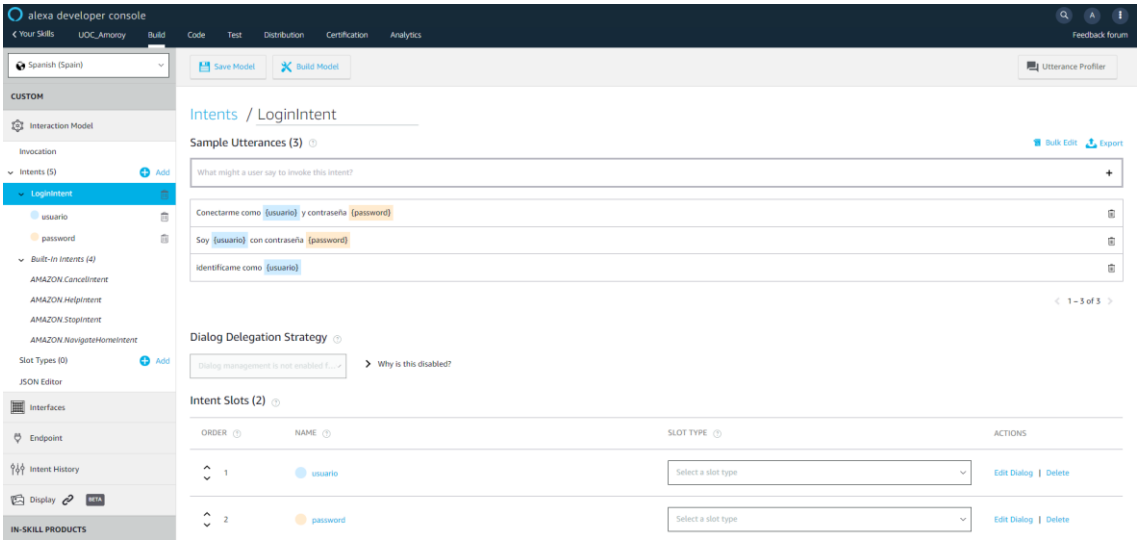


Ilustración 51 - Consola Alexa - LoginIntent

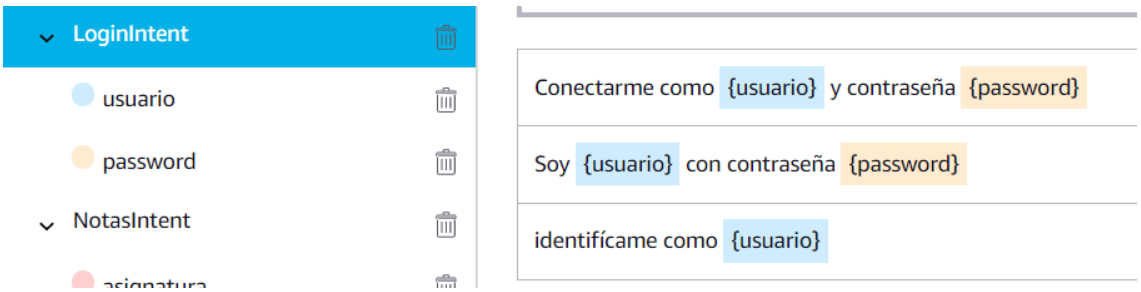


Ilustración 52 - Consola Alexa – LoginIntent - utterances

6.3. Slots (variables) y slot types

Al crear las frases Alexa identifica la necesidad de crear las variables (slots) que se han puesto entre llaves. En este caso {usuario} y {password}.

Estos slots permiten que en medio de la frase se introduzcan diferentes valores para la misma frase, evitando que haya que duplicar dicha frase por cada valor de esa variable.

Estas variables, a su vez, necesitan ser de un tipo (Slot Types) que será el que le diga a Alexa los posibles valores que puede tomar dicha variable y cuál será el valor que se envíe al servicio de backend.

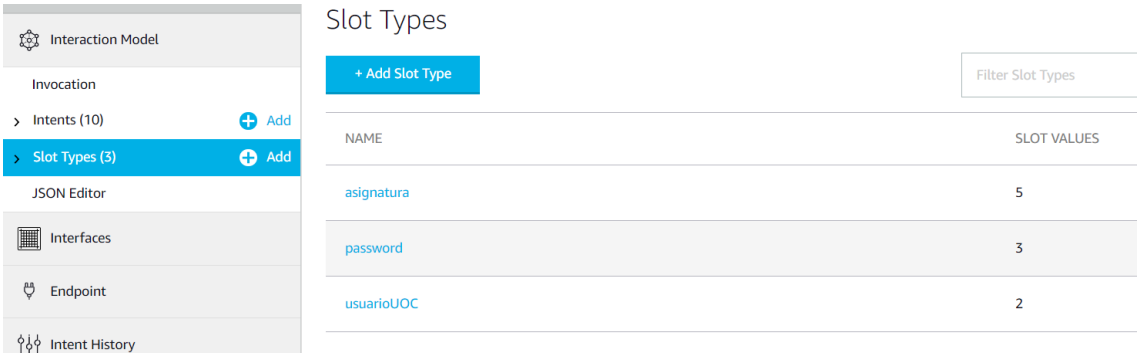


Ilustración 53 - Consola Alexa - Slots

Lo que quiere decir esto es que se pueden crear variables con todos los posibles valores que puede tener y además, se pueden utilizar sinónimos para esos valores.

Para este Intent se han creado dos slot type. Uno para el password y otro para el usuarioUOC. Y de este último se ha permitido los valores grobles, gregorio, amoroy, alvaro.

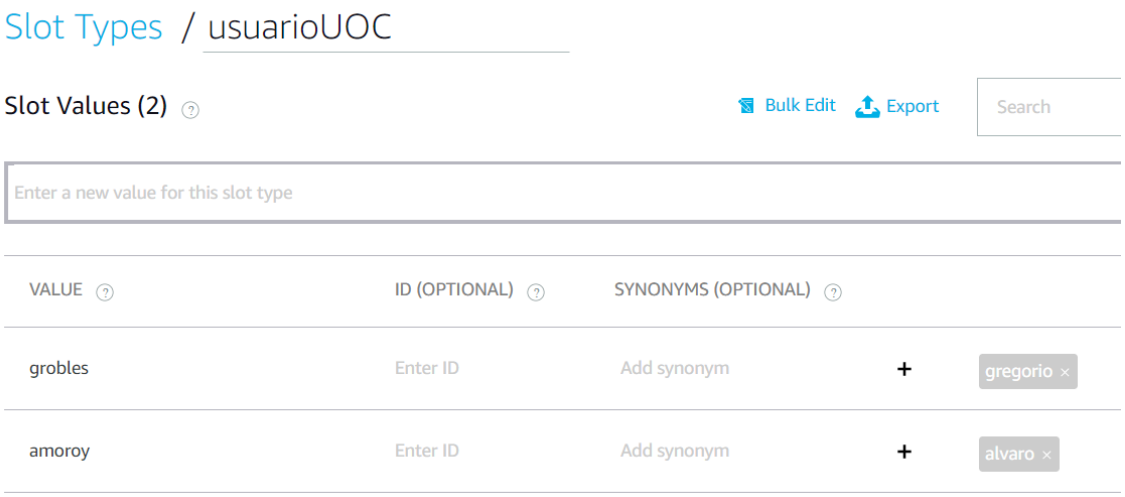


Ilustración 54 - Consola Alexa - Slot Types

Y de estos valores, se puede ver que gregorio es un sinónimo de grobles y alvaro es un sinónimo de amoroy. Pero si se utiliza alvaro en la frase, al servicio le llegará amoroy como valor de la variable.

Nota: este Intent se hace para simular el login, pero en una aplicación real iría configurado con las “Account Linking” y no como Intent.

6.4. Ejemplo

Se va a ver un ejemplo para entender mejor la potencia de las frases con variables y sinónimos:

Si se tiene la frase “Alexa, dame la notas de la asignatura {asignatura}”, la variable {asignatura} tendrá que ser de un tipo que contenga todas las posibilidades.

Así, la variable {asignatura} podría ser del tipo “asignaturaUOC” y sus posibles valores serán Álgebra, Análisis matemático, Estadística, Fundamentos de computadores,….

Ahora, el usuario podrá decir:

“Alexa, dame las notas de la asignatura Análisis matemático”

Pero además, Alexa permite configurar esos valores para que se puedan añadir sinónimos de los mismos y al servicio rest se le envíe el mismo valor independientemente del sinónimo utilizado.

¿Y qué ventaja tiene esto? Que si se quiere tener en cuenta que Análisis matemático puede decirse de varias formas, como puede ser solo Análisis, o AM, al servicio rest solo le llegará un valor y no tendrá que estar comprobando los posibles valores que lleguen.

Ahora el usuario podrá pedir las notas de esa asignatura de varias maneras y sin que el código de backend se vea afectado. Como buena práctica, el valor que debería llegarle al servicio rest es el mismo que requiere la UOC para identificar esa asignatura.

6.5. Intents y los servicios rest

Ahora que se ha comentado como se crean las frases y las variables se puede explicar más a fondo cómo Alexa sabe que se le está diciendo y que tiene que contestar.

Antes se han explicado las utterances (frases) que se han configurado para este Intent. Esas frases serán las que le digan a Alexa que es este Intent el que tiene que atender la petición.

Luego, cada funcionalidad (Intent) esta inmediatamente asociada con una clase en el servicio rest (que se explicará en el siguiente apartado). Esta clase sabe que variables le pueden llegar y que debe devolver en base a esas variables.

Por lo tanto hay una conexión directa entre los Intents que se crean en la parte de Alexa como los Intents que se crean como servicios de backend.

Una vez se ha conseguido tener todo bien configurado en la consola de Alexa saldrá el siguiente mensaje:

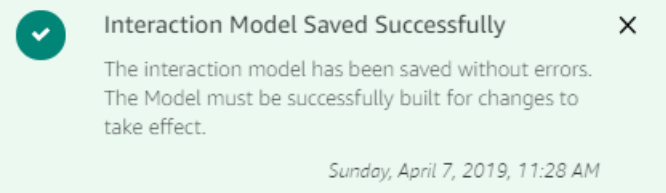


Ilustración 55 - Consola Alexa - Success Message

6.6. Acceso al servicio rest desde Alexa

Cuando se tiene un servicio operativo y se despliega en la nube (explicado en el anexo **¡Error! No se encuentra el origen de la referencia.**), hay que añadir la url donde está expuesta la aplicación con los servicios rest.

En el propio servicio rest se ha configurado para que se exponga el recurso "/alexa". Por lo tanto se pondrá la url que ofrece el servidor cloud Pivotal y se añadirá ese recurso "/alexa".

Así, en la propia consola de Alexa, en el apartado de "Endpoint" se aplicará dicha configuración:

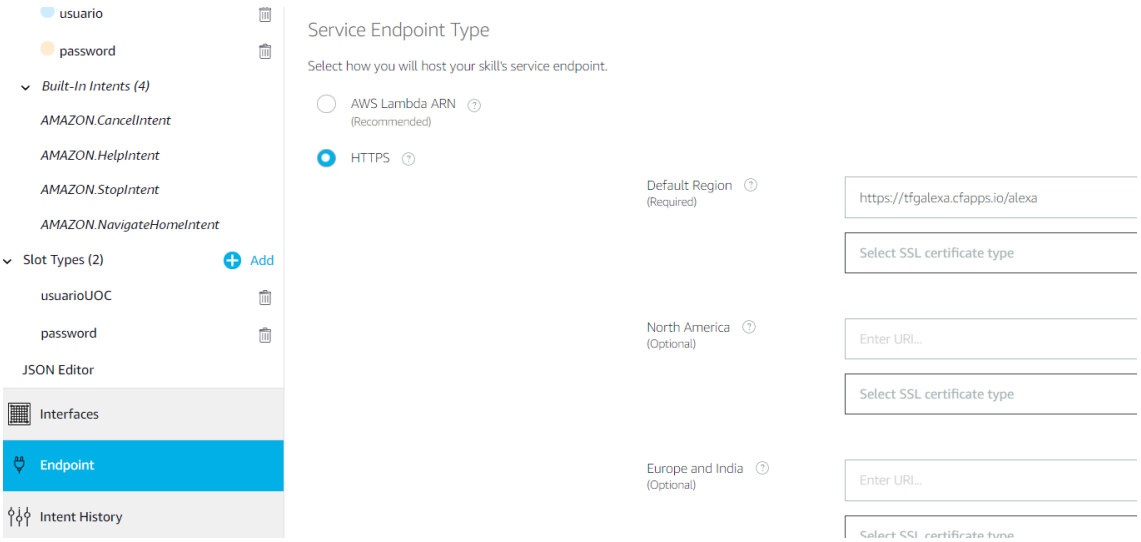
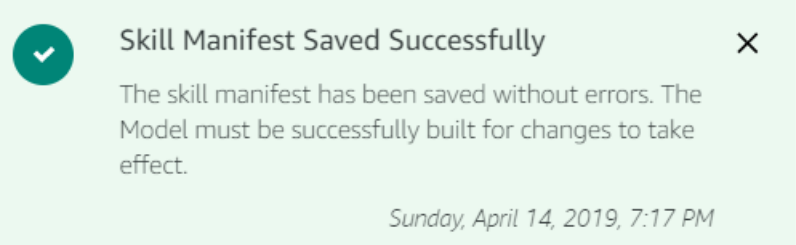


Ilustración 56 - Consola Alexa - Endpoint



Y ya se puede ir a la parte de test de la consola de Alexa (explicado en el apartado Test desde la consola de Alexa) y probar la aplicación:

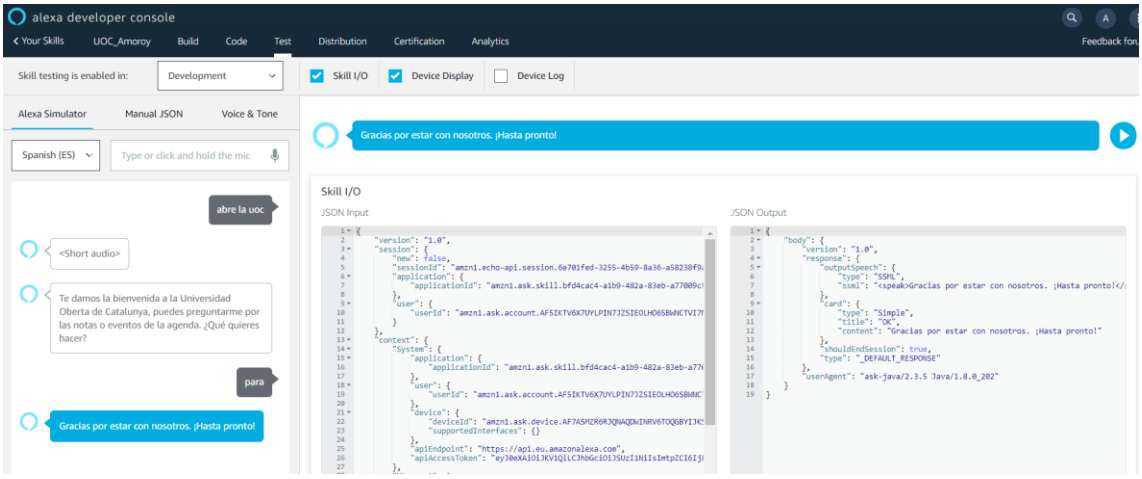


Ilustración 57 - Consola Alexa - Test

6.7. Errores y soluciones

- El “invocation name” de la skill está sin definir:

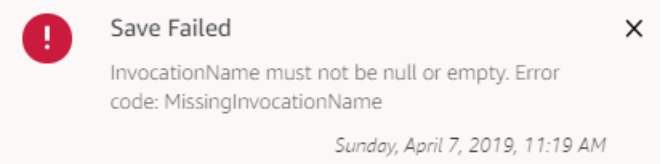


Ilustración 58 - Consola Alexa - Error Message

La solución es agregar el “invocaction name” en la sección de “Invocation”:

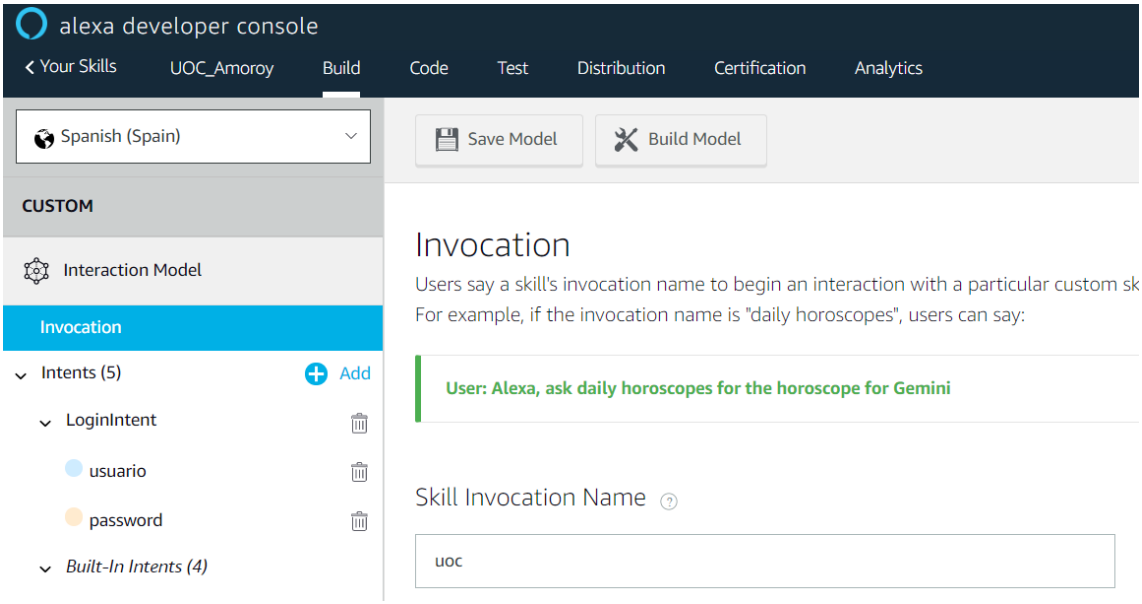
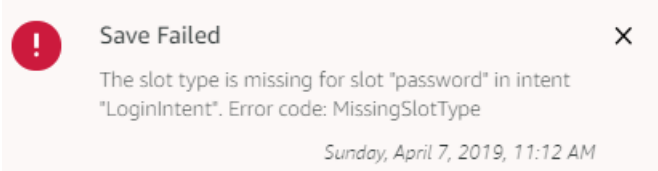


Ilustración 59 - Consola Alexa - Invocation Name Solution

- Error al intentar guardar una frase con una variable sin decirle el tipo de variable que es:



Este error se da cuando se ha incluido una variable {xxx} pero no se ha seleccionado el tipo. La solución es ir al Intent del que se queja y asignarle un tipo a la variable, bien sea creándolo nuevo o seleccionando uno de los que ya existe (como se explica en Slots (variables) y slot types):

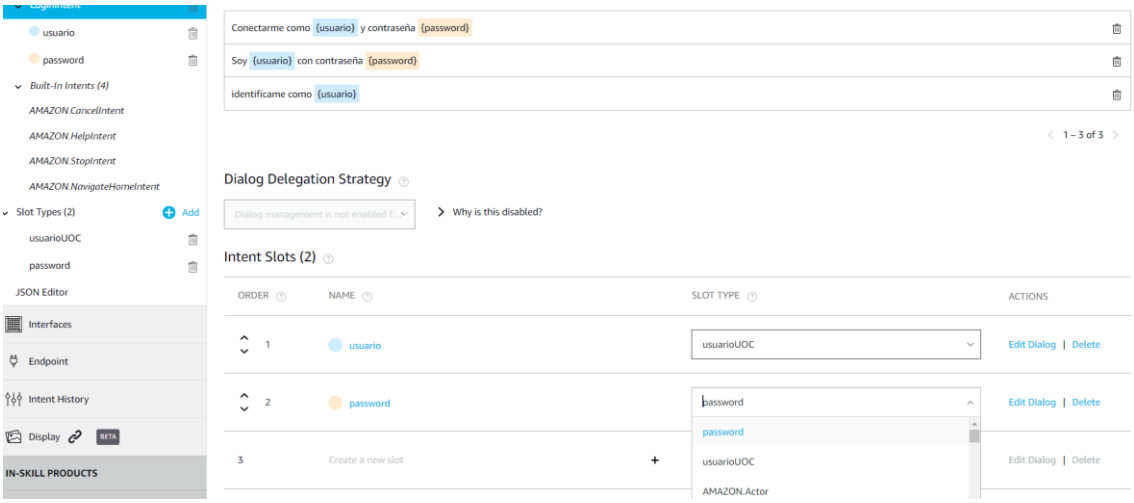


Ilustración 60 - Consola Alexa - Slot Values

7. Creación del backend de la Custom Skill

7.1. Primera prueba

Nota: para no hacer repetitivo este documento, solo se va a hacer un gran hincapié sobre cómo se aplica el TDD en esta primera prueba.

Siguiendo con la metodología de TDD, se comenzará el desarrollo de esta parte con un test que obligará a crear la clase principal donde se configuran todas las clases que van a existir para atender a Alexa:

```
1⊕ /**
4  package es.amoroy.uoc.ask;
5
6⊖ import org.junit.Before;
7  import org.junit.Test;
8
9⊖ /**
10  * Test para la configuración de Alexa
11  * @author amoroy
12  *
13  */
14 public class AlexaConfigTest {
15
16     AlexaConfig alexaConfig;
17
18⊖    @Before
19     public void setup() {
20         alexaConfig = new AlexaConfig();
21     }
22
23⊖    @Test
24     public void servletRegistrationOK() {
25         alexaConfig.registerSkillServlet();
26     }
27
28 }
29
```

The method registerSkillServlet() is undefined for the type AlexaConfig

3 quick fixes available:

- Change to 'registerSkillServlete(..)'
- Create method 'registerSkillServlet()' in type 'AlexaConfig'
- Add cast to 'alexaConfig'

Properties Servers Con

Ilustración 61 - Eclipse - TDD

Este test obliga a crear tanto la clase principal como el método encargado de registrar en Alexa todos los Intents. Así que con la ayuda del IDE Eclipse, se crea la clase y se aumenta el test para que vaya pidiendo lo necesitaría Alexa.

El test irá fallando y se irá agregando cada cosa. Y cada clase fuera de la que se está probando se ira mockeando (creando una respuesta simulada, ver Glosario, mockear) lo que sea necesario:

Runs: 1/1 Errors: 1 Failures: 0

es.amoroy.uoc.ask.AlexaConfigTest [Runner: JUnit 4] (0
servletRegistrationOK (0,038 s)

Failure Trace
java.lang.NullPointerException
at es.amoroy.uoc.ask.AlexaConfig.obtenerSkillId(AlexaConfig.java:81)
at es.amoroy.uoc.ask.AlexaConfig.registerSkillServlet(AlexaConfig.java:41)
at es.amoroy.uoc.ask.AlexaConfigTest.servletRegistrationOK(AlexaConfigTest.java:27)

Ilustración 62 - Eclipse - TDD - Result

Una vez se mockea lo que debe devolver ese método se ve que el código empieza a “coger forma” gracias a la prueba:

```
18  * @author amoroy
19  *
20  */
21 public class AlexaConfigTest {
22
23     AlexaConfig alexaConfig;
24
25     @Mock
26     Configuracion configuracion;
27
28     @Before
29     public void setup() {
30         MockitoAnnotations.initMocks(this);
31         alexaConfig = new AlexaConfig();
32         alexaConfig.setConfiguracion(configuracion);
33         Mockito.when(configuracion.getSkillId()).thenReturn("1234");
34
35     }
36
37     @Test
38     public void servletRegistrationOK() {
39         ServletRegistrationBean servletRegistration = alexaConfig.registerSkillServlet();
40         servletRegistration.getServlet().getServletInfo();
41     }
42
43 }
```

```
Properties  Servers  Console  Problems  Search  Git Repositories  Terminal  JUnit  History
terminated> AlexaConfigTest [JUnit] C:\Program Files\Java\jdk1.8.0_181\bin\javaw.exe (14 abr. 2019 11:50:05)
1:50:06.175 [main] INFO es.amoroy.uoc.ask.AlexaConfig - SKILL ID valido para este entorno: 1234
1:50:06.180 [main] INFO es.amoroy.uoc.ask.ResultadosAlexaSkillServlet - [getSkill] --- verificacion del SkillId
```

Ilustración 63 - Eclipse - TDD - Refactoring

```
Finished after 0,683 seconds
Runs: 1/1   Errors: 1   Failures: 0
es.amoroy.uoc.ask.AlexaConfigTest [Runner: JUnit 4] ((
  servletRegistrationOK (0,657 s)

Failure Trace
java.lang.IllegalArgumentException: handlerAdapters cannot be null
at com.amazon.ask.util.ValidationUtils.assertNotNull(ValidationUtils.java:40)
at com.amazon.ask.dispatcher.impl.DefaultRequestDispatcher.<init>(DefaultRequestDispatcher.java:61)
at com.amazon.ask.dispatcher.impl.DefaultRequestDispatcher$Builder.build(DefaultRequestDispatcher.java:251)
at com.amazon.ask.Skill.<init>(Skill.java:56)
at com.amazon.ask.builder.SkillBuilder.build(SkillBuilder.java:177)
at es.amoroy.uoc.ask.ResultadosAlexaSkillServlet.getSkill(ResultadosAlexaSkillServlet.java:63)
```

Ilustración 64 - Eclipse - TDD - Result 2

Y el código de AlexaConfig también va cogiendo forma:

```
20  *
21  * Clase de configuración de Alexa
22  *
23  * @author amoroy
24  *
25  */
26 @Setter
27 @Configuration
28 public class AlexaConfig {
29
30     /** Instancia de Logger */
31     private static final Logger LOG = LoggerFactory.getLogger(AlexaConfig.class);
32
33     @Autowired
34     private Configuracion configuracion;
35     @Autowired
36     private LaunchRequestHandler launchRequestHandler;
37     @Autowired
38     private CancelandStopIntentHandler cancelAndStopIntentHandler;
39     @Autowired
40     private SessionEndedRequestHandler sessionEndedRequestHandler;
41     @Autowired
42     private HelpIntentHandler helpIntentHandler;
43
```

```java
44    /**
45     * Metodo que registra el servlet
46     *
47     * @return ServletRegistrationBean
48     */
49    @Bean
50    public ServletRegistrationBean<AmoroySkillServlet> registerSkillServlet() {
51
52        AmoroySkillServlet servlet = new UOCAlexaSkillServlet(obtenerSkillId(), launchRequestHandler,
53                cancelAndStopIntentHandler, helpIntentHandler, sessionEndedRequestHandler, true);
54
55        configurarVerificacionFirma(servlet);
56        configurarVerificacionTimestamp(servlet);
57        ServletRegistrationBean<AmoroySkillServlet> servletRegistrationBean = new ServletRegistrationBean<>(servlet,
58                "/alexa");
59        return servletRegistrationBean;
60    }
```

Ilustración 65 - Eclipse - TDD - Refactoring 2

Aún sigue fallando el test aunque ya deja algunas trazas en la consola. El problema es que no se puede construir un CustomSkillBuilder con unos handlers de Alexa sin crear para la prueba.

Esto provoca que haya que mockear (simular) estos objetos ya que queda fuera de la competencia de esta prueba unitaria que solo debe probar la clase AlexaConfig.

7.2. Handlers de Alexa

Los handlers son clases por defecto de Alexa que se encargan de realizar acciones como dar la bienvenida cuando se abre la aplicación, o despedirse cuando se cierra.

Hay varios handlers:

- LaunchRequestHandler: es el encargado de dar la bienvenida. Se lanza cuando se abre la aplicación.
- CancelandStopIntentHandler: es el encargado de decir algo cuando el usuario cancela en medio de una interacción.
- HelpIntentHandler: es el encargado de guiar al usuario con que frases puede decir. Se lanza cuando el usuario pide ayuda.
- SessionEndedRequestHandler: es el encargado de despedir al usuario. Se lanza cuando el usuario cierra la aplicación.

Continuando con el TDD, se ha intentado agregar un solo handler para ver si la prueba pasaba se ha producido otro error:

```
Finished after 0,815 seconds
Runs: 1/1    Errors: 1    Failures: 0
es.amoroy.uoc.ask.AlexaConfigTest [Runner: JUnit 4] ((
  servletRegistrationOK (0,789 s)
Failure Trace
java.lang.IllegalArgumentException: handler cannot be null
at com.amazon.ask.util.ValidationUtils.assertNotNull(ValidationUtils.java:40)
at com.amazon.ask.dispatcher.request.handler.impl.GenericRequestHandlerChain.<init>(GenericRequestHandlerChain.java:34)
at com.amazon.ask.dispatcher.request.handler.impl.DefaultRequestHandlerChain.<init>(DefaultRequestHandlerChain.java:32)
at com.amazon.ask.dispatcher.request.handler.impl.DefaultRequestHandlerChain$Builder.build(DefaultRequestHandlerChain.java:56)
at com.amazon.ask.builder.SkillBuilder.lambda$getConfigBuilder$0(SkillBuilder.java:146)
at java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:193)
```

Ilustración 66 - Eclipse - TDD - Result 3

Así que hay que agregar todos los handlers posibles y por lo tanto crearlos con sus respectivas frases y configuraciones.

Y aunque aún no se esté comprobando nada en la prueba, la prueba ya ha obligado a crear varias clases y configuraciones para conseguir poner la prueba en verde:

```
27      AlexaConfig alexaConfig;
28
29      @Mock
30      Configuracion configuracion;
31
32      @Mock
33      LaunchRequestHandler launchRequestHandler;
34
35      @Mock
36      CancelandStopIntentHandler cancelAndStopIntentHandler;
37      @Mock
38      HelpIntentHandler helpIntentHandler;
39      @Mock
40      SessionEndedRequestHandler sessionEndedRequestHandler;
41
42      @Before
43      public void setup() {
44          MockitoAnnotations.initMocks(this);
45          alexaConfig = new AlexaConfig();
46          alexaConfig.setConfiguracion(configuracion);
47          alexaConfig.setLaunchRequestHandler(launchRequestHandler);
48          alexaConfig.setCancelAndStopIntentHandler(cancelAndStopIntentHandler);
49          alexaConfig.setHelpIntentHandler(helpIntentHandler);
50          alexaConfig.setSessionEndedRequestHandler(sessionEndedRequestHandler);
51          Mockito.when(configuracion.getSkillId()).thenReturn("1234");
52          Mockito.when(configuracion.isFlagDeshabilitarVerificacionFirma()).thenReturn(true);
53          Mockito.when(configuracion.isFlagDeshabilitarVerificacionTimestamp()).thenReturn(true);
54      }
55
56      @Test
57      public void servletRegistrationOK() {
58          ServletRegistrationBean servletRegistration = alexaConfig.registerSkillServlet();
```

Properties Servers Console Problems Search Git Repositories Terminal JUnit History

nished after 1,158 seconds

Runs: 1/1 Errors: 0 Failures: 0

es.amoroy.uoc.ask.AlexaConfigTest [Runner: JUnit 4] Failure Trace

Ilustración 67 - Eclipse - TDD - Refactoring 3

7.3. Siguiendo TDD

Y una vez que ya funciona el test sin errores, se refactoriza el código para seguir buenas prácticas como el principio de responsabilidad única (véase el Glosario) y separar cada responsabilidad en una clase separada.

Ahora ya se puede pedir mediante la prueba que verifique que se ha quedado registrado un servlet que se expone mediante la url “/alexa”:

```
45    @Before
46    public void setup() {
47        MockitoAnnotations.initMocks(this);
48        alexaConfig = new AlexaConfig();
49        alexaConfig.setConfiguracion(configuracion);
50        alexaConfig.setLaunchRequestHandler(launchRequestHandler);
51        alexaConfig.setCancelAndStopIntentHandler(cancelAndStopIntentHandler);
52        alexaConfig.setHelpIntentHandler(helpIntentHandler);
53        alexaConfig.setSessionEndedRequestHandler(sessionEndedRequestHandler);
54        Mockito.when(configuracion.getSkillId()).thenReturn("1234");
55        Mockito.when(configuracion.isFlagDeshabilitarVerificacionFirma()).thenReturn(true);
56        Mockito.when(configuracion.isFlagDeshabilitarVerificacionTimestamp()).thenReturn(true)

57
58    }
59
60    @Test
61    public void servletRegistrationOK() {
62        ServletRegistrationBean<?> servletRegistration = alexaConfig.registerSkillServlet();
63        Collection<String> urls = servletRegistration.getUrlMappings();
64        Assert.assertTrue("No contiene la url /alexa", urls.contains("/alexa"));
65    }
66
```

Console | Problems | Debug Shell | JUnit

Finished after 1,103 seconds

Runs: 1/1 Errors: 0 Failures: 0

es.amoroy.uoc.ask.AlexaConfigTest [Runner: JUnit 4] (1 | Failure Trace

Ilustración 68 - Eclipse - TDD - Refactoring 4

Este primer test ya ha obligado a crear varias clases, que por el momento no tienen test, y por lo tanto, no tienen funcionalidad. Así que hay que comenzar con el resto de test:

Primero la clase UOCAlexaSkillServlet, que será la encargada de construir el CustomSkillBuilder que Alexa necesita para “entenderse” con la parte backend.

```
AlexaConfigTest.java   UOCAlexaSkillServletTest.java   UOCAlexaSkillServlet.java
29      @Mock
30      private HelpIntentHandler helpIntentHandler;
31      @Mock
32      private SessionEndedRequestHandler sessionEndedRequestHandler;
33      private boolean deshabilitarVerificacionSkillId = true;
34
35      @Before
36      public void setup() {
37          MockitoAnnotations.initMocks(this);
38      }
39
40      @Test
41      public void createUOCAlexaSkillServletOK() {
42          uocAlexaSkillServlet = new UOCAlexaSkillServlet(skillId, launchRequestHandler, cancel
43                  helpIntentHandler, sessionEndedRequestHandler, deshabilitarVerificacionSkillI
44          Assert.assertNotNull(uocAlexaSkillServlet);
45      }
46  }
47
Console   Problems   Debug Shell   JUnit
Finished after 1,125 seconds
Runs: 1/1   Errors: 0   Failures: 0
es.amoroy.uoc.ask.UOCAlexaSkillServletTest [Runner: J   Failure Trace
```

Ilustración 69 - Eclipse - TDD - Refactoring 5

Y partir de aquí se irán creando todos los test que necesite la aplicación como por ejemplo todo lo que se haya mockeado, habrá que crearlo a partir de un test. Y así ya se consigue la primera funcionalidad, la de dar la bienvenida al abrir la aplicación (ademas de la ayuda, cancelación y finalización de la sesión).

8. Test desde la consola de Alexa

Alexa permite probar tanto la configuración que se realiza desde la consola como el propio backend mediante la pestaña “Test” de la consola de Alexa:

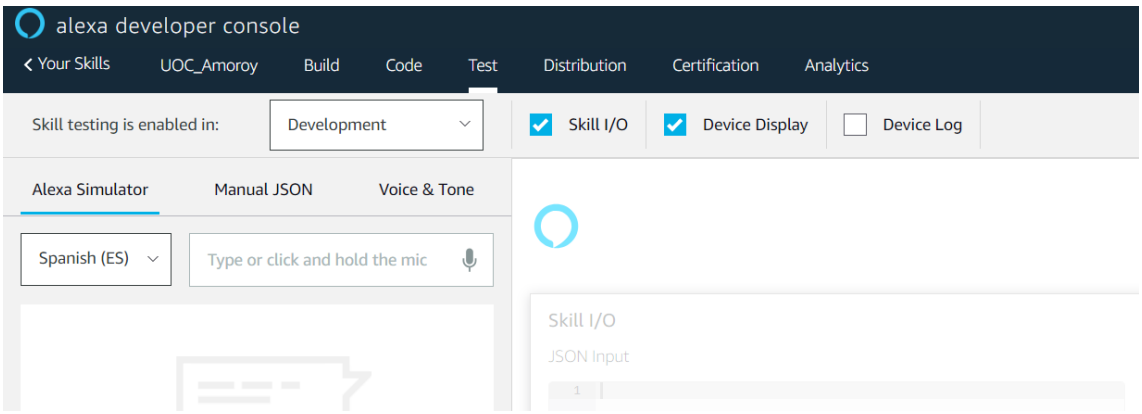


Ilustración 70- Consola Alexa - Test 2

Esta pestaña permite probar la interacción que haría el usuario con los Echo de Amazon y ver la respuesta que daría la skill que se esta probando.

De esta manera se puede simular todo tipo de interacciones, tanto escritas como habladas (mediante un micro). Y la consola pintará lo que ha entendido y la respuesta que dará la skill:

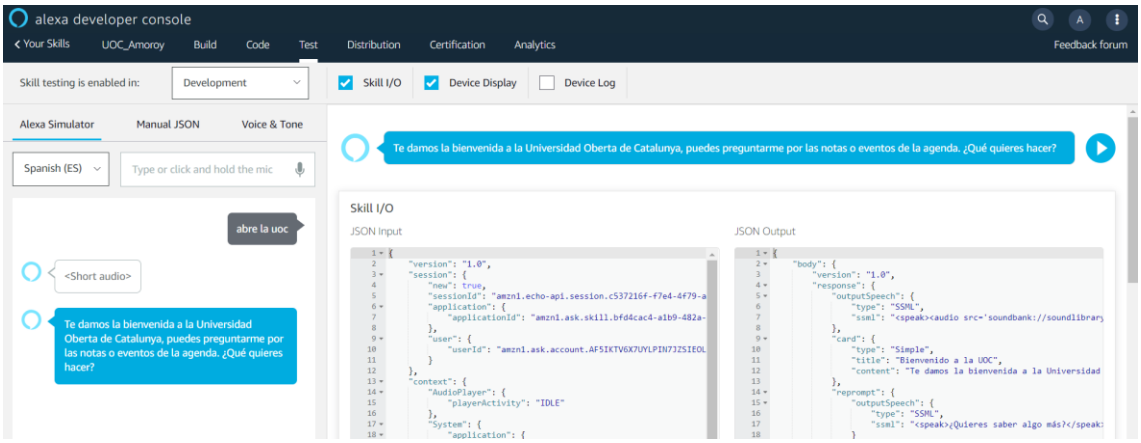


Ilustración 71- Consola Alexa - Test 3

En el panel de la izquierda se puede escribir lo que se desee probar o pulsar sobre el micrófono y pedirlo. La consola dirá la respuesta como si del propio Echo se tratara y la mostrara en ese panel.

Ilustración 72- Consola Alexa - Test 4

En el panel de la derecha se muestran los objetos json que se envía al backend (primer recuadro) y el objeto json que devuelve el backend a Alexa (recuadro de la derecha).

En caso de que se produzca un error, se puede analizar que es lo que le llega al backend y así intentar resolver dicho problema. Por otro lado, el propio log del servidor cloud debería aportar información suficiente si se ha seguido una buena trazabilidad del sistema.

Otra parte interesante y que permite sacarle mucho más partido a la interacción con los usuario es la parte de pruebas de como dice Alexa las frases.

Alexa permite configurar mediante etiquetas xml (explicadas en el siguiente apartado) el comportamiento de la voz que responde. Por ejemplo se puede hacer que susurre, que haga una pausa, que ponga especial énfasis en algo…

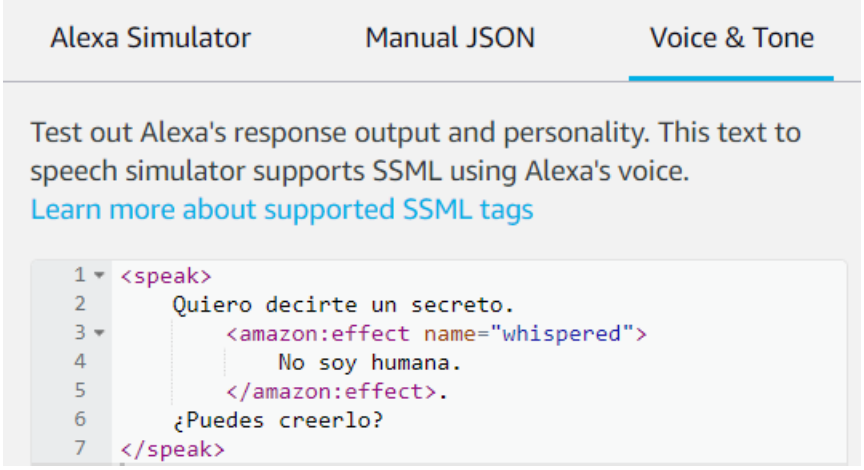


Ilustración 73- Consola Alexa - Test 5

Desde esta consola se podrá ver el comportamiento real que tendrá Alexa con la utilización de esas etiquetas. En este caso se probará como Alexa susurra.

Pero también sirve, no solo para los efectos, sino para ver que se ha hecho un uso de la puntuación correcta. Por ejemplo, si no se pusiera el punto despues de secreto, diría la frase y el susurro seguido, sin una pausa. Esto provocaría una reacción negativa del usuario.

Como conclusión de este apartado de test desde la consola de Alexa, se puede ver lo útil y potente que es, así como evitar la necesidad de tener un dispositivo Echo para probar las skills.

9. Intents (funcionalidades) de la aplicación

Los Intents son las funcionalidades que se declaran tanto en la parte de Alexa, donde se especifican las utterances (frases) y slots (variables), como en el backend, donde se realiza la conexión con los servicios y se devuelve lo que Alexa debe decir y cómo debe decirlo.

9.1. Configuraciones de respuesta de Alexa

Alexa tiene infinidad de formas y combinaciones para decir las respuestas de la mejor manera posible. Estas configuraciones se especifican en el backend mediante etiquetas xml o funcionalidades.

Alexa permite realizar acciones como pausas, introducir sonidos, poner énfasis a una parte de una frase… Y todas estas accione se especifican mediante código.

En este apartado se va a explicar algunas de estas posibilidades y algunos ejemplos. Aunque estos ejemplos no tienen por qué aparecer en el código de esta aplicación de Alexa.

9.1.1. Efectos

Los efectos son comportamientos de la voz. Estos efectos se realizarán sobre las frases que vayan a decir y ofrecen un acercamiento al comportamiento humano.

Amazon Alexa llama a este apartado Lenguaje de marcado de síntesis, en inglés Speech Synthesis Markup Language (SSML) [7] y describe cómo debe usarse y cuando es mejor usarlo (en Bibliografía se puede ver el enlace a la documentación).

Claramente no es lo mismo decir una frase de seguido que realizar una serie de efectos sobre esta frase como son las pausas o las diferentes entonaciones.

Por ejemplo, no es lo mismo si se quiere decir la frase “Has sacado una A en Análisis Matemático” normal que decirla dejando una pequeña pausa antes d la “A” y poner un énfasis en dicha nota.

Table of Contents

- Using SSML in your response
- Supported SSML tags
- amazon:effect
- audio
- break
- emphasis
- lang
- p
- phoneme

- prosody
- s
- say-as
- speak
- sub
- voice
- w
- Other SSML Reference Materials

Ilustración 74 - Alexa Documentation - Effects

Cada apartado tiene su forma específica de uso y se debe seguir esta manera con los consejos que se dan a lo largo de la documentación.

Por ejemplo se puede introducir pausas que interesan para que el usuario se entere de lo que acaba de oír sin añadirle más información antes de que lo pueda procesar:

```
<speak>
    Aquí se mete una pausa de tres segundos <break time="3s"/>
    Y después se continúa.
</speak>
```

O para el caso que se ha hablado de las notas se puede poner énfasis únicamente en la nota para cuando es una buena nota (o incluso al revés para cuando la nota es mala):

```
<speak>
   Has sacado una
    <emphasis level="strong">A en Análisis Matemático.</emphasis>
</speak>
```

9.1.2. Sonidos

Alexa ofrece una serie de sonidos de serie que pueden ser utilizados por las aplicaciones. Los sonidos aportan identidad a las aplicaciones ya que los usuarios suelen identificar dichos sonidos con las aplicaciones.

Alexa ofrece una documentación específica para el tratamiento de sonidos de donde se ha sacado esta información (se puede ver en el apartado de Bibliografía [1]).

La manera de agregar un sonido a la respuesta es añadiendo una etiqueta xml donde se le especifique la url donde se encuentra el sonido que ofrece Alexa. Y siguiendo el ejemplo que especifican en la documentación, es el siguiente:

“<speak>Esto es un ejemplo de cómo usar un sonido en Alexa. En este caso el rugido de un oso.

<audio src='soundbank://soundlibrary/animals/amzn_sfx_bear_groan_roar_01'/> </speak>”

Alexa ofrece una serie de sonidos que se pueden encontrar en su documentación. Primero la categoría de los sonidos y dentro todos los sonidos y como se usan:

The sounds are organized into the following categories:

- Ambience Sounds
- Animal Sounds
- Battle Sounds
- Cartoon Sounds
- Foley Sounds
- Game Show Sounds
- Home Sounds
- Human Sounds
- Impact Sounds
- Magic Sounds
- Musical Sounds
- Nature Sounds
- Office Sounds
- SciFi Sounds
- Transportation Sounds

Ilustración 75 - Alexa Documentation - Sounds

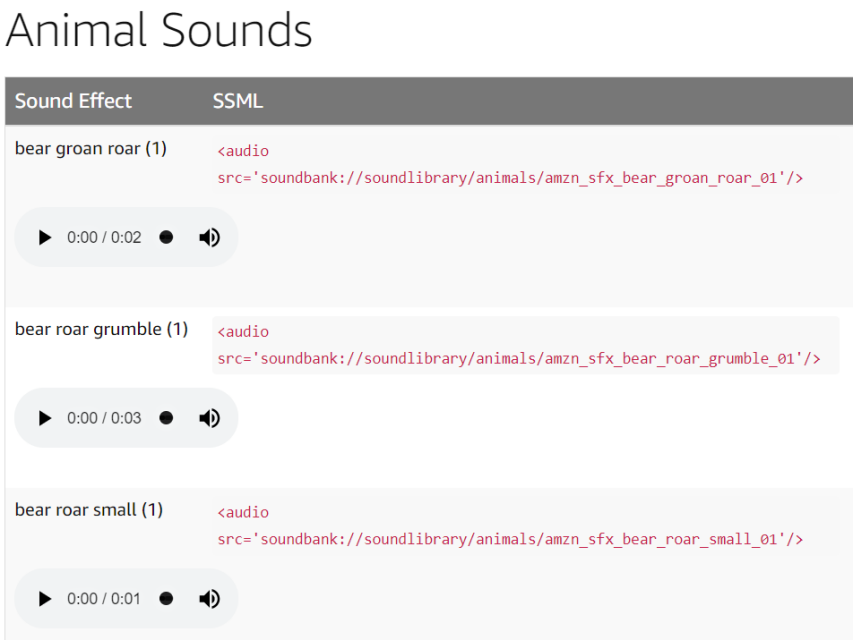


Ilustración 76 - Alexa Documentation - Sounds 2

9.1.3. Comportamiento de ALEXA

Otra de las configuraciones que se puede hacer con Alexa es especificar si se quiere que el usuario pueda seguir hablando a Alexa después de devolver la frase y si se desea que se devuelva alguna frase si el usuario no dice nada.

Estas configuraciones se hacen a nivel del objeto que se le devuelve a Alexa, donde entre otras cosas, se encuentra la frase con la que se contestará al usuario.

```
return
input.getResponseBuilder().withSpeech(speechText).withSimpleCard(literalesCardTitle.getLogin(),
conversorCard.convertir(speechText)).withShouldEndSession(false).withReprompt(literales.getPreguntaSaberMas()).build();
```

Si se analiza este ejemplo (extraído de la clase LoginItent) se puede observar que tiene cuatro partes personalizables que se van a explicar individualmente:

- withSpeech(): es el método que permite especificar la frase que va a decir Alexa cuando se le devuelva este objeto
- withSimpleCard(): es el método que permite especificar tanto el título como la frase que se va a mostrar por pantalla. En este proyecto se ha creado un conversor para que no salgan por pantalla los tags de xml y solo salga la frase.
- withShouldEndSession(): este método específica a Alexa si tiene que permitir al usuario continuar hablando o no. Se utilizará “false” para que se permita continuar hablando y “true” para que se termine la sesión cuando Alexa diga la frase.

- withReprompt(): este método sirve para que pasado un tiempo (tiempo que es configurable, por defecto son 10 segundos) si el usuario no ha contestado cuando la sesión sigue abierta, Alexa dirá la frase que se le pase como parámetro a este método. Es una buena práctica del desarrollo de aplicaciones de Alexa utilizar este método siempre que se ponga withShouldEndSession(false).

9.2. LoginIntent

La parte de la creación del Intent en Alexa está explicada en el apartado Creación de la Custom Skill. Por lo que en este Intent se partirá directamente del backend

Esta funcionalidad consiste en simular el apartado de Login ya que en este proyecto no da tiempo de configurar un login de Alexa. Por lo tanto se implementa un nuevo Intent con dicha funcionalidad.

Esta funcionalidad consiste en permitir a dos usuarios identificarse en la aplicación. Los dos usuarios son grobles y amoroy que corresponden a Gregorio y Alvaro respectivamente.

Una vez creado el Intent en la parte de Alexa, en la parte del backend habrá que crear dicho Intent y “publicarlo”, o lo que es lo mismo, decirle a Alexa que ese Intent está disponible.

Para hacerlo, cuando se configura el proyecto de backend para exponer la funcionalidad se debe añadir el Intent al objeto que ejerce de entrada al backend del proyecto.

Si se recuerda del apartado de “Creación del backend de la Custom Skill” la clase donde se va a exponer un bean que hará de nexo de unión entre Alexa y el backend será “AlexaConfig”:

```
@Bean
public ServletRegistrationBean<AmoroySkillServlet> registerSkillServlet() {
        AmoroySkillServlet servlet = new
UOCAlexaSkillServlet(obtenerSkillId(), launchRequestHandler,
   cancelAndStopIntentHandler, helpIntentHandler, sessionEndedRequestHandler,
loginIntent, true);
        configurarVerificacionFirma(servlet);
        configurarVerificacionTimestamp(servlet);
        ServletRegistrationBean<AmoroySkillServlet> servletRegistrationBean
= new ServletRegistrationBean<>(servlet, "/alexa");
        return servletRegistrationBean;
   }
```

Se puede ver remarcado en amarillo el Intent que se desea añadir, y por lo tanto, Alexa ya lo tendrá en cuenta en caso de que se necesite contestación de ese Intent.

Si se va a más bajo nivel del código se verá que en la clase UOCAlexaSkillServlet está el método que se encargará de decirle a Alexa que ese Intent este disponible mediante el objeto CustomSkillBuilder y el método addRequestHandlers:

```
private static Skill getSkill(String skillId, LaunchRequestHandler
launchRequestHandler, CancelandStopIntentHandler cancelAndStopIntentHandler,
HelpIntentHandler helpIntentHandler, SessionEndedRequestHandler
sessionEndedRequestHandler, LoginIntent loginIntent,
                boolean deshabilitarVerificacionSkillId) {
            CustomSkillBuilder csb = new CustomSkillBuilder();
            csb.addRequestHandlers(launchRequestHandler,
cancelAndStopIntentHandler, helpIntentHandler,
                        sessionEndedRequestHandler, loginIntent);
            LOG.info("[getSkill] --- verificacion del SkillId");
            if (!deshabilitarVerificacionSkillId) {
                csb.withSkillId(skillId);
            }
            return csb.build();
    }
```

Por otra parte se creará la clase LoginIntent que extienda de RequestHandler para poder manejar las peticiones de Alexa para estos casos.

En esta clase, lo primero que hay que hacer es sobrescribir el método canHandle que será el encargado de decir si el Intent que ha elegido Alexa se corresponde con el que este:

```
@Override
public boolean canHandle(HandlerInput input) {
    return input.matches(intentName("LoginIntent"));

}
```

En caso positivo, se invocará al método handle de este Intent (que se deberá sobrescribir también) y que será el encargado de recoger las variables que envía Alexa y hacer la petición a los servicios de la UOC.

```
@Override
public Optional<Response> handle(HandlerInput input) {
    LOG.info("-------------- Entrando en LoginIntent ");
    Request request = input.getRequestEnvelope().getRequest();
    LOG.info("-------------- request:t " + request);
    IntentRequest intentRequest = (IntentRequest) request;
    LOG.info("-------------- intentRequest: " + intentRequest);
    Intent intent = intentRequest.getIntent();
    LOG.info("-------------- intent: " + intent);
    Map<String, Slot> slots = intent.getSlots();
    LOG.info("-------------- slots: " + slots);
    LOG.info("-------------- slots.get(\"usuario\").getValue(): " + slots.get("usuario").getValue());

    UsuarioUOCEnum usuarioUOC = UsuarioUOCEnum.fromString(slots.get("usuario").getValue());
    LOG.info("El usuario es:" + usuarioUOC);
    String passwordUOC = slots.get("password").getValue();
    String speechText;

    if (usuarioUOC == null) {
        speechText = literalesErrores.getUsuarioNoReconocido();
        LOG.info("-------------- Texto salida para LoginIntent con error: " + speechText);
        return input.getResponseBuilder().withSpeech(speechText)
                .withSimpleCard(literalesCardTitle.getLogin(), conversorCard.convertir(speechText))
                .withShouldEndSession(false).withReprompt(literales.getPreguntaSaberMas()).build();
    } else {
        speechText = servicio.doLogin(usuarioUOC.name(), passwordUOC);
        LOG.info("-------------- Texto salida para el Login: " + speechText);
        return input.getResponseBuilder().withSpeech(speechText)
                .withSimpleCard(literalesCardTitle.getLogin(), conversorCard.convertir(speechText))
                .withShouldEndSession(false).withReprompt(literales.getPreguntaSaberMas()).build();
    }

}
```

Ilustración 77- Eclipse - Code

En este código se ve cómo se recogen las variables que envía Alexa y se utilizan para llamar al servicio que realizará –de forma simulada– login en la aplicación.

Finalmente, se construirá la respuesta con todas las configuraciones como lo que se mostrará por pantalla, el mantener la sesión abierta y la frase que se dirá en caso de que el usuario no conteste.

También se puede ver como se utilizan clases para cada apartado, separando las responsabilidades y siguiendo las buenas prácticas de desarrollo.

Una vez terminadas las pruebas y el código necesario para las pruebas –siguiendo el caso de uso–, se puede exponer el servicio y probarlo desde la consola de Alexa o incluso desde el propio dispositivo Echo:

Ilustración 78 - Consola Alexa - Test Login

Como se aprecia, el servicio es capaz de transformar el usuario pedido (amoroy) por el nombre real del mismo (Alvaro). Ahora toca hacer alguna mejora para la interacción correcta con el usuario, por ejemplo que pregunte después “¿En qué puedo ayudarte?”.

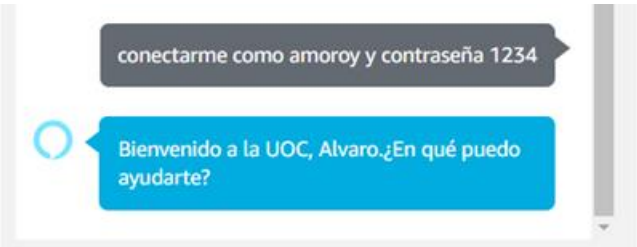


Ilustración 79 - Consola Alexa - Test Login 2

Ahora ya se dispone del LoginIntent y se puede pasar a realizar otros Intent. Siempre se puede mejorar cualquier Intent, y de hecho, cuando se pasa a certificar la aplicación, los encargados de Amazon piden mejoras tanto obligatorias como opcionales que conviene seguir.

9.3. NotasIntent

Este Intent será el encargado de decir que notas ha sacado hasta el momento el alumno, bien sea de una asignatura como de todas las que está cursando en este momento.

El primer paso es crear el Intent en la parte de Alexa tal y como se explicó en el apartado “Intents (funcionalidades) en Alexa”. Aquí se expondrán las partes clave de este Intent.

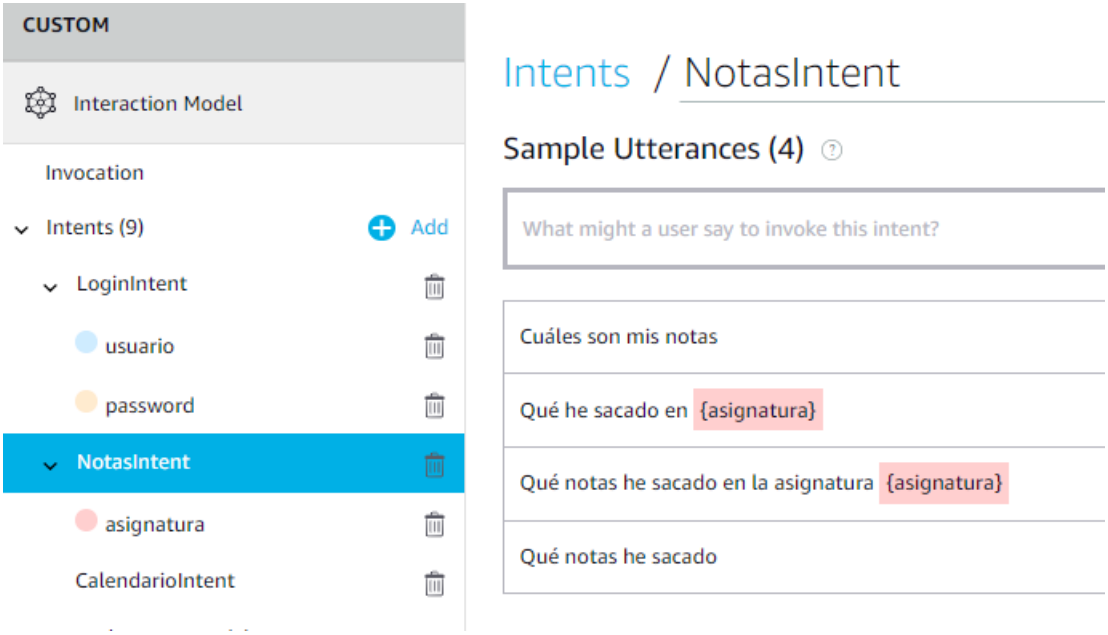


Ilustración 80 - Consola Alexa - NotasIntent

Slot Types / asignatura

Slot Values (5)

Enter a new value for this slot type

VALUE	ID (OPTIONAL)
ingeniería del software	Enter ID
estadística	Enter ID
física	Enter ID
álgebra	Enter ID
análisis matemático	Enter ID

Ilustración 81 - Consola Alexa - Slot Asignatura

Una vez se ha creado el Intent con las utterances, los slots y slot types, ya se puede ir a la parte del backend para crear el servicio rest que atienda esta petición de Alexa.

En el backend, se crea un test, y por tanto la implementación, tanto para el servicio que simularía la conexión con los servicios de la UOC, como otra para el propio Intent con la recogida de los parámetros que envía Alexa:

```
/**
 * metodo de entrada para el intent
 *
 * @param input input de entrada
 * @return se devuelve una respuesta con el texto a Alexa
 */
@Override
public Optional<Response> handle(HandlerInput input) {
    LOG.info("-------------- Entrando en NotasIntent ");
    Request request = input.getRequestEnvelope().getRequest();
    LOG.debug("-------------- request: " + request);
    IntentRequest intentRequest = (IntentRequest) request;
    LOG.debug("-------------- intentRequest: " + intentRequest);
    Intent intent = intentRequest.getIntent();
    LOG.info("-------------- intent: " + intent);
    Map<String, Slot> slots = intent.getSlots();
    LOG.info("-------------- slots: " + slots);
    Slot asignatura = slots.get("asignatura");
    String asignaturaUOC = "";
    if (asignatura != null && asignatura.getValue() != null) {
        LOG.debug("-------------- slots.get(\"asignatura\").getValue(): " + asignatura.getValue());
        asignaturaUOC = slots.get("asignatura").getValue();
    }
```

```
    LOG.info("La asignatura es:" + asignaturaUOC);
    String speechText;
    speechText = servicio.getNotas(asignaturaUOC);
    LOG.info("-------------- Texto salida para Notas: " + speechText);
    return input.getResponseBuilder().withSpeech(speechText)
            .withSimpleCard(literalesCardTitle.getNotas(), conversorCard.convertir(speechText))
            .withShouldEndSession(false).withReprompt(literales.getPreguntaSaberMas()).build();

}
```

Ilustración 82 - Eclipse - Code 2

Una vez se tiene todo el código probado y publicado en el servidor cloud –siguiendo el caso de uso–, ya se puede probar con la consola de Alexa para verificar que el resultado es el que se desea:

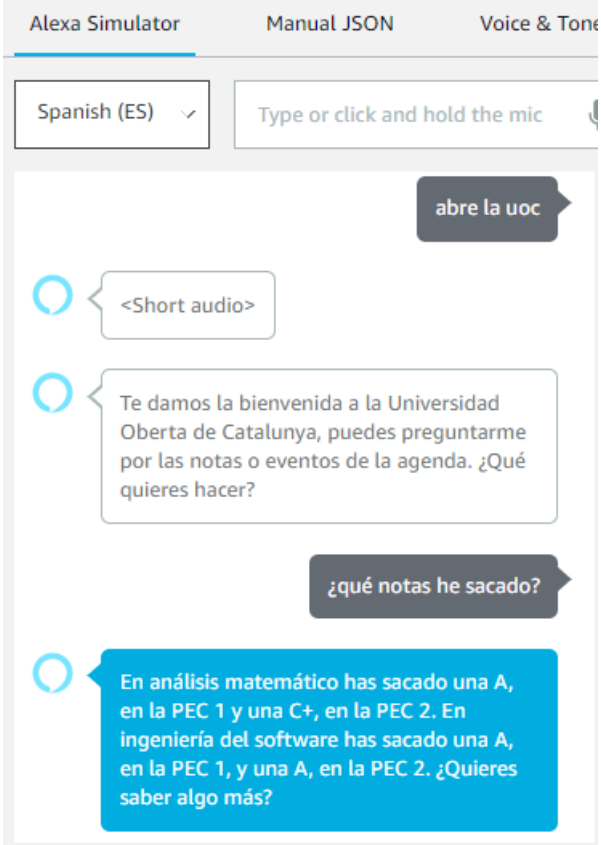


Ilustración 83 - Consola Alexa - Test Notas

Para cuando se pide una nota de una asignatura en concreto se ha agregado una aleatoriedad tanto de las notas como de la actividad que dice Alexa:

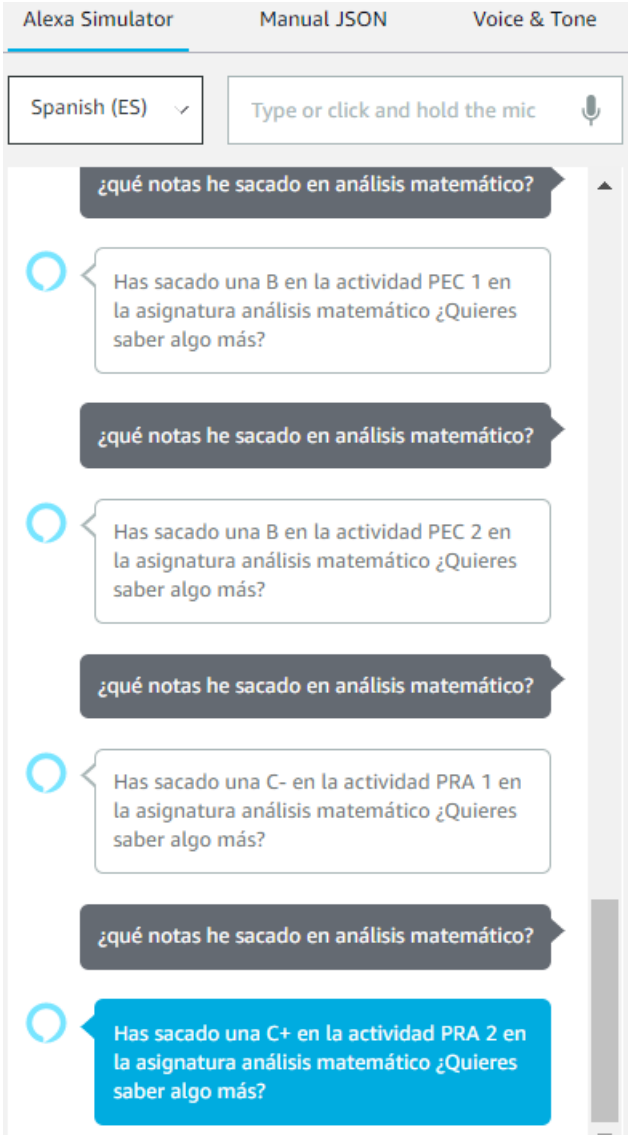


Ilustración 84 - Consola Alexa - Test Notas 2

9.4. CalendarioIntent

El calendarioIntent será el encargado de proporcionar la agenda del alumno para la próxima semana. Así el usuario sabrá lo que tiene que entregar para poder planificarse.

Así, el primer paso vuelve a ser crear el Intent en Alexa, con sus utterances (frases) y, en esta ocasión, sin necesidad de utilizar slots (variables) para este Intent.

Una novedad que se va a utilizar para este Intent es añadir un diálogo con Alexa en el que Alexa pregunte si quiere saber los eventos de los próximos 7 días:

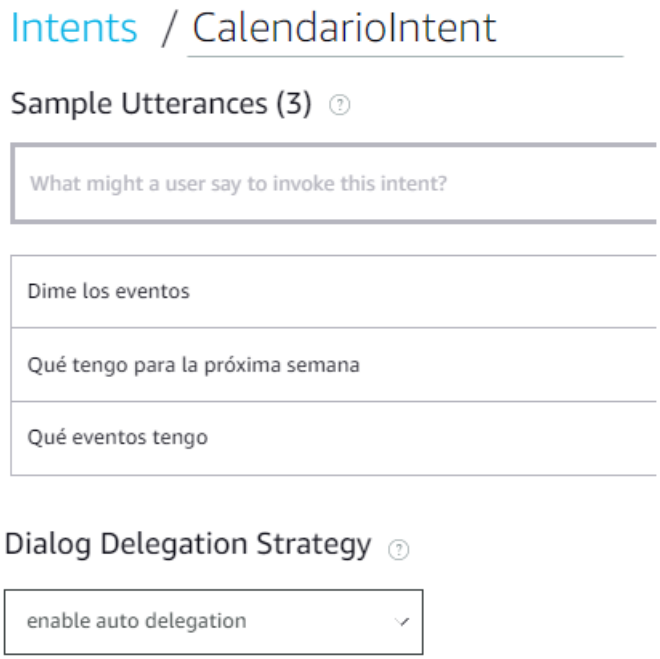
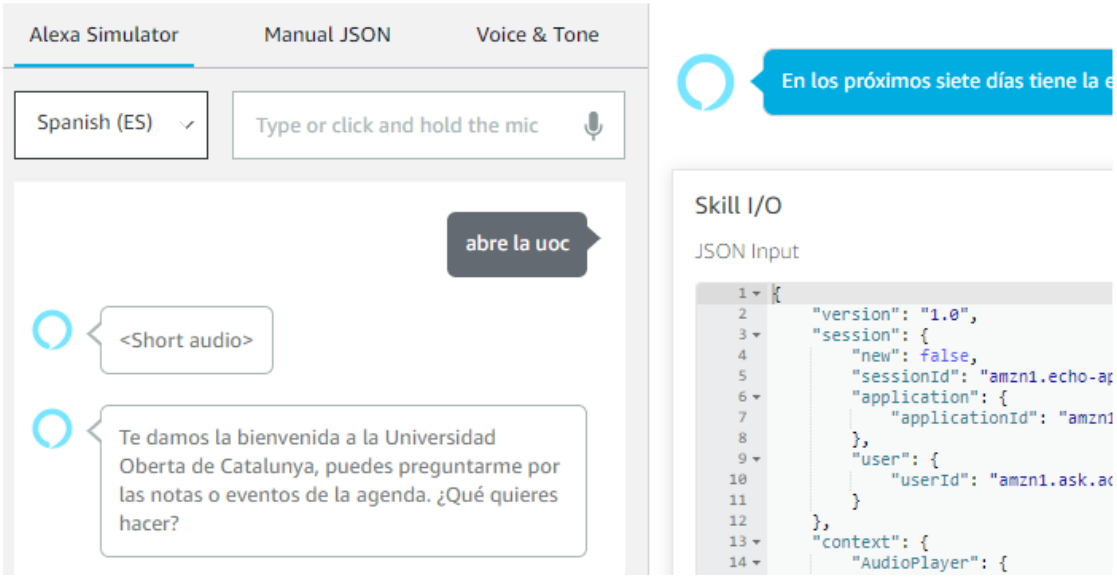


Ilustración 85 - Consola Alexa - CalendarioIntent

Una vez realizado el backend –siguiendo el caso de uso– al que solo se llegará si se ha contestado a Alexa sobre si se quiere los eventos de la próxima semana, ya se puede probar la aplicación:



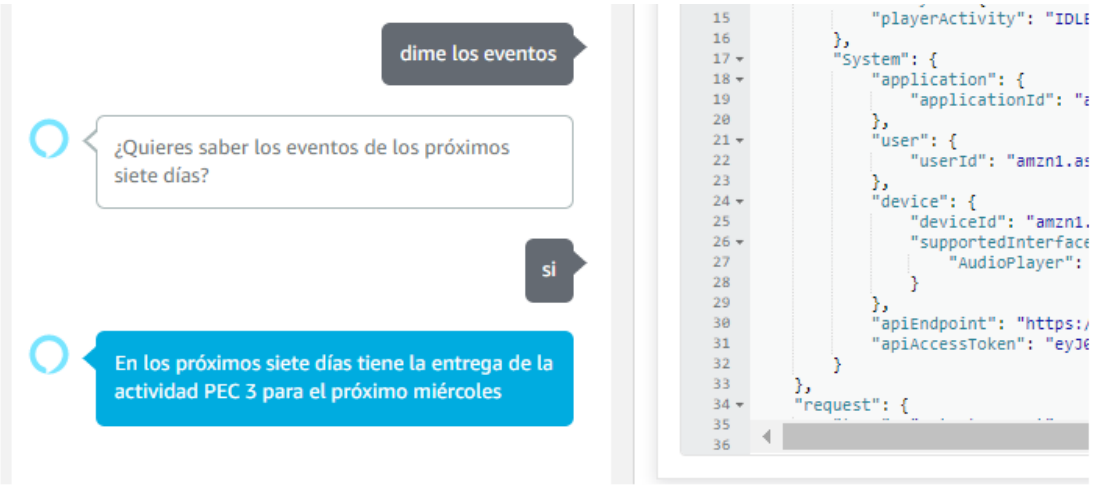


Ilustración 86 - Consola Alexa - Test Calendario

9.5. NoIntentHandler

Para terminar con los Intent, se va a implementar el Intent de Alexa que atiende a todas las peticiones que sean una negación a una pregunta. Si Alexa pregunta "¿Quieres hacer algo más?" y el usuario responde "No", este será el Intent que se active.

Así se puede personalizar el mensaje cuando el usuario no quiera continuar con la aplicación y despedirse de él en nombre de la UOC con un mensaje del estilo: "Gracias por estar con la Universidad Oberta de Catalunya. ¡Hasta pronto!".

En la parte de la consola de Alexa basta con añadir el Intent propio de Alexa, ya que Alexa proporciona varios Intent ya configurados que suelen ser comunes:

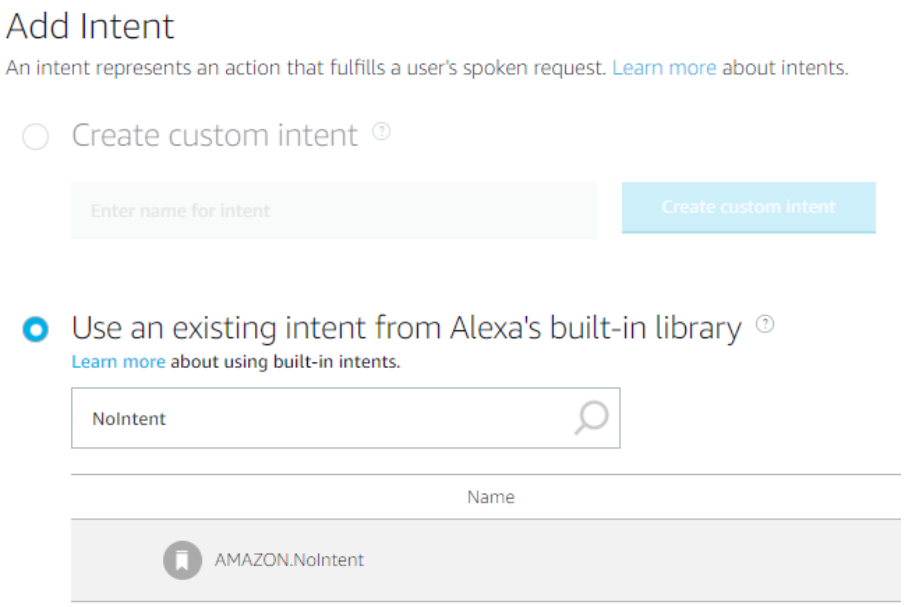


Ilustración 87 - Consola Alexa - NoIntent

Y construir en la parte backend exactamente igual que el resto de Intent pero con el mensaje personalizado que se desea decir al usuario. Así, una vez se han realizado estas dos partes ya se puede observar el resultado:

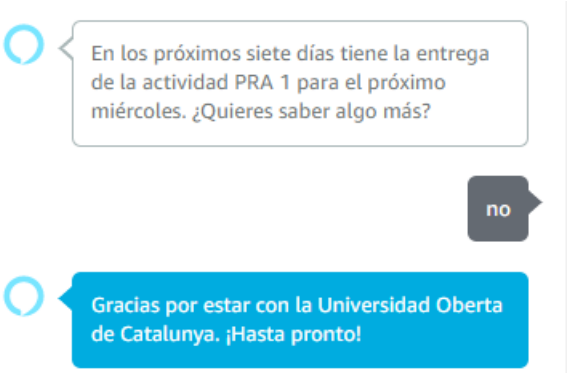


Ilustración 88 - Consola Alexa - Test NoIntent

10. Conectar Skill en desarrollo con el dispositivo Echo

La propia consola de Alexa trae un apartado muy potente para probar las Skills, como ya se ha explicado anteriormente. Tanto es así que no es necesario tener un dispositivo Echo para poder saber cuál será el comportamiento de la Skill.

Las Skills deben pasar por un proceso de certificación para poder publicarse –al estilo de las app en Google Play o Apple Store–. Por este motivo no es posible añadir al dispositivo Echo, de forma normal, una Skill en desarrollo que no haya pasado la certificación.

Pero para eso Amazon ofrece la posibilidad de probar contra el dispositivo Echo cualquier Skill en desarrollo sin la necesidad de pasar por la certificación.

Esto lo hace mediante una opción de Beta test. La consola de Alexa permite agregar cuentas (correos electrónicos) que hagan de Beta test y puedan agregar la Skill en desarrollo y probarla en su dispositivo Echo:

Para conseguirlo hay que ir a la pestaña “Distribution” y rellenar todos los campos:

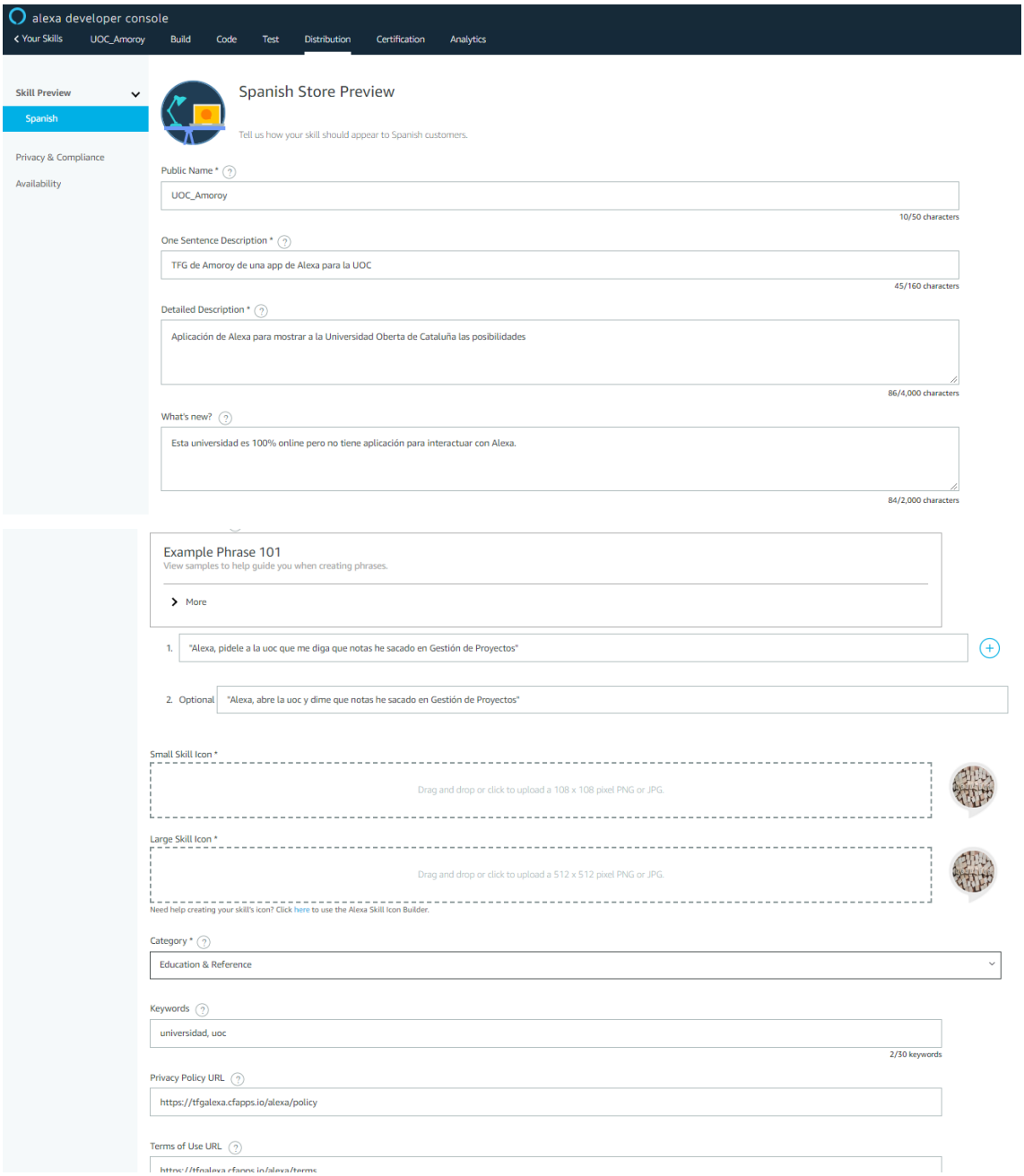


Ilustración 89 - Consola Alexa, pestaña "Distribution" [6]

Nota: es muy importante rellenar todos los campos, incluidos los campos de las url donde se encuentran la política de privacidad y los términos de uso (por eso en este proyecto se ha tenido que añadir un controlador para exponer dichos campos).

Una vez se han rellenado todos los campos se pulsa sobre “Save & continue” para pasar a la siguiente fase:

Ilustración 90 - Consola Alexa - Distribution Save

En esta fase se configurará algunos puntos importantes para las Skills, donde se dirá si la aplicación tiene pagos, usa datos de usuarios, etc.

En este proyecto, y para poder pasar a la siguiente fase sin problemas se ha puesto que no a todo, pero en una aplicación real hay que tener en cuenta que sí que se gestionarían datos personales.

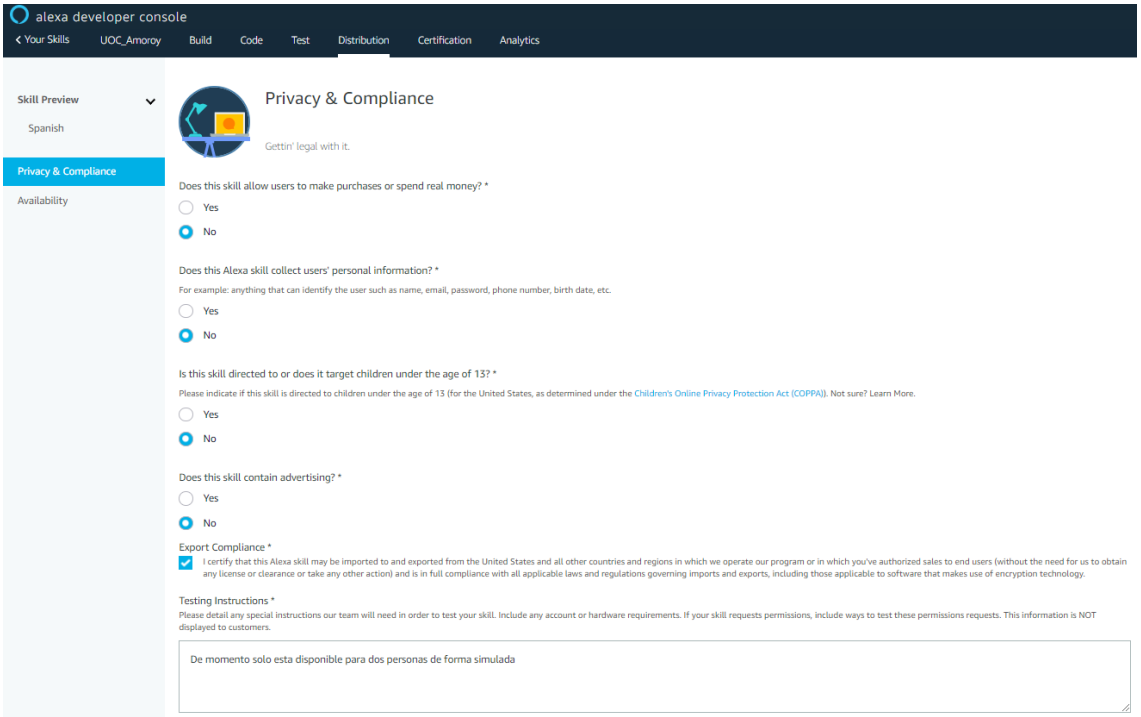


Ilustración 91 - Consola Alexa - Privacy

En el siguiente apartado es donde se debe configurar la opción de Beta Test. Aquí se agregarán las cuentas para poder probar la Skill en el dispositivo Echo configurado con esas cuentas:

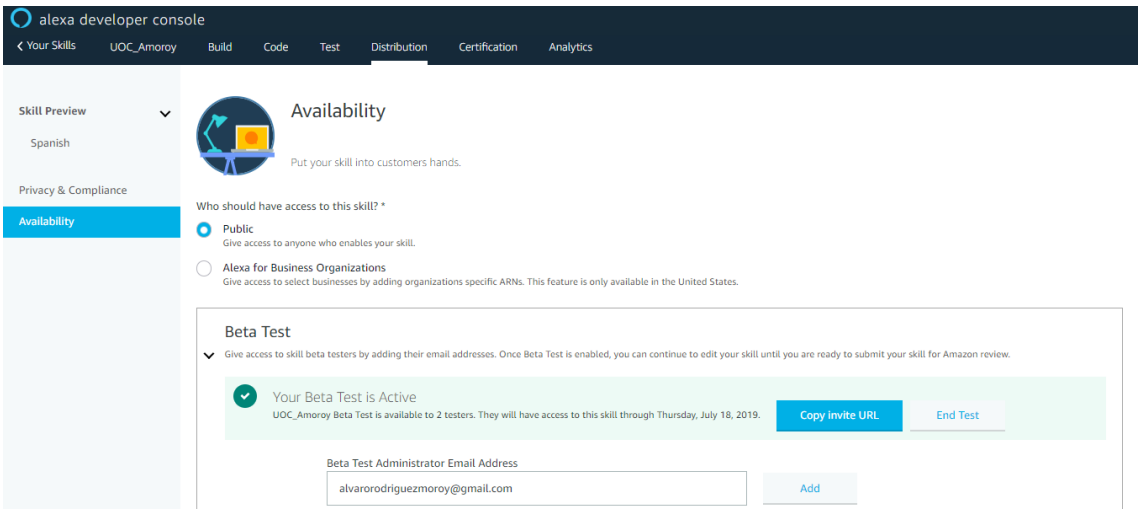


Ilustración 92 - Consola Alexa - Availability

Una vez enviada la invitación al correo del usuario, se podrá acceder a una url que contendrá dicha aplicación en desarrollo:

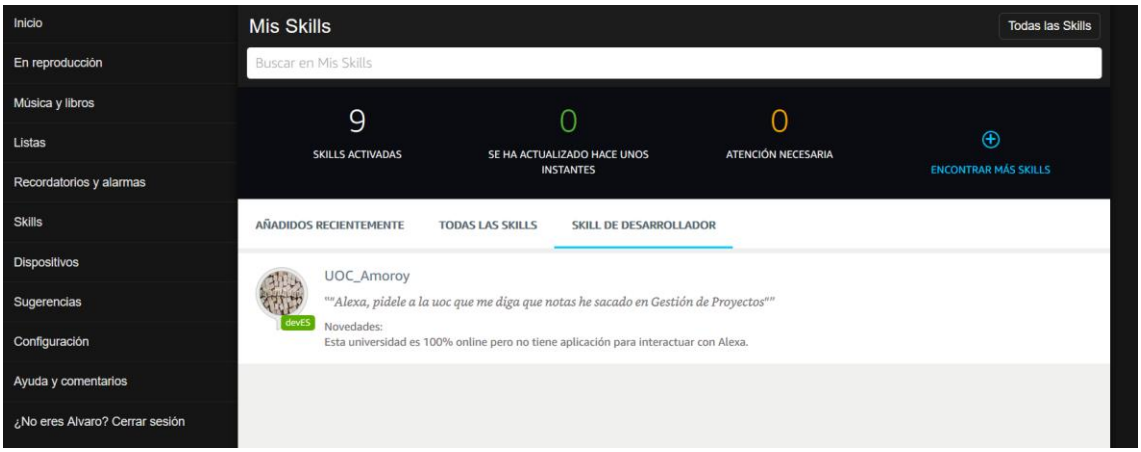


Ilustración 93 - App Alexa - UOC_Amoroy

Y cuando se haya aceptado la invitación saldrá reflejado en la consola de Alexa que esa cuenta de correo electrónico ha aceptado ya la invitación:

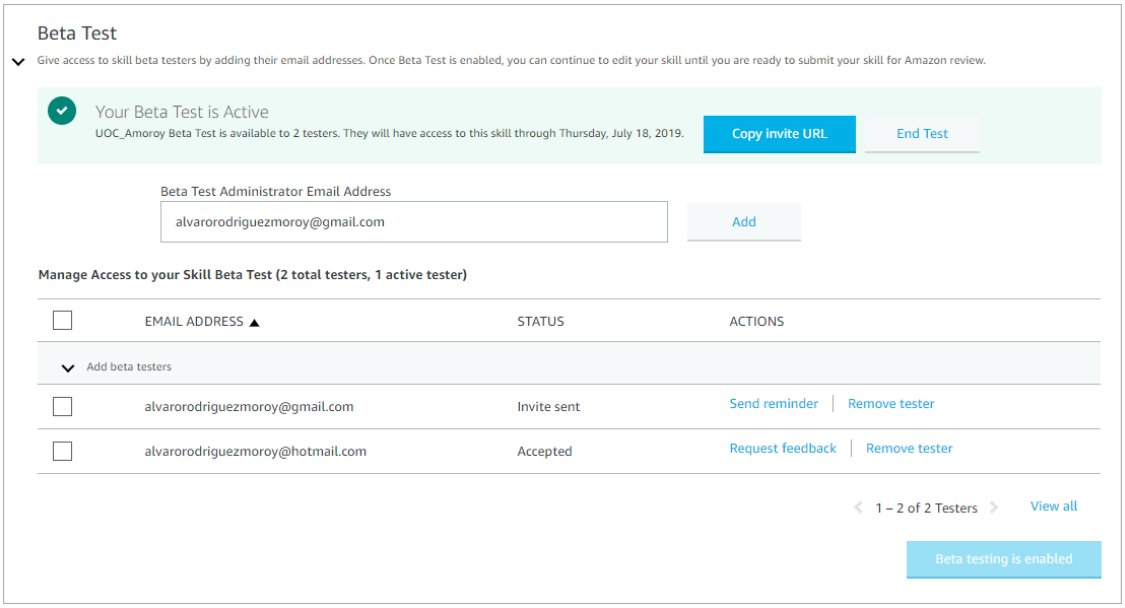


Ilustración 94 - Consola Alexa - BetaTest

Ahora ya se podrá utilizar el dispositivo Echo para probar la Skill en desarrollo.

11. Proyecto java en Eclipse

El código fuente de la aplicación se puede encontrar en:

https://github.com/amoroy/TFG-Alexa

En este apartado se pretende mostrar cómo está estructurado el proyecto de backend en Eclipse, qué buenas prácticas se han seguido y por qué.

11.1.1. Estructura del proyecto

Aquí se muestra el proyecto visto desde el IDE Eclipse:

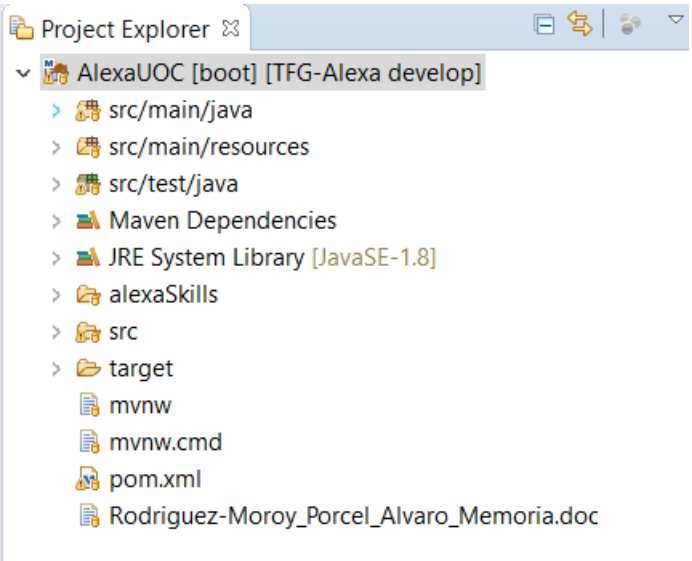
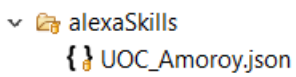


Ilustración 95 - Eclipse - Project

El primer nivel del proyecto se compone de la estructura típica de un proyecto maven donde se separa el proyecto de la siguiente manera:

- **src/main/java**: es donde se encuentra el código de la aplicación
- **src/main/resources**: es donde se encuentran todos los ficheros de configuración de la aplicación
- **src/test/java**: es donde se encuentran todos los test que prueban el código de la aplicación.
- **src/test/resources**: es donde se encuentran los ficheros de configuración para las pruebas. Esta carpeta no está ya que no se ha necesitado ningún fichero de configuración gracias a la potencia de JUnit y las anotaciones.
- **Maven Dependecies**: son las librerías externas que se necesitan para el proyecto.
- **JRE System Library**: son las librerías propias de la máquina virtual de java.
- **alexaSkills**: es donde se va a alojar el archivo json con toda la configuración de la consola de Alexa. Es un backup de la consola.



- **src**: es la carpeta raíz del código y los ficheros de configuración. Para proyectos web, tendría toda la parte de las vistas además del código fuente y los test.
- **target**: es donde se construye la aplicación. Tendrá el código compilado (.class) así como el jar de la aplicación con todas las librerías necesarias.
- **Ficheros**: los cuatro ficheros que hay en este nivel son dos ficheros propios de maven para poder construir la aplicación, el pom.xml que es el fichero de configuración de maven y que define el proyecto (como el nombre, el tipo de artefacto, las dependencias,…). Y por último, se ha colgado esta memoria para aprovechar el control de versiones del proyecto para tener esta memoria también versionada.

Si se baja al siguiente nivel en cada carpeta ya se entra en la estructura que se ha definido para el proyecto:

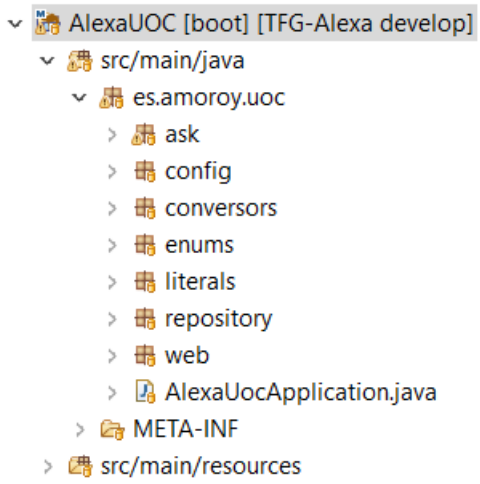
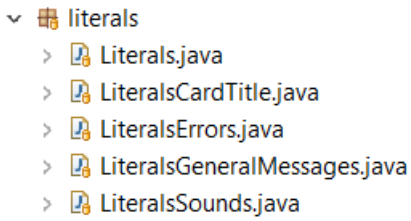


Ilustración 96 - Eclipse - Project 2

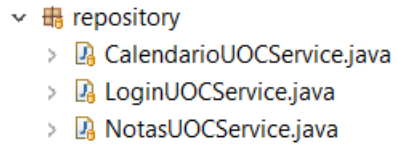
La manera en la que se ha querido separar el código del proyecto es la siguiente:

- **ask**: son las siglas de Alexa Skill Kit, y es donde se aloja todo el código que atiende a la IA de Alexa, así como los intents (funcionalidades) que se explican en el siguiente nivel.
- **config**: contiene una clase donde se asignan configuraciones de Alexa, como el skillId o los verificadores (que verificaciones que hace Alexa por defecto a modo de seguridad, como que coincida el skillId, que el timestamp sea razonable desde que se pidió hasta que se recibió,…).
- **conversors**: contiene una clase que se encarga de convertir los textos que se devuelven a Alexa (con tags xml para hacer pausas, poner énfasis,…) y les quita los tags para mostrar por pantalla esos textos. Esto se utiliza para los dispositivos Echo que cuentan con una pantalla y además de decirte la frase Alexa la muestran por pantalla.

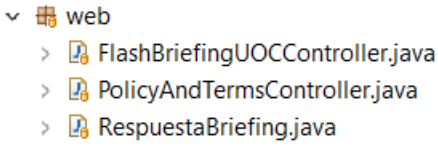
- **enums**: Aquí se encuentran los enumerados de la aplicación. En este caso se ha necesitado un enumerado para poder devolver el nombre al hacer el login. Si se pasa al servicio amoroy, que devuelva en la frase de bienvenida el nombre de Alvaro.
- **literals**: esta carpeta tiene cada uno de los literales que se devuelven a Alexa, separados por tipos de literales. En realidad tienen el atributo que utilizará la aplicación, pero el contenido estará ubicado en un fichero de configuración que se verá más adelante.



- **respository**: esta carpeta es la que contiene los servicios encargados de conectarse a los servicios de la UOC (en este caso simulados). La capa de repository representa la capa de acceso a base de datos. En este caso se ha querido dejar claro al introducir esta capa, que la conexión a los servicios de la UOC representan los datos de esta aplicación. En otras palabras, la propia UOC será la base de datos.



- **web**: la carpeta web representa el punto de entrada de algunas funcionalidades necesarias, como son el Flash Briefing y los controladores de la política y los términos de uso. El del Flash Briefing es porque este tipo de Skills necesitan un endpoint propio, al que Alexa llamará y devolverá directamente la noticia. Y sobre la política y los términos de uso fueron necesarios para poder crear una cuenta de beta test.



- **AlexaUocApplication.java**: es la clase de Spring que se encarga de levantar la aplicación cuando se despliega en un servidor. Todo queda delegado a las librerías de Spring gracias a las anotaciones y a Spring Cloud.

Una vez explicados los niveles, ahora se va a explicar la el nivel “ask”. Este nivel es el que contiene todas las funcionalidades de las Custom Skills, así como la configuración como punto de entrada de Alexa.

- ask
  - intents
  - AlexaConfig.java
  - AmoroySkillServlet.java
  - SkillRequestSignatureVerifierNulo.java
  - SkillRequestTimestampVerifierNulo.java
  - UOCAlexaSkillServlet.java

Estas clases, como ya se han ido explicando a lo largo de la memoria, son las encargadas de configurar el punto de entrada para la Custom Skill de Alexa, así como los diferentes Intents que escuchará la aplicación.

Además hay dos clases necesarias para evitar que la configuración por defecto de Alexa realice unas verificaciones de la firma del servicio y del timestamp. Así se podrá realizar las pruebas sin necesitar dichas verificaciones.

En la carpeta “intents” es donde se alojarán cada una de las funcionalidades que deben atender a Alexa, y que se han configurado en Alexa. Además de los handler por defecto que trae Alexa y que se han personalizado.

- ask
  - intents
    - CalendarioIntent.java
    - CancelandStopIntentHandler.java
    - HelpIntentHandler.java
    - LaunchRequestHandler.java
    - LoginIntent.java
    - NoIntentHandler.java
    - NotasIntent.java
    - SessionEndedRequestHandler.java

En el paquete src/main/resources se encuentran dos ficheros con la configuración de la aplicación (boostrap.yml) y con todos los properties para los literales (application.yml).

- src/main/resources
  - application.yml
  - bootstrap.yml

El paquete src/test/java es donde se encuentran todas las pruebas del proyecto. La estructura de los test es seguir la misma estructura que el código fuente.

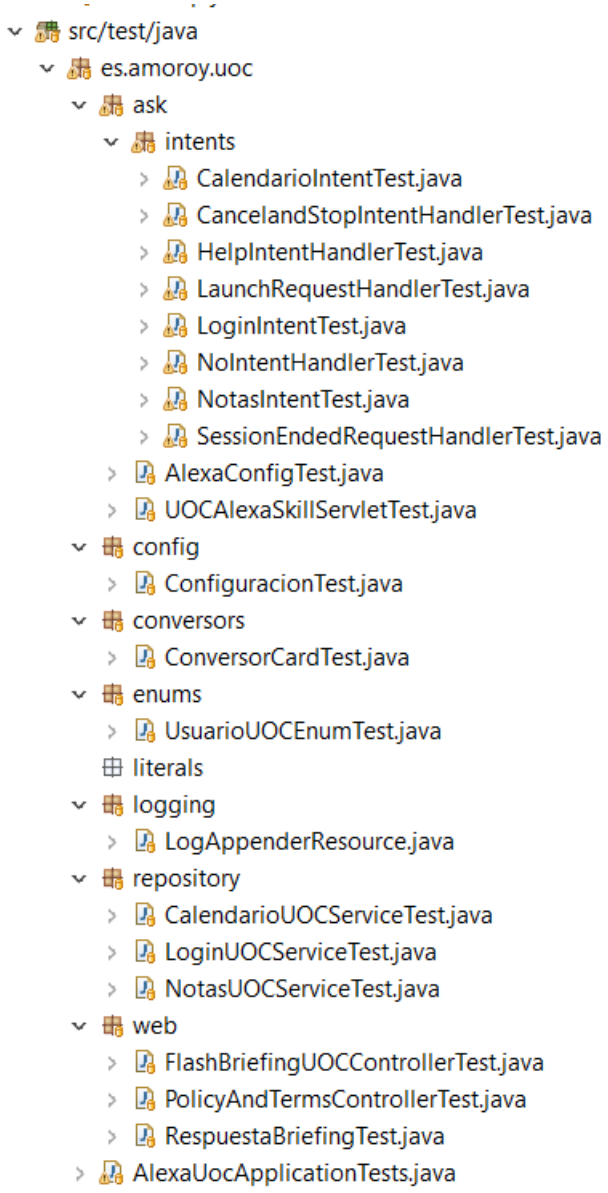


Ilustración 97 - Eclipse - Project 3

Y por último la carpeta target, que es donde se generan todas las clases compiladas y el fichero jar con la aplicación que se desplegará en Pivotal:

Por lo tanto, como resumen, se ha seguido la estructuración que definen los proyectos maven para la estructura básica, y luego se ha separado el contenido del proyecto según el ámbito y la responsabilidad.

Por muy pequeña que quedara una clase, se ha mantenido separadas las responsabilidades, siguiendo el principio de responsabilidad única de los principios SOLID (Glosario).

11.1.2. Buenas prácticas

Para este proyecto se han seguido una serie de máximas en las que se intenta conseguir un código de calidad y una cobertura de pruebas superior al 75%.

Como se ha dicho desde el principio, la metodología principal –o más bien práctica de desarrollo– ha sido el TDD para conseguir una buena cobertura de pruebas y ceñirse a lo únicamente necesario (lo que la prueba exige).

Para verificar la cobertura de código se ha aprovechado las herramientas integradas en el IDE Eclipse para el análisis de código, en este caso, jacoco (plugin para analizar la cobertura de código).

Así, ejecutando la herramienta como se muestra a continuación, se verá cual es la cobertura total del proyecto, así como la cobertura de cada clase:

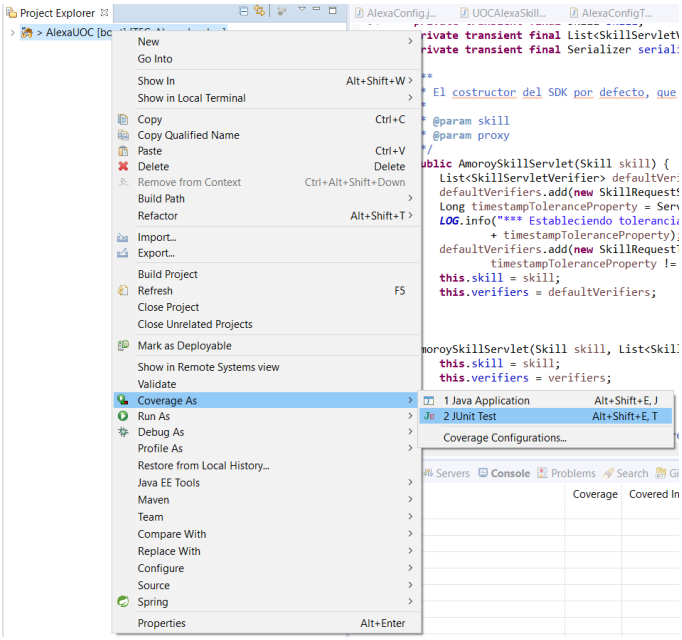


Ilustración 98 - Eclipse - Coverage

Properties | Servers | **Console** | Problems | Search | Git Repositories | Terminal | Coverage

AlexaUOC (08-jun-2019 18:55:27)

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
AlexaUOC	89,9 %	3.341	376	3.717
src/main/java	79,4 %	1.438	372	1.810
es.amoroy.uoc.literals	0,0 %	0	217	217
es.amoroy.uoc.ask	68,5 %	289	133	422
es.amoroy.uoc.repository	94,8 %	221	12	233
es.amoroy.uoc	0,0 %	0	8	8
es.amoroy.uoc.enums	98,1 %	105	2	107
es.amoroy.uoc.ask.intents	100,0 %	668	0	668
es.amoroy.uoc.config	100,0 %	48	0	48
es.amoroy.uoc.conversors	100,0 %	8	0	8
es.amoroy.uoc.web	100,0 %	99	0	99
src/test/java	99,8 %	1.903	4	1.907

Ilustración 99 - Eclipse - Coverage 2

Si se quiere ver que se está analizando, el propio plugin ofrece una visión rápida por colores de cada línea de código. Dando una información completa de la cobertura.

Así, si el color es verde, significa que esa línea está cubierta, si es roja, significa que no, y si es amarilla significa que tiene condiciones y no se ha realizado una prueba de cada una de esas condiciones:

```
58    /**
59     * El costructor del SDK por defecto, que establece no proxy
70     *
71     * @param skill objeto de alexa con la funcionalidad que necesita.
72     */
73    public AmoroySkillServlet(Skill skill) {
74        List<SkillServletVerifier> defaultVerifiers = new ArrayList<>();
75        defaultVerifiers.add(new SkillRequestSignatureVerifier(Proxy.NO_PROXY));
76        Long timestampToleranceProperty = ServletUtils.getSystemPropertyAsLong(TIMESTAMP_TOLERANCE_SYSTEM_PROPERTY);
77        LOG.info("*** Estableciendo tolerancia desviacion timestamp (por defecto 30 segs): "
78                + timestampToleranceProperty);
79        defaultVerifiers.add(new SkillRequestTimestampVerifier(
80                timestampToleranceProperty != null ? timestampToleranceProperty : DEFAULT_TOLERANCE_MILLIS));
81        this.skill = skill;
82        this.verifiers = defaultVerifiers;
83
84    }
85
86    AmoroySkillServlet(Skill skill, List<SkillServletVerifier> verifiers) {
87        this.skill = skill;
88        this.verifiers = verifiers;
89    }
```

Ilustración 100 - Eclipse - Coverage 3

Otra buena práctica que se ha seguido es añadir javadoc (documentación en formato java acerca del método o clase que se está codificando) para que cuando se vaya a usar el método o clase se exponga dicha documentación al desarrollador:

```
speechText = servicio.getNotas(asignaturaUOC);
LOG.info("-------------
return input.getResponseB
        .withSimpleCard(l
        .withShouldEndSes
```

getNotas(String asignatura) : String - NotasUOCService

Metodo para pedir las notas (simulado)
Parameters:
asignatura es el nombre de la asignatura de la UOC
Returns:
el mensaje con las notas de la asignatura o de todas las asignaturas en caso de no pasar ninguna asignatura

Ilustración 101 - Eclipse - JavaDoc

También se ha intentado seguir una práctica (que cada desarrollador tiene su opinión) sobre no comentar el código que se realiza, ya que el propio código debería ser auto explicativo gracias a los nombres de las variables o métodos.

Para la exposición de la memoria se han seguido buenas prácticas aprendidas en la asignatura 'Competencia comunicativa para profesionales de las TIC' como por ejemplo que el tamaño de los párrafos no varie entre unos y otros.

También el uso de diferentes signos de puntuación, así como intentar no repetir, en la medida que ha sido posible, palabras y expresiones como por ejemplo "Como se puede observar", "Como se puede apreciar", "Como se puede ver".

12. Coste del proyecto

Para la consecución de este proyecto en un caso real, se necesita contar con una serie de recursos (personas) y servicios que permitan obtener un producto de calidad.

Por un lado se necesitarán recursos especializados, ya que es un proyecto con necesidades muy concretas. Por otro lado; se necesitarán programadores java (se podría hacer en cualquier lenguaje) que son bastante más demandados.

Los especialistas serán un arquitecto de software capaz de entender las necesidades del proyecto y el posible crecimiento. Así tendrá en cuenta la necesidad tanto de escalabilidad del software como de su mantenimiento.

El siguiente especialista es un experto en la parte de Alexa. Sería el equivalente al rol de UX (experiencia del usuario). Este rol sería el encargado de crear todas las utterances (frases) que podrá decir el usuario así como la manera en la que deberá comportarse Alexa.

Este rol es muy difícil de encontrar puesto que este tipo de tecnologías son muy recientes y no hay mucha oferta para estos puestos –y menos aún en España, que es más reciente todavía esta tecnología de inteligencias artificiales–.

Luego se necesitará un par de desarrolladores para crear toda la parte de backend y de comunicación con los servicios de la UOC. En este proyecto se podría pedir un "senior" (5 años o más en Java) y un "junior".

El tiempo estimado del proyecto será de 3 meses a tiempo completo del arquitecto y los programadores, y de mes y medio del experto en Alexa. Cada mes se estima que serán 22 días.

Los salarios se han sacado de las páginas [5] y [2], en donde la primera expone el salario medio de los roles más conocidos, y el segundo es

una web con ofertas para desarrolladores de Alexa (para este último se ha cogido el salario más bajo del rango de la oferta).

ROL	DÍAS	COSTE / AÑO	COSTE TOTAL
Arquitecto	66	47.854,36 € [5]	11.963,59 €
Programador Senior	66	39.806,29 € [5]	9.951,57 €
Programador Junior	66	28.405,24 € [5]	7.101,31 €
Experto Alexa	33	89.000,00 € [2]	11.125,00 €

Por lo que el coste de los recursos humanos para este proyecto sería de: **40.141,47 €**

13. Arquitectura de referencia

Si este proyecto pasara a hacerse real, debería llevar una arquitectura algo diferente, ya que la que se ha hecho en ese proyecto es un servicio rest publicado en un servidor cloud.

Esta arquitectura no permite la escalabilidad necesaria (aun publicándolo en un servidor cloud), ni permite realizar cambios de configuración en “caliente” sin necesidad de volver a desplegar la aplicación, entre otras cosas.

Por tanto, se va a proponer una arquitectura de referencia para poder sacarle el máximo provecho y poder tener todas las ventajas (e inconvenientes) que ofrece la arquitectura basada en microservicios.

La arquitectura de microservicios se basa en pequeños servicios completamente independientes unos de otros (cada uno tendrá su propia base de datos, su propia responsabilidad, configuración…).

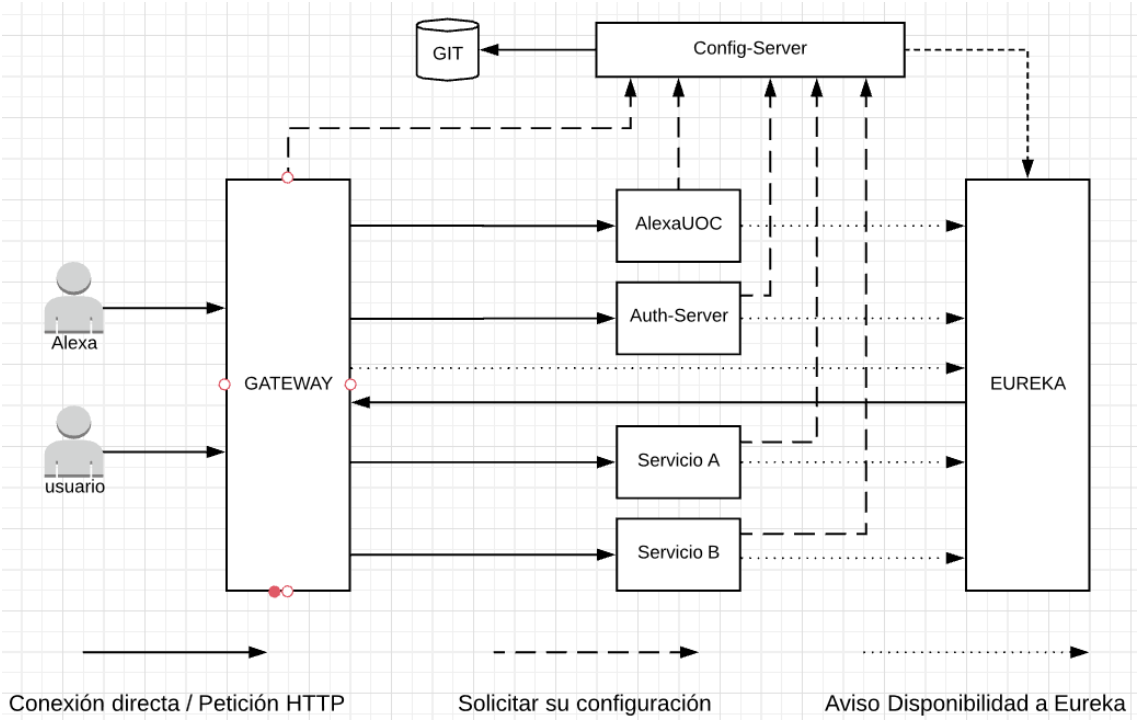


Ilustración 102 - Arquitectura de referencia (Microservicios) [4]

Esta arquitectura concretamente tendrá un microservicio encargado de "saber" que microservicios están levantados y donde, así como cuáles son las instancias de cada uno de ellos.

A este tipo de microservicios se les conoce como descubridores. Y Spring Cloud Netflix ya han implementado una solución ampliamente probada y utilizada, llamada Eureka.

El Eureka será el que facilite la tarea de balanceo que se hará a cada microservicio, así como mejorar la tolerancia a fallos como se verá con los microservicios Gateway e Hystrix (explicados a continuación).

Otro microservicio fundamental para conseguir que se puedan realizar cambios de configuración en caliente es el microservicio de configuración y el repositorio donde se alojen los ficheros.

Este microservicio es el que irá a dicho repositorio (que puede ser Git, SVN,…) y se traerá todos los ficheros de configuración de cada microservicio y cada microservicio pedirá a éste su configuración.

Por ejemplo, para el caso de este proyecto se tendrían en un fichero de configuración todas las url de los servicios de la UOC, de tal manera que si cambia alguna se podría cambiar en caliente.

El siguiente microservicio es uno de los más importantes ya que va a ser el punto de entrada de todas las peticiones de Alexa. La implementación de Spring Cloud se conoce como Gateway.

El Gateway será el encargado de recibir todas las peticiones y decidir a que microservicio debe ir. También utilizará un balanceador que permitirá una mejor escalabilidad y rendimiento.

En este microservicio se configurarán todas las normas de acceso como por ejemplo que no haya más de 10 peticiones por segundo de una misma máquina o que la petición lleve una cabecera concreta.

El Gateway es una herramienta muy potente de seguridad y permite todo tipo de configuraciones y filtros. Así que será este el punto clave para los accesos al resto de microservicios.

Una vez se tienen estos tres microservicios ya se puede empezar con los servicios del propio proyecto y con las funcionalidades que se deseen.

Por ejemplo, se añadirá el microservicio que se ha realizado en este proyecto al Eureka para que se exponga como recurso y que el Gateway permita llegar hasta él.

Esto significaría que en Alexa, en donde se define el punto de entrada de los backend (endpoint) se pondrá el Gateway y el recurso al que se quiere ir.

Esta arquitectura de microservicios ofrece diferentes ventajas que se agregan o se quitan añadiendo microservicios que ofrecen unas u otras ventajas según las necesidades.

Por ejemplo, se puede añadir un microservicio encargado de la monitorización del resto de microservicios, o uno encargado de la autenticación del usuario.

Y siempre ofreciendo las ventajas de la escalabilidad de cada uno de los servicios y de una tolerancia a fallos adecuada gracias a otros microservicios como Hystrix.

Hystrix, implementa el patrón Circuit Breaker que, muy a alto nivel, consiste en abrir el circuito cuando se encuentra un problema constante y que lo cierra cuando vuelve a funcionar correctamente.

Como curiosidad, es importante señalar que muchos de estos microservicios son creaciones de Netflix. La misma plataforma para ver películas y series.

Netflix es una de las empresas más importantes en la comunidad de desarrolladores. Y son los responsables de uno de los códigos abiertos más utilizados en este tipo de arquitectura.

14. Conclusiones

Como en todo proyecto, se han mejorado, incluso se han adquirido, algunas habilidades propias de hacer proyectos prácticos. Y más aún al hacerlos solos y tener que enfrentarse a los problemas técnicos sin ayuda.

Entre otras, en este proyecto se ha aprendido que una buena planificación, sobre todo una planificación realista, es clave para poder realizar un proyecto de calidad con éxito.

Por otro lado, una metodología adecuada permite cumplir los hitos marcados y ver la evolución del proyecto gracias a un panel que muestra las tareas finalizadas, así como las que quedan por hacer.

En este caso, la utilización de una metodología iterativa ha facilitado el seguimiento y el cumplimiento de los hitos marcados. También, gracias al consultor, se han detectado algunas problemáticas en las que no se había pensado, como por ejemplo el login.

Gracias a esta metodología ágil se ha podido absorber esa nueva problemática sin provocar un retraso en el proyecto. En un proyecto con metodología en cascada seguramente hubiera resultado en un retraso directo.

También se han aplicado los conocimientos que se han ido aprendiendo durante la carrera a la hora de redactar el documento y realizar el proyecto para maximizar la calidad del mismo.

El resultado final es un proyecto funcional y de calidad y una memoria que describe el trabajo y demuestra los conocimientos adquiridos en este proyecto.

Como objetivo, se espera que este pequeño proyecto sirva a la UOC para mostrar la nueva necesidad de evolución y para abrir un nuevo camino para mantenerse en la cumbre como número uno de las universidades online de España.

15. Glosario

Custom Skill

Skill de Alexa destinada a interactuar con el usuario. Muestra la información que el usuario pide a Alexa., 9

CustomSkillBuilder

Clase del sdk de Alexa para construir todos los Intents que compondrán la aplicación, 50

Echo

Dispositivo físico de la marca Amazon con el que el usuario puede interactuar con la inteligencia artificial Alexa., 9, 54, 55, 69, 71, 72

Flash Briefing Skill

Skill de Alexa orientado a ofrecer información general cuando se invoca un evento de alexa de dar las noticias generales. Es la skill que actúa cuando se dice ‘Alexa, dame las noticias del día’, 9

handlers

Son clases encargadas de gestionar un comportamiento por defecto de Alexa. Suelen implementarse 4 (LaunchRequestHandler, CancelandStopIntentHandler, HelpIntentHandler, SessionEndedRequestHandler), 50

Intent

Funcionalidad concreta de una skill. Cada Intent debe ofrecer una funcionalidad distinta como por ejemplo las notas de una asignatura o las fechas de entrega de las próximas actividades, 9

javadoc

Documentación en formato java acerca del método o clase que se está codificando para que cuando se vaya a usar dicho método o clase se exponga dicha documentación al desarrollador y tenga la información suficiente para saber que hace ese método o clase, que son sus parámetros y que devuelve., 79

kanban

Metodología de trabajo que basa su filosofía no tener muchas tareas a la vez para centrarse en terminar las que están e curso, *10*

mockear

Verbo creado en el ámbito de desarrollo profesional. El significado es imitar el comportamiento del objeto que se imita. Sirve para no depender de ese objeto y tener una respuesta controlada. Suele usarse, a través de librerías como Mockito, en pruebas unitarias y de integración., 48

principio de responsabilidad única

Es el primero de los principios SOLID. Unos principios que debe seguir todo desarrollador para conseguir código de calidad.Est principio aboga por crear clases con una única funcionalidad o objetivo, es decir, que no se encargue de hacer muchas cosas, sino una sola., 51

Scrum

Metodología (o más biuen marco de trabajo) ágil orientada a equipos de trabajo auto-organizados y con entregas incrementales al cliente., 11

Skill

Aplicación de Alexa. Esta aplicación contendrá las funcionalidades que se le ofrecerá al usuario, 9, 32

TDD

Test-driven development. Esta estrategia requiere que primero se cree una prueba unitaria que pida el requisito en cuestión y después se refactorice el código para que se ponga en verde dicha prueba. Se puede saber más sobre esta práctica en el enlace dentro de la bibliografía de este trabajo, 35

16. Bibliografía

[1]
«Alexa Skills Kit Sound Library | Alexa Skills Kit». [En línea]. Disponible en: https://developer.amazon.com/es/docs/custom-skills/ask-soundlibrary.html. [Accedido: 08-may-2019].

[2]
«Amazon Alexa developer Jobs», *Glassdoor*. [En línea]. Disponible en: https://www.glassdoor.com/Jobs/Amazon-alexa-developer-Jobs-EI_IE6036.0,6_KO7,22.htm. [Accedido: 08-jun-2019].

[3]
«Desarrollo guiado por pruebas - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas. [Accedido: 22-mar-2019].

[4]
«Documentos». [En línea]. Disponible en: https://www.lucidchart.com/documents#docs?folder_id=home&browser=icon&sort=saved-desc. [Accedido: 15-mar-2019].

[5]
«Estos son los profesionales informáticos más demandados y sus salarios», *Channel Partner*, 17-ene-2019. [En línea]. Disponible en: http://www.channelpartner.es/negocios/noticias/1109660002202/estos-profesionales-informaticos-mas-demandados-y-salarios.1.html. [Accedido: 08-jun-2019].

[6]
«Imagen gratis en Pixabay - Letras Positivas, Scrabble, Texto». [En línea]. Disponible en: https://pixabay.com/es/photos/letras-positivas-scrabble-texto-2355685/. [Accedido: 25-abr-2019].

[7]
«Speech Synthesis Markup Language (SSML) Reference | Alexa Skills Kit». [En línea]. Disponible en: https://developer.amazon.com/es/docs/custom-skills/speech-synthesis-markup-language-ssml-reference.html. [Accedido: 06-may-2019].

[8]
«Tom's Planner - Gantt charts for the rest of us.», *TomsPlanner*. [En línea]. Disponible en: https://www.tomsplanner.com/. [Accedido: 22-feb-2019].

17. Anexos

17.1. Creación de una cuenta de estudiante de Github SDP

17.2. Pivotal

17.3. Instalación de Lombok