ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Tabajo Fin de Estudios

UPCTbot: Desarrollo de un bot aplicado a la docencia

AUTOR: Pablo Moreno Box
DIRECTOR: Mathieu Kessler
CODIRECTOR: Daniel Pérez Berenguer

Septiembre / 2018

Tabla de contenido

0. Introducción

Con la invención de los bots se ha conseguido la simplificación de tareas que para el ser humano pueden ser repetitivas y tediosas. Con ellos el número de acciones a realizar para conseguir algo es mucho menor y cualquier acción podría ser casi inmediata. Todo esto intentando simular el comportamiento humano. Se puede manejar remotamente otros dispositivos, crear recordatorios, hacer pedidos sin necesidad de realizar los pasos de registro u organizar un viaje en cuestión de pocos minutos con bots especializados en ello.

Además, es posible dotar a los bots de inteligencia artificial. Dependiendo de nuestros gustos y actividades cotidianas este nos puede ofrecer las diferentes opciones que más convengan a los diferentes usuarios.

El uso de bots está cada vez más extendido gracias al desarrollo de nuevo hardware con mayor capacidad de cómputo y auge de aplicaciones de mensajería instantánea. Usamos bots a diario y en ocasiones no somos conscientes de ello. Cortana, Siri o Google Now son bots de uso cotidiano que forman parte de nuestro día a día.

También se pueden adquirir dispositivos de voz inteligentes para su uso en el hogar. Dos de los dispositivos más famosos en este ámbito son Google Home Mini y Amazon Echo bot. Estos dispositivos son capaces de controlar otros dispositivos inteligentes, como una televisión.

En un corto periodo de tiempo, ha aumentado el uso de aplicaciones de mensajería instantánea. Según un estudio de Statista, WhatsApp posee 1.500 millones de usuarios activos, y otras aplicaciones como Telegram y Skype rondan los 300 millones de usuarios. Existen otras aplicaciones que potencian el trabajo en grupo y que admiten el uso de bots como Microsoft Teams o Slack.

Los bots no solo están centrados en la búsqueda de información, compras o asistentes personales. Es posible crear bots especializados en la educación, que ayuden al alumno en su formación o al docente en su labor. Algunos ejemplos de bots para la docencia son LOLA y EduarBot. Estos son bots tipo FAQ que

responden a los usuarios sobre preguntas relacionadas con fechas de matriculación, notas, asignaturas u horarios.

LOLA es un bot diseñado por 1MillionBot que está enlazado con diferentes sistemas de la Universidad de Murcia (gestión académica, cita previa, gestión de centros, etc.). Los usuarios de este bot pueden preguntar sobre notas de corte de la EBAU, fechas de matriculación o pedir cita entre otras.

EduarBot ha sido creado por Encamina, diseñado para convertirse en asistente. Gracias a la conexión con los servicios de los centros, los alumnos podrán recibir información sobre horarios, tutorías o servicios universitarios. EduarBot posee un entorno multicanal, pudiendo comunicarse a través de diferentes aplicaciones.

Este trabajo se centra en demostrar el potencial de los bots en la educación. Para ello, se creará un bot que estará conectado con una plataforma e-learning, y capaz de comunicarse con aplicaciones externas para el envío de notificaciones.

0.1. Objetivos

Los objetivos de este trabajo son:

- Saber qué es un bot y aplicar los diferentes principios de diseño para conseguir que sea practico.
- Conocer la tecnología que nos ofrece Microsoft para la creación de bots.
- Aprender conceptos básicos de desarrollo de bots en el lenguaje C#.
- Desarrollar un bot funcional con aplicaciones orientadas a la docencia.

0.2. Estructura del Trabajo Fin de Estudios

El Trabajo Fin de Estudios se estructura en dos bloques.

En la primera parte, se aborda la creación de un bot, dividido en cuatro apartados. Principios de diseño para la creación de un bot con las mejores prácticas, la parte de gestión de un bot que nos ofrece Microsoft, el desarrollo del mismo mediante el lenguaje C# y, por último, algunas herramientas que existen para la creación de estos.

En la segunda parte, se abordará a la creación de un bot utilizando lo aprendido en la primera parte. Pasando por las diferentes fases, de toma de decisiones en un entorno real con dificultades que abordar, diseño conforme a las decisiones tomadas, desarrollo y testeo.

Por último, se formularán unas conclusiones y unas líneas futuras que se podrán seguir para mejorar la comprensión, y la inteligencia del bot creado, además de unas posibles modificaciones en unos casos determinados.

1. ¿Qué es un bot?

Un bot es una aplicación con la que los usuarios interactúan con el fin de obtener determinada información o realizar acciones de forma más natural sin la necesidad de realizar una serie de pasos. Existen diferentes formas de comunicación entre un usuario y un bot, ya sea a través de texto, voz o imágenes. Esta comunicación puede ser más o menos sofisticada dependiendo de la elección del desarrollador y la finalidad del bot, pudiendo ser conversaciones complejas o simplemente acción-respuesta.

Un bot puede estar diseñado en cualquier lenguaje de programación, funcionar en un servidor o un cliente, ser un agente móvil, etc. Lo más común es que sean especialistas en cumplir una función específica.

La programación de un bot puede estar diseñada para cumplir tareas básicas como puede ser el recordatorio de alguna tarea o para automatizar algún proceso. Existen bots más complejos que buscan realizar actividades con toma de decisiones. Estas decisiones son tomadas a partir de parámetros que se incluyen en el código de programación.

Los usuarios necesitan un medio de comunicación para interactuar con bots. Estos medios pueden ser aplicaciones móviles o de escritorio, aplicaciones de mensajería instantánea, dispositivos hardware o incrustaciones en web. A estos medios de comunicación los llamaremos **canales**.

2. Creación de un bot

Existen diferentes lenguajes para la creación de bots al igual que hay diferentes plataformas con las que el bot puede comunicarse con los usuarios. Indiferentemente a esto, existen unas pautas que el desarrollador debe seguir para la correcta creación de un bot, estas son: la **planificación**, el **desarrollo** de código, hacer **pruebas** en busca de fallos, la **publicación** en un servicio web, **conexión** con los canales donde va a estar operativo y por último la **evaluación**

para conocer el rendimiento y las posibles mejoras volviendo así al paso del desarrollo para aplicar cambios.

A continuación, se explican estas pautas:

a. Planificación

Se deben tener en cuenta cuáles van a ser las necesidades que el bot debe cumplir, además de concretar la manera en la que el bot se comunicará con los usuarios teniendo en cuenta el conseguir una buena experiencia de usuario. Después de tomar estas primeras decisiones se debe elegir el lenguaje de desarrollo y la plataforma que más se adecuen a las características del bot.

b. Desarrollo

En este paso se debe construir con el lenguaje que se haya elegido las diferentes funcionalidades que se desean implementar.

c. Pruebas

Una vez que se tiene un primer desarrollo terminado, es necesario realizar las pruebas para verificar si las acciones que realiza el bot son las deseadas por el desarrollador y que no se producen errores. Es conveniente realizar las pruebas en entornos cerrados sin que el bot esté público, también en algunos entornos existen diferentes herramientas que ayudan al desarrollador a depurar su código. Si se encuentran errores se debe volver al paso “b” (Desarrollo) para realizar los cambios pertinentes.

d. Publicación

Una vez se tiene el desarrollo completo y sin errores, el siguiente paso es publicar la aplicación en un servicio web, centro de datos o en la nube.

e. Conexión

Cuando el bot ya está listo para su uso, es necesario conectarlo a los diferentes canales. Estos son aplicaciones de mensajería (Telegram, Skype, Teams, etc.), webs u otras aplicaciones (Cortana, Google, etc.).

f. Evaluación

Por último, y quizás la parte que más tiempo ocupa en el desarrollo de un bot es el mantenimiento y evaluación de este. Se debe analizar su rendimiento y entorno de trabajo para mejorar y garantizar la mejor experiencia de usuario posible y volver al paso “b” (Desarrollo) para subsanar los diferentes puntos en los que se pueda mejorar.

3. Principios de diseño

A la hora de diseñar un bot se deben tener en cuenta unos factores para que este ofrezca la mejor experiencia de usuario posible y garantizar su éxito. Estos factores son:

- **Eficacia:** el bot realiza la acción correctamente.
- **Eficiencia:** el bot realiza la acción requerida en el menor número de pasos posibles que cualquier alternativa.
- **Sencillez:** el bot es intuitivo y los usuarios saben cómo usarlo.
- **Alcance:** se desea que el bot llegue al mayor número de usuarios posible.

Otros factores como la inteligencia, la comprensión de lenguaje o si es capaz de comunicarse mediante la voz no garantizan el éxito. Al usuario solo le preocupa si el bot soluciona sus necesidades. Para conseguir el éxito de un bot, no es necesario que este posea muchas funcionalidades o dé información que no sirve para nada.

3.1. Diseño de interacciones

Es importante que las interacciones con el usuario sean lo más claras posibles y que no se dé la opción de respuestas abiertas. Existe la opción de dejar que el usuario comience la conversación pidiendo lo que necesita o que sea el bot el que pregunte al usuario dando las posibles opciones que el este podría abordar.

Hay que conseguir que el usuario no pierda el interés por el bot. Proporcionar la información necesaria y no realizar preguntas con respuestas ambiguas garantiza la atención. Así se evita que los usuarios tengan dudas y dejen de usar el bot.

3.2. Diseño de flujo de conversación

Los bots se componen de cuadros de diálogo. Permiten separar las funcionalidades dependiendo de la tarea a realizar. Los diálogos pueden procesar la respuesta del usuario e invocar otros diálogos.

Existe un diálogo principal que es el encargado de redirigir a diferentes diálogos secundarios dependiendo de las necesidades del usuario en cada momento. Los diálogos secundarios pueden invocar otros diálogos o volver el control al diálogo principal.

Los diálogos son muy prácticos para separar las diferentes acciones en los bots, pero los usuarios por norma general no siguen una estructura guiada a la hora de comunicarse y es posible que cambie de opinión en medio de un proceso y elija realizar otro. Se debe contar con esta posibilidad y el desarrollador deberá

decidir cómo debe actuar el bot en estos casos, teniendo la opción de insistir en terminar lo que estaba haciendo, reiniciar el diálogo o realizar la nueva acción que el usuario ha ordenado. No existe una opción óptima para este escenario, esto dependerá de la situación y cómo el desarrollador quiera abordarlo.

Para evitar que un usuario se pierda por los diálogos, se debe considerar la implementación de las opciones de **volver atrás**, **volver al diálogo principal** y **cancelar operaciones**.

3.3. Diseño de la experiencia de usuario

Para conseguir una buena experiencia de usuario es muy importante la comunicación con el bot. Al igual que las personas nos comunicamos de forma diferente, los bots también pueden, ya sea mediante controles enriquecidos, texto o voz. Se puede elegir un tipo de comunicación o usar todos en la misma implementación. Esta decisión dependerá de la finalidad del bot y la preferencia de los usuarios a la hora de interactuar con él.

Los **controles enriquecidos** pueden ser botones, imágenes, menús o tarjetas, todo esto dependerá del canal en el que se desarrolla el bot. Utilizar este modo de comunicación ayuda al usuario a la hora de tomar decisiones consiguiendo guiar la conversación.

El **texto** es la forma más sencilla de comunicación con un bot. Puede usarse para contestar preguntas con respuestas cortas, como por ejemplo “¿Cuántos años tienes?” ¿Quieres información sobre el tiempo? También es útil si se usan comandos, estos devuelven el resultado esperado, son fáciles de aprender y es fácil desarrollar la comprensión de comandos en un bot.

También es posible que nuestro bot tenga capacidades de comprensión de lenguaje pero para esto es necesario la utilización de aplicaciones externas y hace que la complejidad del bot aumente considerablemente.

Otra opción es la **voz**, es posible desarrollar bots con comprensión del habla y ofrecer respuestas con voz, estos bots son diseñados para aplicaciones donde no es posible utilizar otro método de entrada o por las características del bot.

4. Gestión

Existen diferentes herramientas para la gestión de bots, dependiendo de la plataforma y lenguaje utilizados. Podemos encontrar diferentes herramientas para análisis de estadísticas, pruebas, gestión de recursos o desarrollo.

Este estudio se centra en la creación de bots con las herramientas que nos ofrece Microsoft. Este apartado se centra en hacer una pequeña mención de algunas de estas herramientas. Primero se introducirá **Azure** (servicio en la nube) donde se implementa y administra el bot creado para este trabajo.

4.1. Portal Azure

Azure es una nube pública de pago. Permite implementar y administrar de forma rápida y cómoda todas las aplicaciones alojadas en la red global de centros de datos de Microsoft (nube). Estos centros de datos disponen de gran seguridad y protección de datos, además de respaldo por posibles errores consiguiendo así una disponibilidad del 100% para el usuario.

En Azure existen diferentes servicios de infraestructura y plataformas para el desarrollo de aplicaciones. Se dispone de servicios de infraestructura de almacenamiento (redes, máquinas virtuales, etc...). También servicios de plataformas como bases de datos de alta disponibilidad SQL, CMS para el desarrollo de web o backend para aplicaciones móviles.

Uno de los servicios que ofrece Azure es la creación y administración de bots. Podemos encontrar diferentes funcionalidades:

- **Pruebas:** Gracias al chat web integrado, se puede comprobar el funcionamiento del bot.
- **Análisis:** Es posible la recolección de datos para mostrar gráficas de nº de usuarios, tráfico, latencia, etc…
- **Comunicación:** Se puede configurar y administrar los canales disponibles para la comunicación entre bot y usuarios (Ver **Anexo)**. Para

la comunicación entre bot y canales existe **Bot Framework** que será explicado más adelante.

Para la implementación del bot en Azure existen tres opciones:

- **Edición en línea:** Es posible modificar el código de la aplicación sin necesidad de utilizar un entorno de desarrollo integrado.
- **Descarga de código:** Para trabajar con el código del bot en un entorno de desarrollo integrado y hacer pruebas en local si fuese necesario. Una vez realizado los cambios habría que volver a subir el código.
- **Implementación continua:** Con Visual Studio el desarrollador será capaz de trabajar localmente y una vez verificados los cambios, estos se aplicarán automáticamente en la nube.

En la **figura 1** se muestra el panel de gestión de bots en Azure, concretamente la función de conexión de canales.

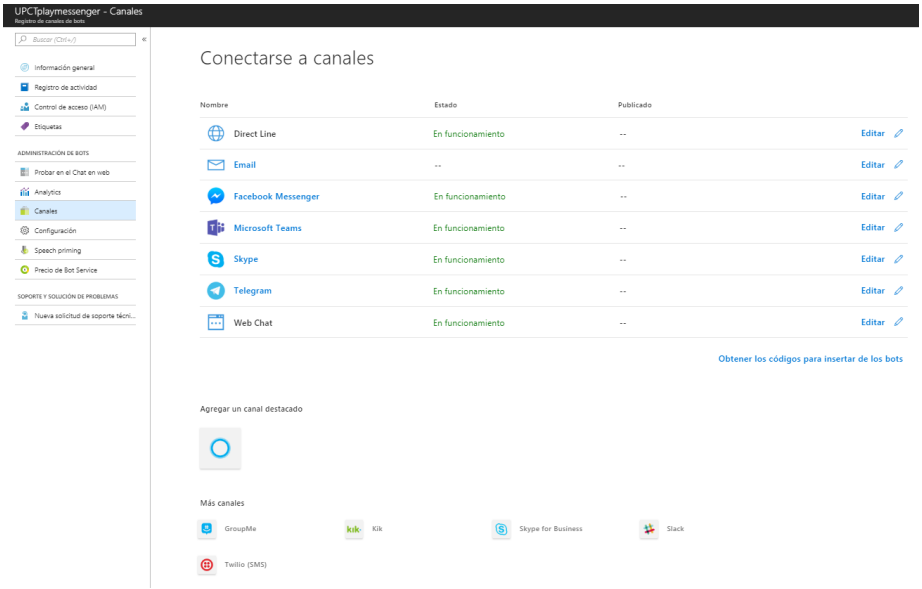


Figura 1: Panel de gestión de un bot.

4.2. Channel Inspector

Como ya se ha mencionado anteriormente, los bots pueden comunicarse mediante texto, botones, voz, y tarjetas enriquecidas. Sin embargo, una misma tarjeta se mostrará de manera diferente en Telegram, Skype o Facebook Messenger por ejemplo. Hay que estar seguro de cuál será el resultado final en cada uno de los canales utilizados para crear la mayor homogeneidad posible entre canales. Se debe tener en cuenta que existe la posibilidad de que los diferentes canales no sean compatibles con una característica en concreto.

Para que el desarrollador tenga una vista previa de cuál será la apariencia y disponibilidad de la característica utilizada, Microsoft cuenta con la herramienta web **Channel Inspector.** Desde la url (https://docs.botframework.com/en-us/channel-inspector/channels/Skype/) se puede visualizar el diseño de cada una de las características en los diferentes canales. Así, se podrá diseñar el bot de la manera más optimizada posible para los canales utilizados teniendo en cuenta la finalidad de este. En la siguiente **figura** vemos una captura de la interfaz web.

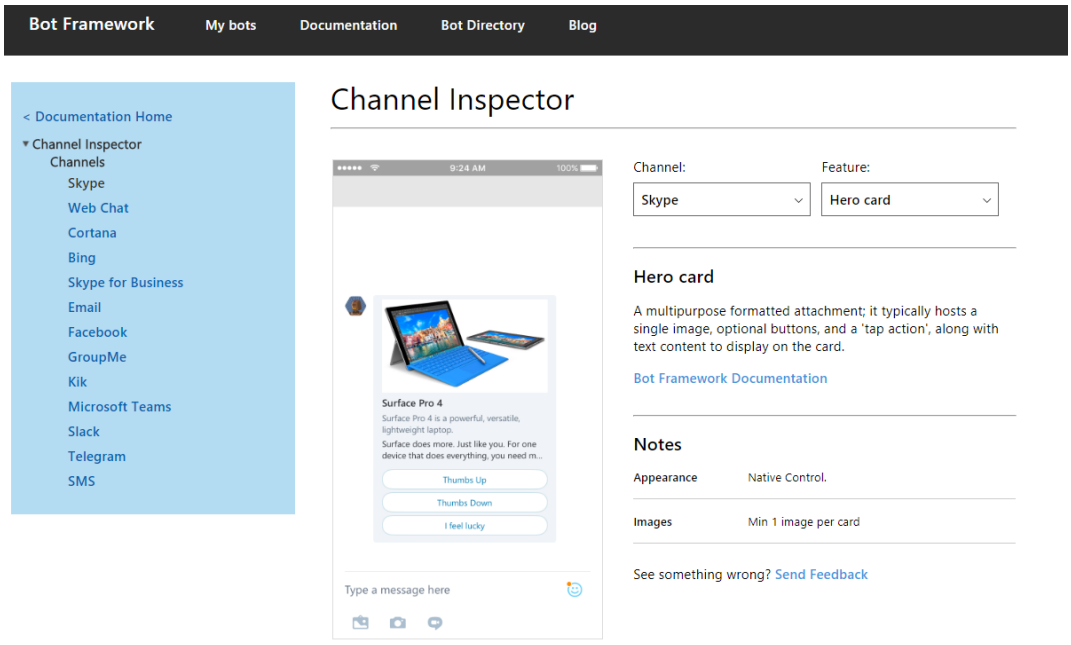


Figura 2: Channel Inspector.

5. Tecnologías

Como se ha mencionado en el apartado anterior, se han utilizado tecnologías disponibles en Microsoft para el desarrollo de un bot funcional. Además, se ha hecho uso de otras tecnologías para la conexión con bases de datos o servicios web. A continuación, se enumerarán las diferentes tecnologías usadas.

5.1. Lenguaje C#

Lenguaje de programación orientado a objetos. Desarrollado por Microsoft como parte de su plataforma .NET. Es una derivación de los lenguajes C y C++. Lenguaje de programación independiente diseñado para la generación de programas en la plataforma .NET. Esta plataforma administra aplicaciones que tienen como destino el Framework de .NET.

.NET ofrece el Kit de Desarrollo de Software **Bot Builder SDK.** Un conjunto de herramientas para el desarrollo de bots. Cuenta con sistema de diálogos aislados y apilables. Existe la posibilidad de implementación de inteligencia artificial con LUIS (Language Understanding Intelligent Service). Para la realización de este trabajo se ha tenido que aprender los siguientes conceptos clave:

- **Connector:** Proporciona una API REST permitiendo que un bot se comunique a través de múltiples canales. Facilita la comunicación entre bot y usuario transmitiendo mensajes a los diferentes canales.

- **Activity:** El objeto Activity es usado por el conector (Connector) para pasar información entre bot y canal (usuario). El tipo más común de actividad (Activity) es el mensaje, pero existen otros tipos de actividad.

- **Dialog:** Cuando se crea un bot usado Bot Builder SDK, podemos usar “Dialogs” para modelar una conversación y administrar el flujo de conversación. Este puede estar compuesto por otros cuadros de diálogo

y así maximizar la reutilización. Existe una pila de diálogos donde se almacenan los diálogos activos y la ruta entre diálogos.

5.1.1. Actividades

Dentro del objeto **Activity**, existen diferentes tipos actividades que podemos usar para la comunicación del bot con el canal:

message

El más común. Representa una comunicación entre bot y usuario. Algunos mensajes pueden estar formados solo por texto, mientras que otros pueden contener contenido más completo, como texto hablado, acciones sugeridas, archivos multimedia adjuntos, tarjetas enriquecidas y datos específicos del canal.

conversationUpdate

Indica que el bot se agregó a una conversación, otros miembros se agregaron o eliminaron de la conversación o los metadatos de la conversación cambiaron.

contactRelationUpdate

Indica que el bot fue agregado o eliminado de la lista de contactos de un usuario.

Typing

Indica que el usuario o bot en el otro extremo de la conversación está escribiendo una respuesta. Un bot puede enviar una actividad “typing” para indicar al usuario que está trabajando para cumplir una solicitud o compilar una respuesta.

ping

Se utiliza para determinar si nuestro bot es accesible a nivel de red.

deleteUserData

Indica a un bot que un usuario ha solicitado que se elimine cualquier dato del usuario que pueda estar almacenado.

endOfConversation

Indica el final de una conversación. El bot puede recibir o enviar este tipo de actividad.

event

Representa una comunicación enviada a un bot que no es visible para el usuario. Nuestro bot puede recibir un “event” de un proceso o servicio externo que quisiera comunicar información con él para que se la haga saber al usuario.

invoke

Nuestro bot puede recibir una actividad “invoke” que sea una solicitud para que realice una operación específica.

messageReaction

Indica que un usuario ha reaccionado a una actividad existente. Por ejemplo, si un usuario da a me gusta a un mensaje enviado por el bot la propiedad ReplyTold indicará a qué actividad reaccionó.

5.1.2. Diálogos

Los diálogos modelan la conversación. Es posible componer un diálogo con otros diálogos para maximizar la reutilización y dotar al bot de más inteligencia. Los diálogos son administrados por un **context dialog,** así se sabe en todo momento en qué diálogo se encuentra y los cuadros de diálogo por los que ha tenido que pasar.

Una conversación compuesta por diálogos es compatible en cualquier canal, haciendo así la implementación más sencilla. Utilizando diálogos, el estado de la conversación se guarda automáticamente permitiendo que el código del bot sea sin estado.

5.2. PHP

Lenguaje de código abierto especialmente diseñado para el desarrollo web y puede ser incrustado en HTML. Es un lenguaje de programación del lado del servidor. El código es interpretado por un servidor web con un módulo de procesador PHP.

No es un lenguaje muy complicado por lo que es fácil de aprender, pero a su vez ofrece muchas características avanzadas para desarrollos más profesionales.

Como es un lenguaje enfocado a la programación de scripts del lado del servidor, se puede recopilar datos de formularios, generar páginas con contenidos dinámicos, enviar y recibir cookies y muchos más.

5.3. Conexión Bases de datos SQL Server y MySQL

Para el almacenamiento de información de usuario, se utilizará SQL Server alojada en la nube de Microsoft. El lenguaje SQL fue diseñado para la administración de bases de datos relacionales. Este lenguaje permite la inserción, actualización, borrado y consultas de datos.

SQL es un lenguaje declarativo de alto nivel, permite la orientación a objetos y alta productividad de codificación. Además, es un lenguaje compatible con otros lenguajes de programación como C++, java, C#, Pascal, etc…

5.4. JSON

Lenguaje sencillo con formato de texto. Se utiliza para el intercambio de información entre sistemas. Es un lenguaje independiente de cualquier otro.

Posee librerías para codificar y decodificar datos en JSON. Su estructura es fácil de entender. Es posible la comunicación de objetos mediante este lenguaje.

En este caso, JSON se utilizará para el envío de mensajes entre aplicación externa y bot. Los JSON enviados tendrán un campo “tipo”, dependiendo de este, el resto de campos y el mensaje de notificación que crea el bot varían.

6. Herramientas

6.1. Visual Studio

Para el desarrollo de código, es necesario un editor de texto o entorno de desarrollo integrado. En este trabajo se ha trabajado con Visual Studio. Este permite trabajar en local y además enlazar el proyecto con la nube para que cualquier cambio sea actualizado al instante. Además, también tiene una herramienta para la visualización de bases de datos alojadas en la nube de Microsoft. Lo que nos servirá de ayuda para visualizar el estado de la base de datos en todo momento.

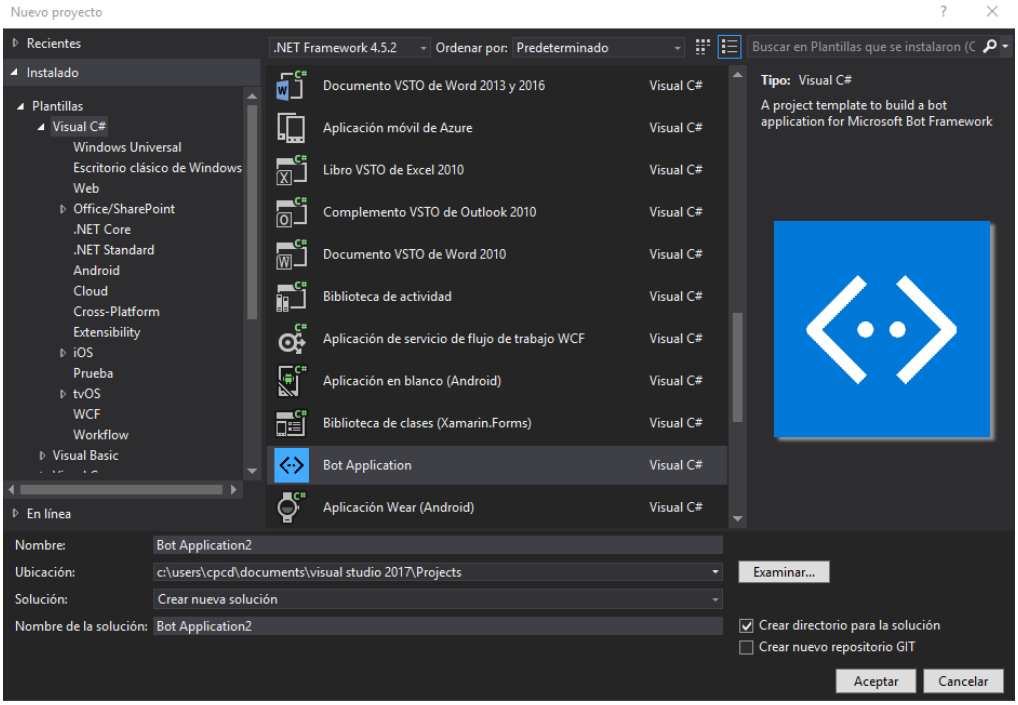


Figura 3: IDE Visual Studio 2017.

6.2. Bot Framework Channel Emulator

Aplicación de escritorio que permite a los desarrolladores probar y depurar sus bots de forma local o remota. Usando el emulador (ver **figura 4**), es posible interactuar con el bot e inspeccionar los mensajes que envía y recibe. Son mostrados de la misma forma que aparecen en la interfaz de usuario de chat web. Las solicitudes y respuestas se registran en lenguaje JSON al intercambiar mensajes con nuestro bot.

Para conectarse a un bot que se ejecuta localmente, dentro de la aplicación, en la barra azul introducimos la URL (endpoint) que se abre en nuestro navegador cuando iniciamos la aplicación en local.

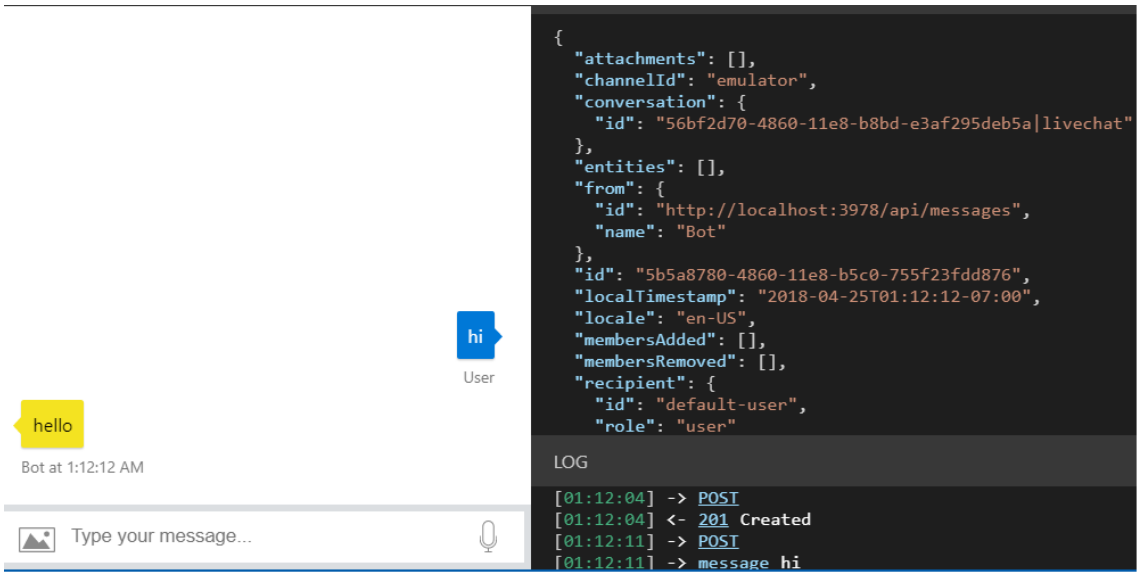


Figura 4: Captura de la interfaz de Channel Emulator.

Haciendo clic en cualquier mensaje dentro de la ventana de conversación es posible inspeccionar la actividad JSON sin procesar usando la función **“inspector”** a la derecha de la ventana. La información JSON incluye metadatos clave que incluyen el “channelID”, tipo de actividad, id de conversación, URL endpoint, etc.

Tras la fase de estudio y documentación, pondremos en práctica todo lo aprendido. Se desarrollará un bot haciendo uso de las tecnologías y herramientas mencionadas en apartados anteriores.

Se pretende mostrar el potencial de los bots en escenarios en el contexto educativo. Para ello, se crearán funcionalidades que ayuden a los usuarios en los diferentes escenarios:

Módulo de motivación y alertas

Se utilizarán las notificaciones para el programa educativo “Rétame y Aprendo”. Se consigue una experiencia de los participantes más atractiva. El concurso consiste en retos entre participantes, donde deben contestar una serie de preguntas y el que más acierte gana el reto. El juego está diseñado para navegadores web, para las notificaciones se usa el bot.

El bot notifica a los participantes cuando han sido retados o si ganan o pierden un reto. Además, al terminar cada partida el bot le recuerda el número de preguntas acertadas/falladas y dependiendo del ratio de acierto envía un mensaje personalizado.

Ayuda a docencia

En las plataformas e-learning, los docentes pueden crear tareas, añadir unidades de la asignatura, crear tipos test, etc… Los docentes pueden hacer un seguimiento desde la plataforma de los alumnos, ver notas y resultados.

El bot recoge la información de un alumno y la muestra al docente. Con un simple comando consigue obtener toda la información que necesita.

Control remoto

En la UPCT existen aulas de grabación de clases “UPCTstream”. En ellas hay instalado hardware para la grabación o retransmisión en directo de las clases. Mediante comandos en el bot, el docente es capaz de grabar, pausar y retransmitir en directo únicamente escribiendo un comando.

Cada escenario podría haber sido solucionado en un bot diferente. Se ha decidido crear solo uno para centralizar el desarrollo y mostrar la posibilidad de creación de perfiles de usuario.

En primer lugar, se analizarán las necesidades que queremos cubrir para no crear funcionalidades innecesarias. Después, se deben tomar las decisiones convenientes para adaptar el contenido para las funcionalidades y canales más adecuados. Y por último, se procederá al desarrollo de código.

7. Análisis de necesidades

Teniendo en cuenta los escenarios mencionados anteriormente, hay que tomar decisiones para el diseño y el desarrollo. Se tiene como objetivo conseguir un bot más amigable, sencillo y escalable pensando en posibles desarrollos futuros.

El bot necesitará:

- **Canales** de comunicación.
- **Listado de preguntas frecuentes (FAQ).**
- Comunicación mediante **estímulo-respuesta**.
- Diferentes **Perfiles de usuario.**
- **Almacenar** información de usuario.
- **Conexión con aplicaciones externas.**
- **Conexión con hardware remoto.**

Cumplir estas necesidades conllevará tomar una serie de decisiones en el desarrollo que veremos en el siguiente apartado.

8. Toma de decisiones

Fue necesario tener en cuenta las limitaciones de tiempo y tecnologías, y se aprovecha la conexión entre la Universidad Politécnica de Cartagena y Microsoft. Por esto, se tomó la decisión de crear y alojar el bot haciendo uso de los servicios que nos ofrece Microsoft. Alojar el bot en Azure garantiza:

1. **Escalabilidad.** No existe un número fijo de usuarios, ni el número de conexiones simultaneas. Azure se encarga de forma automática de obtener los recursos si fuese necesario.
2. **Seguridad.** Azure cuenta con grandes niveles de seguridad y protección de datos.
3. **Disponibilidad.** Azure dispone de respaldo de servidores.

8.1. Canales

Para conseguir la mayor difusión posible, se debe decidir los canales de comunicación con nuestro bot. Activar demasiados canales, aumenta el trabajo de administración y el desarrollo. También se tuvo en cuenta el público al que iba dirigido el bot, alumnos de instituto y profesores.

Se llegó a la conclusión de que los canales más utilizados por los usuarios serían **Facebook Messenger**, **Telegram** y **Skype**. También se activará el canal **Microsoft Teams,** es menos utilizado debido a que es relativamente nuevo, pero tiene un gran potencial, ya que es la apuesta de Microsoft como herramienta para la docencia y trabajo en equipo.

Para la conexión con una aplicación externa, se utilizará el **canal Direct Line.** Este canal crea un enlace entre la aplicación y el bot para la comunicación entre ambos mediante un servicio REST. Gracias a Direct Line la aplicación es capaz de crear y enviar mensajes con un formato que el bot es capaz de comprender. Bot Framework genera una clave que deberá utilizar la aplicación para conectar al canal Direct Line. Será necesario crear un desarrollo en ambas partes (aplicación cliente y bot). Estos mensajes pasan desapercibidos para el usuario.

El bot envía distintos avisos al usuario, dependiendo del tipo de mensaje recibido por la aplicación externa.

8.2. Interacción con bot

Otra decisión a tener en cuenta es cómo el usuario interacciona con el bot, ya que como se comentó en apartados anteriores, las primeras interacciones son muy importantes y estas pueden ser por comandos o por menús para ser más guiados.

Se pretende crear un bot con funcionalidades muy diferentes unas de otras. Crear interacciones guiadas con menús no es demasiado práctico, ya que un participante del programa educativo “Rétame y aprendo” solo necesita autentificarse en el bot para que este compruebe que lo es. En cambio un docente no necesita este tipo de autenticación, sino que necesita entrar al “modo docente”.

Se decide utilizar comandos, así se consigue que el bot sea más intuitivo. Los comandos se diferencian del lenguaje natural mediante una barra delante de ellos. A continuación se muestran todos comandos repartidos en los diferentes diálogos:

- **/ayuda:** Devuelve la lista de comandos que se pueden utilizar.
- **/faq:** Devuelve una url que nos lleva a las preguntas frecuentes de rétame y aprendo.
- **/start:** En el diálogo principal, inicializamos el registro.
- **/profesor:** Accedemos al modo profesor.
- **/exit:** Salimos del diálogo actual.
- **/stats:** En modo profesor, con este comando obtendremos información de un alumno.

- **/remote:** En modo profesor, accederemos al modo control remoto.

- **/grabar:** En modo control remoto, activaremos la grabación del dispositivo multimedia.

- **/pausar:** En modo control remoto, pausaremos la grabación del dispositivo multimedia.

- **/parar:** En modo control remoto, pararemos la grabación del dispositivo multimedia.

- **/directo:** En modo control remoto, retransmitiremos en streaming.

- **/parardirecto:** En modo control remoto, pararemos la retransmisión del streaming.

8.3. Comprensión del bot

Microsoft Bot Framework nos permite utilizar una API de comprensión de lenguaje natural LUIS. Otra opción es analizar la entrada del usuario a través de expresiones regulares.

Se llegó a la conclusión de que el implementar la API LUIS tendría un coste de tiempo de desarrollo y tiempo de aprendizaje demasiado elevado, quedando fuera del alcance de este proyecto y se deja como trabajo futuro.

Se decidió **utilizar expresiones regulares** para diferenciar comandos además de alguna que otras palabras clave para dar la sensación de mayor inteligencia y comprensión de nuestro bot. Los comandos se diferenciarían del resto mediante la barra inclinada o la barra inclinada inversa, en el caso de Skype.

8.4. Almacenamiento de información

Los mensajes que envía el usuario contienen, además de texto, otra información útil en la conversación. Esta información es necesaria a la hora de crear notificaciones (el mensaje enviado no es la respuesta a un mensaje de usuario). La información que se necesita para la creación de notificaciones es:

- **ChannelID:** ID de canal donde se envía el mensaje.
- **ChatID:** Identificador único de conversación.

Para obtener estos datos y almacenarlos para futuros usos, se ha implementado el diálogo de registro “StartDialog”. Si un usuario de “Rétame y Aprendo” quiere recibir notificaciones se registrará con un canal.

Con el comando “/start” empieza el registro de usuario. Solo si es participante de “Rétame y Aprendo”, DNI, “ChatID”, “ChannelID” se guardarán en una base de datos en la nube para su uso en las notificaciones.

Un usuario puede tener dos registros en la base de datos con “ChatID” diferentes, si se ha registrado en dos canales (Skype y Telegram por ejemplo).

La información almacenada y como se utiliza se explicará en capítulos futuros en el apartado **desarrollo de bot.** Al igual que las funciones necesitadas y la creación del mensaje sin necesidad de ser un mensaje retroactivo.

8.5. Flujo de conversación

Se desea un flujo de conversación lo más práctico y funcional posible. Hay que tener en cuenta el número funcionalidades del bot y crear diálogos dependiendo de estas. Como ya se ha comentado en apartados anteriores, tenemos las funcionalidades de registro, docente y control remoto. Esta última solo podrán acceder los docentes por lo que irá anidado dentro del diálogo de docente.

Con toda esta información, la distribución de diálogos para el flujo de conversación queda de la siguiente manera, como se ve en la **figura 5**:

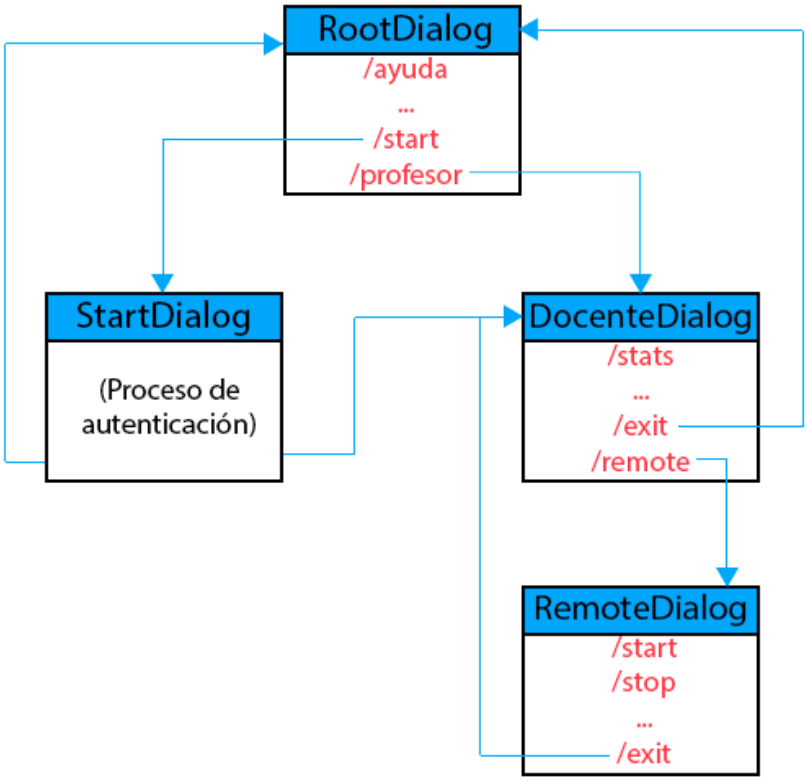


Figura 5: Esquema flujo de conversación (Elaboración propia).

Como vemos en la imagen, al interactuar la primera vez con el bot, el usuario se encontrará en **RootDialog**, el nivel más alto donde se podrá acceder a los comandos de información general, preguntas frecuentes además de poder interactuar con nuestro bot en conversaciones simples. Con el comando "/start" el control pasa a **StartDialog** donde se registrará al usuario si pertenece a "Rétame y Aprendo" o si es docente se le redirigirá al diálogo de docente. Cuando el proceso de registro termine el control volverá al diálogo principal (RootDialog). Con el comando "/profesor", **RootDialog** pedirá autenticación, solo si el usuario es docente se accederá a **DocenteDialog** donde los comandos a utilizar son diferentes (por ejemplo el comando "/ayuda" devuelve una ayuda diferente que en el diálogo principal). Con el comando "/remote" se accede al diálogo **RemoteDialog** con comandos para controlar remotamente el sistema de retransmisión online.

Los diálogos **RemoteDialog** y **DocenteDialog** no devuelven el control de la conversación automáticamente, por lo que para volver a diálogos anteriores se debe utilizar el comando “/exit”.

Se ha decidido hacer así ya que los alumnos no tienen acceso a las funcionalidades de docente y un docente no recibe avisos motivacionales en el diálogo principal.

8.6. Tipo de mensajes enriquecidos

Para los mensajes de motivación, y para mostrar el perfil de usuario se ha decidido enviar mensajes con tarjetas enriquecidas. De entre todas las posibles se decidió utilizar “Hero Cards”. Con la herramienta web “Channel Inspector” de Bot Framework, se vio que este tipo de mensaje era compatible con los canales que utiliza nuestro bot.

Las “Hero Cards” contarán con una imagen, un título, un pequeño texto y además botones para acceder a enlaces externos. Así conseguiremos dotar a nuestro bot con un diseño más personalizado y dinámico, no siendo todas las respuestas en texto plano.

9. Desarrollo

Después de realizar el análisis de necesidades y la toma de decisiones, se procede a la creación del bot. Desde Azure, en el apartado “bots”, pulsando en “Create a bot”, se elige “Web App Bot” e introducir los datos que se piden. Una vez creado, se genera un **id** y **contraseña** que deben ser guardados para utilizarlos en el archivo de configuración.

En la siguiente **figura**, se muestra cómo es la interfaz de creación de bot y los diferentes campos que se deben completar.

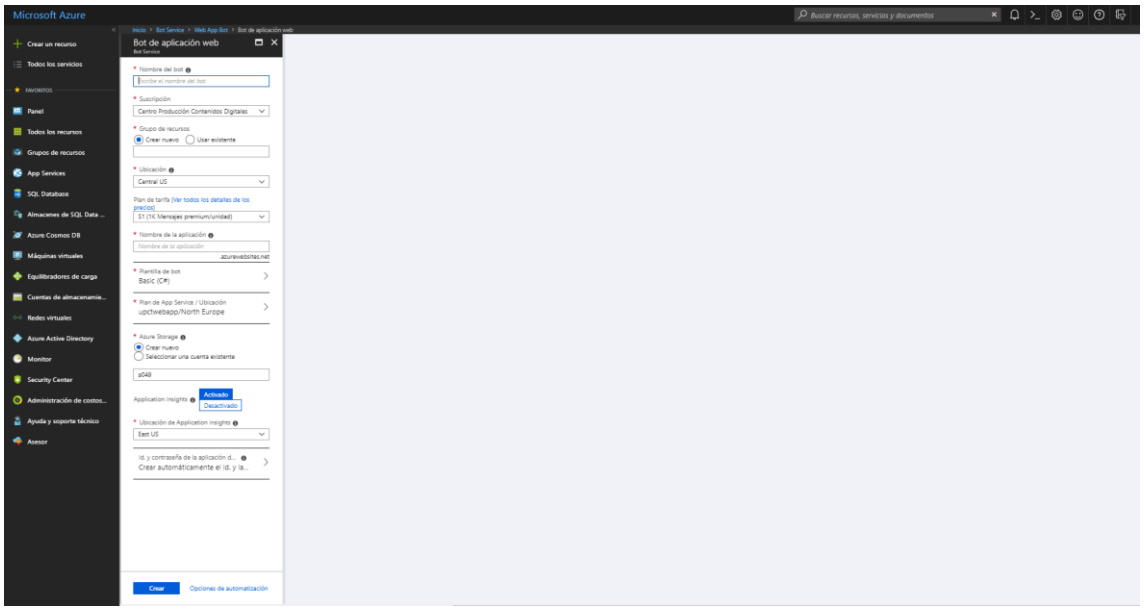


Figura 6: Interfaz creación de bot Azure.

Para el desarrollo de código en Visual Studio 2017, se crea un nuevo proyecto con una plantilla básica. En Visual Studio 2017, pulsamos en Archivo/nuevo/proyecto, nos aparecerá una ventana emergente y deberemos elegir dentro de las plantillas de visual C# la plantilla “Bot Application” como se muestra en la **figura 7**. Comenzar con una plantilla ayuda a tener el sistema de archivos y carpetas ya estructurado.

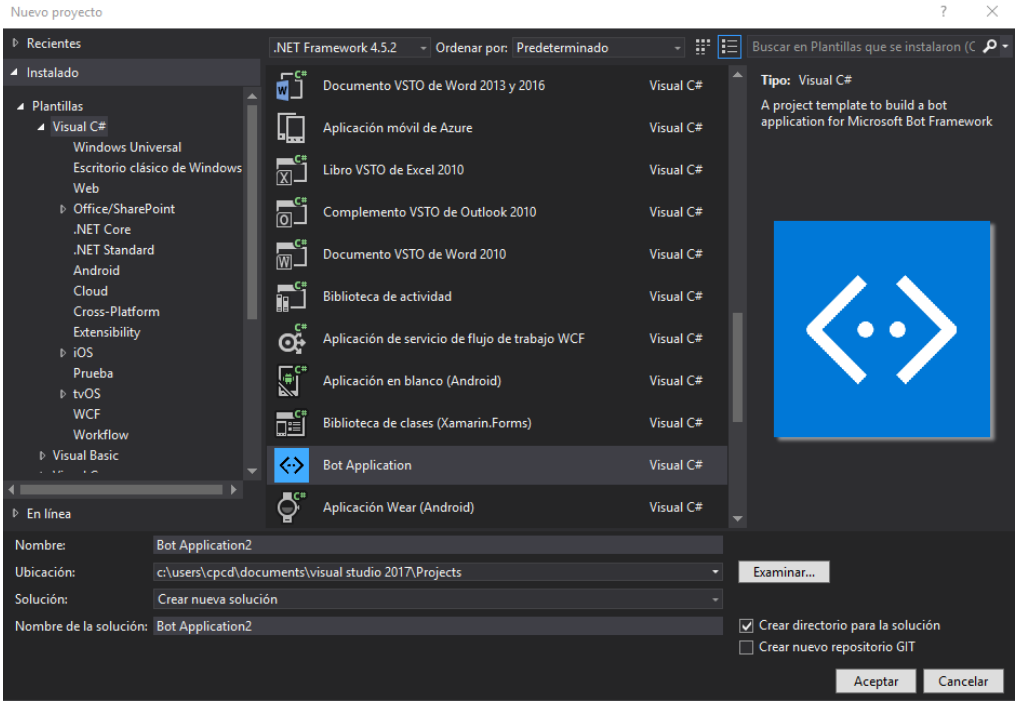


Figura 7: Selección de plantilla en la creación de proyecto.

Una vez creado nuestro proyecto, la disposición de los archivos y carpetas queda de la siguiente forma:

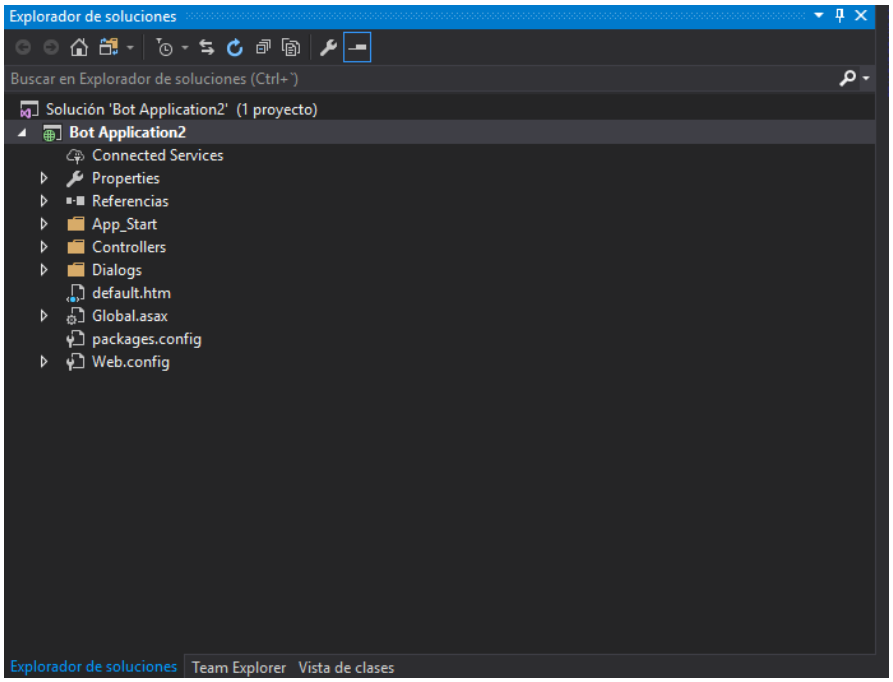


Figura 8: Disposición de carpetas en el proyecto de Visual Studio.

Los archivos y directorios en los que se trabajará: serán los directorios “Controllers” y “Dialogs”, y el archivo “Web.config”.

- Directorio **Controllers:** dentro de este directorio se encuentra “MessageController.cs”, cerebro encargado de redirigir a diferentes flujos de conversación. Dependiendo del tipo de mensaje llama a diálogos o realiza operaciones internas.
- Directorio **Dialogs:** en él se ubican “RootDialog.cs” encargado del flujo principal de la conversación. Desde él también se llama a otros diálogos.
- Archivo **Web.config:** donde se debe introducir en los campos “value” de “MicrosoftAppId” y “MicrosoftAppPassword” (ver **figura 9**) el id y contraseña que se obtiene al crear un bot.

```
<configuration>
  <appSettings>
    <!-- update these with your BotId, Microsoft App Id and your Microsoft App
Password-->
    <add key="BotId" value="YourBotId" />
    <add key="MicrosoftAppId" value="(ID A INTRODUCIR)" />
    <add key="MicrosoftAppPassword" value="(PASSWORD A INTRODUCIR)" />
  </appSettings>
```

Figura 9: Parte de código a modificar en el archivo Web.config

9.1. MessageController

Cuando un usuario se comunica con el bot, se recibe una actividad que es analizada por el controlador “MessageController.cs” y este decide qué hacer con el mensaje recibido. Si la actividad es del tipo mensaje, se llama al diálogo principal (RootDialog).

Sin embargo, no todas las actividades son mensajes. Estas pueden ser de diferentes tipos. Aquí es donde entra el método HandleSystemMessage(), encargado de decidir qué sucede cuando hay otro tipo de acción.

```
public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
{
      if (activity.Type == ActivityTypes.Message)
      {
```

```
                await Conversation.SendAsync(activity, () => new
Dialogs.RootDialog());
        }
        else
        {
                await HandleSystemMessage(activity);
        }
        var response = Request.CreateResponse(HttpStatusCode.OK);
        return response;
}
```

Figura 10: Método Post() de MessageController.cs

```
private async Task<Activity> HandleSystemMessage(Activity message)
{
        if (message.Type == ActivityTypes.Event)
        {
                // Conexión con aplicación externa y módulo de motivación
        }
        else if (message.Type == ActivityTypes.DeleteUserData)
        {
        }
        else if (message.Type == ActivityTypes.ConversationUpdate)
        {
        }
        else if (message.Type == ActivityTypes.ContactRelationUpdate)
        {
        }
        else if (message.Type == ActivityTypes.Typing)
        {
        }
        return null;
}
```

Figura 11: Método HandleSystemMessage sin acciones de MessageController.cs

En el caso de nuestro bot, este último método se modificará para la comunicación con la aplicación externa para el proceso de motivación y envío de notificaciones, utilizando el canal **Direct Line** de Bot Framework y otras tecnologías que se desarrollarán en el siguiente apartado.

9.2. Diálogos y flujos de conversación

Nuestro bot contiene 4 diálogos: **RootDialog**, **StartDialog**, **DocenteDialog** y **RemoteDialog**. Para detectar los comandos y palabras clave, los diálogos llaman a una función auxiliar (Apartado 9.3). Dependiendo del valor obtenido por

la función auxiliar, los diálogos actuarán de diferente manera. Se va a proceder a explicar el código más relevante en cada uno de ellos.

9.2.1. RootDialog

Es el diálogo principal, encargado del flujo de conversación inicial. Dependiendo de las palabras clave o comandos utilizados, llama a otros diálogos o muestra la información solicitada.

Existen diferentes reacciones en el diálogo principal. Las más sencillas son las que corresponden a los índices 3, 4, 5 y 6. Estas respuestas sirven para dotar al bot de una falsa consciencia e inteligencia artificial. Lo hace capaz de contestar a un saludo, una despedida, un insulto o un halago.

```
else if (indice == 3)
{
       await context.PostAsync($"No me insultes yo solo soy un robot :(");
}
else if (indice == 4)
{
       await context.PostAsync($":)");
}
else if (indice == 5)
{
       await context.PostAsync($"Hola! {activity.From.Name}");
}
else if (indice == 6)
{
       await context.PostAsync($"Que descanses!");
}
```

Figura 12: Respuestas para dotar de consciencia.

En los índices 1 y 2 se encuentran las respuestas a los comandos de ayuda (/ayuda) o preguntas frecuentes (/faq). El primero devolverá uno a uno los comandos almacenados en un diccionario. Mientras que el segundo devuelve una URL donde se encuentran las preguntas frecuentes.

```
If (indice == 1)
{
       await context.PostAsync($”Veo que necesitas ayuda, en un momento te mando
una lista de todo lo que sé hacer”);
       await Task.Delay(TimeSpan.FromSeconds(1));
```

```
        AllHelps helps = new AllHelps();
        if (activity.ChannelId == “skype”)
        {
                foreach (KeyValuePair<string, string> command in helps.AyudaSk)
                {
                        await context.PostAsync($”{command.Key}: {command.Value}”);
                }
        }
        else
        {
                foreach (KeyValuePair<string, string> command in helps.Ayuda)
                {
                        await context.PostAsync($”{command.Key}: {command.Value}”);
                }
        }

}
```

Figura 13: Respuesta al comando “ayuda”.

```
else if (indice == 2)
{
await context.PostAsync($"Aquí tienes el faq");
await context.PostAsync($"http://retame.upct.es/preguntas_mas_frecuentes.html");
}
```

Figura 14: Respuesta al comando de preguntas frecuentes.

Los índices 7 y el 8 llaman a diálogos secundarios para que tomen el control de la conversación. En el primero de estos, se pasa el control a “StartDialog” para pasar al registro de usuarios. Y el segundo, pasa el control a “DocenteDialog”, si antes el DNI introducido corresponde con el de un docente registrado. El DNI se verifica en la función “IntroDNI”.

```
else if (indice == 8)
{
        await context.PostAsync($"Bienvenido al perfil del profesor");
        PromptDialog.Text(context, IntroDNI, $"¿Cuál es tu DNI?");
}
```

```
private async Task IntroDNI(IDialogContext context, IAwaitable<string> result)
{
Activity activity = context.Activity as Activity;
string dni = await result;

if (ConnectMethods.SearchDocente(dni))
{
        await context.PostAsync($"DNI de docente correcto");
        if (activity.ChannelId == "skype")
        {
                await context.PostAsync($"Recuerda que con el comando \\ayuda puedo
decirte que comandos puedes utilizar y con \\exit salir de este modo");
        }
```

```
        else
        {
                await context.PostAsync($"Recuerda que con el comando /ayuda puedo
decirte que comandos puedes utilizar y con /exit salir de este modo");
        }
        context.Call(new DocenteDialog(), AfterOperacion);
}
else
{
        await context.PostAsync($"No eres docente");
}
```

Figura 15: Acceso a modo docente y función IntroDNI.

Si el índice devuelto por la función auxiliar no coindice con ninguno de los anteriores, se devuelve una respuesta estándar notificando que el bot no ha sido capaz de entender al usuario.

```
else
{
        if (activity.ChannelId == "skype")
        {
                await context.PostAsync($"No te he entendido intenta utilizar algún
comando. Puedes pedirme ayuda con el comando \\ayuda");
        }
        else
        {
                await context.PostAsync($"No te he entendido intenta utilizar algún
comando. Puedes pedirme ayuda con el comando /ayuda");
        }
        context.Wait(MessageReceivedAsync);
}
```

Figura 16: Respuesta a comando desconocido.

9.2.2. StartDialog

Cuando es “StartDialog” el que tiene el control de la conversación, su primera acción es pedir al usuario su DNI. Se coteja en la tabla de usuarios de “Rétame y Aprendo” y en la base de datos de Docentes.

Existen tres opciones:

1. **Usuario de “Rétame y Aprendo”:** DNI, “ChatID”, “ChannelID” se guardarán en una base de datos en la nube para su uso en futuras notificaciones. Por último, se devuelve el control de la conversación al diálogo principal con el código “Context.Done(true)”.

2. **Usuario docente:** El bot no almacena ninguna información y redirigirá a este usuario al diálogo “DocenteDialog” con el código “context.Call(new DocenteDialog(), AfterOperacion)”.
3. El usuario que **no cumple** ninguna de estas **condiciones**, no es registrado y no recibirá notificaciones.

9.2.3. DocenteDialog

Este diálogo, al igual que “RootDialog”, utiliza la función auxiliar. Acepta tres comandos, con el primero se devuelve el progreso académico de un alumno. El segundo, devuelve el control de la conversación al diálogo principal. Y el tercero entrega el control de conversación “RemoteDialog”.

```
if (indice == 1)
{
       PromptDialog.Text(context, IntroNombre, $"Introduce nombre completo del
alumno a consultar sus estadísticas");
}
```

Figura 17: PromptDialog que llama a la función de progreso de alumno.

9.2.4. RemoteDialog

Este último diálogo sirve de control remoto para el dispositivo multimedia de grabación y transmisión en streaming. Dependiendo del comando utilizado por el profesor, el bot ejecuta un archivo PHP alojado en un servidor interno de la UPCT en la misma red interna donde se encuentra el dispositivo multimedia, evitando así retrasos y otros posibles fallos. A continuación se muestra en la **figura 18**, 3 de los 5 posibles comandos a utilizar en este diálogo.

```
if (indice == 1)
{
       await context.PostAsync($"Grabando, actualiza el reproductor");
       var client = new WebClient();
       var content = client.DownloadString("URL_Comando_Start");
}
else if (indice == 2)
{
```

```
        await context.PostAsync($"Video pausado");
        var client = new WebClient();
        var content = client.DownloadString("URL_Comando_Pause");
}
else if (indice == 3)
{
        await context.PostAsync($"Video parado");
        WebClient client = new WebClient();
        var content = client.DownloadString("URL_Comando_Stop");
}
```

Figura 18: Acciones del control remoto.

9.3. Funciones auxiliares

9.3.1. Comandos

Como se menciona en apartados anteriores, para la comprensión de comandos se creó el archivo “GetCommand.cs”. En él se tienen tantas funciones como diálogos con posibilidad de escribir comandos.

Cada función consta de una variable “i” inicializada a 0 y un “Array” con un número de entradas igual a respuestas del diálogo. La función recorre el Array buscando coincidencias. Si la palabra o comando introducido por el usuario coincide con alguna expresión regular, la función devolverá el valor de “i”. Si no coincide, el valor de “i” aumenta.

```
public static int ExtractCommand(string command)
{
        int i = 0;
        int comand = 0;
        string text = command;
        string [] pat
={@"(ayudar|ayuda|ayudame|ayúdame|ayudarme|help|que\shacer|qué\shacer|Ayuda|\/ayuda)"
,

@"(preguntas\sfrecuentes|dudas|preguntas|duda|faq|Faq|\/faq)",
                                @"(puto|puta|cabron|cabrón|muertos|a\sla\smierda)",
                                @"gracias|muchas\sgracias|te\squiero|guapo|guapa",
                                @"buenos\sdias|buenos\sdías|buenas\stardes|hola|Hola",
                                @"buenas\snoches|que\sdescanses|dormir|buena\snoche",
                            @"\/start",
                            @"\/profesor"};
        foreach(string pattern in pat)
        {
        i++;
        Regex r = new Regex(pattern);
```

```
        bool m = r.IsMatch(text);

        if (m)
        {
        comand = i;
        }
        }
        return comand;
}
```

Figura 19: Función de extracción de comando.

En la siguiente figura (**figura 20**) se muestran las funciones que utilizaremos para los diálogos remoto y docente. Se recuerda, que existen funciones idénticas a estas con la barra de comandos diferente para la comprensión de comandos en Skype.

```
public static int ExtractCommandDoc(string command)
{
        int i = 0;
        int comand = 0;
        string text = command;
        string[] pat ={@"\/stats",
                       @"\/exit",
                       @"\/remote",
                       @"\/ayuda"};
        foreach (string pattern in pat)
        {
                i++;
                Regex r = new Regex(pattern);
                bool m = r.IsMatch(text);

                if (m)
                {
                        comand = i;
                }
        }
        return comand;
}


public static int ExtractCommandRemote(string command)
{
        int i = 0;
        int comand = 0;
        string text = command;
        string[] pat ={@"\/grabar",
                       @"\/pausar",
                       @"\/parar",
                       @"\/directo",
                       @"\/parardirecto",
                       @"\/exit",
                       @"\/ayuda"};
        foreach (string pattern in pat)
        {
```

```
                i++;
                Regex r = new Regex(pattern);
                bool m = r.IsMatch(text);

                if (m)
                {
                        comand = i;
                }
        }
        return comand;
}
```

Figura 20: Función de extracción de comando.

9.3.2. Lista de comandos

Cuando un usuario pide ayuda, el bot responde con la lista de comandos disponibles, esta lista está creada con la instancia “Dictionary” que representa una colección de claves valores.

Cuando un usuario introduce “/ayuda”, el diálogo va recorriendo el diccionario de comandos y devolviendo uno a uno los comandos con una pequeña descripción.

En la **figura 21** se muestra el código implementado en “RootDialog.cs” que resuelve la respuesta cuando un usuario pide ayuda.

```
await context.PostAsync($"Veo que necesitas ayuda, en un momento te mando una
lista de todo lo que sé hacer");
await Task.Delay(TimeSpan.FromSeconds(1));
AllHelps helps = new AllHelps();
if (activity.ChannelId == "skype")
{
        foreach (KeyValuePair<string, string> command in helps.AyudaSk)
        {
                await context.PostAsync($"{command.Key}: {command.Value}");
        }
}
else
{
        foreach (KeyValuePair<string, string> command in helps.Ayuda)
        {
                await context.PostAsync($"{command.Key}: {command.Value}");
        }
}
```

Figura 21: Respuesta cuando un usuario pide ayuda.

Y por último vemos un ejemplo de colección clave valor.

```
private readonly Dictionary<string, string> ayuda = new Dictionary<string, string>()
{
        { "\"/ayuda\"", "Te devolveré todo lo que puedo hacer" },
        { "\"/faq\"", "Te diré qué es lo que se preguntan los demás usuarios" },
```

```
        { "\"/start\"", "Registraré tu DNI con la red social que estés utilizando para
los avisos de UPCTplay"},
        { "\"/profesor\"", "Si eres profesor, entrarás en modo \"docente\" y tendras
acceso a otras funcionalidades"},
        { "\"/exit\" en modo \"docente\"", "Salir del modo docente"}
};
```

Figura 22: Ejemplo colección clave valor.

9.3.3. Conexiones a base de datos

Por último, en el archivo “ConnectMethods.cs” se encuentran todas las conexiones a las bases de datos, en este proyecto se hace conexión a tres bases de datos diferentes: “Rétame y Aprendo”, base de datos de formación y base de datos de registro de nuestro bot.

La base de datos interna utilizada es una base de datos de alta disponibilidad SQL que tenemos a nuestra disposición en “Azure Portal” como ya se mencionó en un apartado anterior, así esta base de datos tendrá una disponibilidad casi del 100% y en la nube.

La base de datos del programa educativo “Rétame y aprendo” usa la tecnología MySQL por lo que las consultas son diferentes pero no supone ningún problema de desarrollo.

9.4. Módulo de motivación y notificaciones

Se procede a explicar el proceso completo desde que el juego “La ruleta” de “Rétame y Aprendo” envía un evento hasta que se recibe el mensaje de notificación o motivación en el usuario. A continuación, se verá una pequeña explicación de las tecnologías utilizadas, además de explicar el módulo de motivación que se encuentra dentro de nuestro bot.

En la siguiente figura (**figura 23**) se muestra la arquitectura que se sigue desde que se crea el evento hasta que el mensaje es recibido por el usuario. Todos estos servicios se encuentran en la nube, así conseguimos tener un sistema

escalable. Esto se debe a que no sabemos los momentos del día con más carga o el número de usuarios totales.

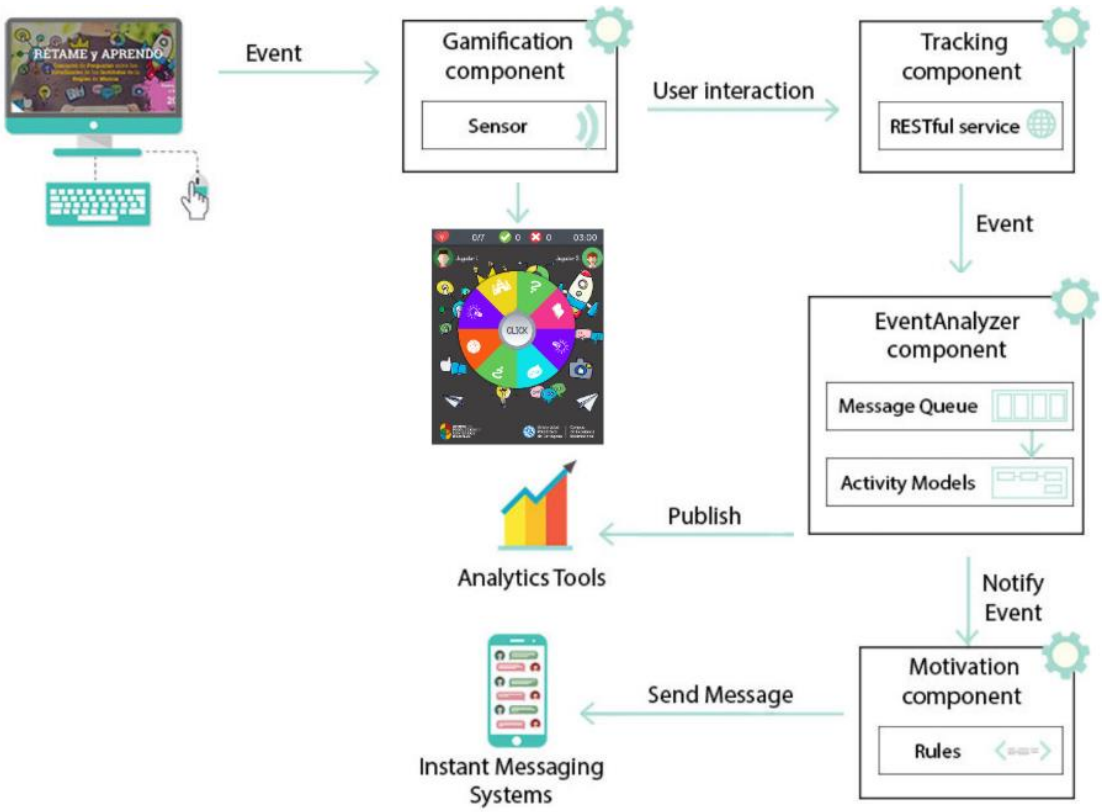


Figura 23: Arquitectura de eventos. (Creación Daniel Pérez Berenguer)

9.4.1. Sensor

Es necesario utilizar un sensor para recoger toda la información que el juego genera. Para el análisis se utiliza el estándar **Caliper Analytics.**

El estándar Caliper Analytics, permite etiquetar eventos basados en otros estándares para poder ser procesados en aplicaciones externas. Caliper puede recopilar e intercambiar datos de actividades de aprendizaje. Las aplicaciones que reciben esta información deben ser capaces de recibir mensajes Caliper a través del protocolo HTTP.

En nuestro archivo sensor, creamos un nuevo sensor ($sensorplay) e introducimos los eventos que queremos analizar en el bot. Esta información es

almacenada en formato tipo JSON y en el siguiente paso el servicio REST la recoge para enviarla.

```
$sensorplay = new SensorUPCTplay('SensorUPCTplay');
$sensorplay->currentUserSext($dni, $person1['nombre'],
$person1['email']);
$sensorplay->newAssessementEvent($id, $person2['dni'], $person2['email'],
$actividad, $person2['nombre']);
```

Figura 24: Código del sensor del componente de gamificación.

9.4.2. RESTfull Service

Un servicio REST es cualquier interfaz entre sistemas que utiliza directamente HTTP para obtener datos o indicar ejecución de operaciones sobre datos en cualquier formato ya sea XML, JSON, etc. El servicio REST nos ofrece gracias a su diseño:

- **Protocolo cliente/servidor sin estado:** El cliente y el servidor no necesitan recordar ningún estado de las comunicaciones entre mensajes ya que cada mensaje HTTP contiene toda la información necesaria. Aunque en la práctica, muchas aplicaciones utilizan cookies y otros mecanismos para mantener el estado de sesión.

- **Conjunto de operaciones bien definidas:** Como funciona sobre HTTP, este define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.

- **Sintaxis universal:** En un sistema REST, los recursos se identifican a través de su URI.

- **Uso de hipermedios:** Es posible navegar de un servicio REST a muchos otros simplemente siguiendo enlaces sin necesidad de usar registros u otra infraestructura adicional.

Para nuestro uso, el servicio REST obtiene el archivo JSON creado a partir del sensor de eventos y se envía un mensaje con método POST() a la cola de mensajes.

```
var queueMessage = new CloudQueueMessage(JsonConvert.SerializeObject(value));
var message = new BrokeredMessage(JsonConvert.SerializeObject(value));
eventosUPCTRequestQueue.Send(message);
```

Figura 25: Código RestFul

9.4.3. Cola de mensajes

La cola de mensajes está implementada en el servicio de Azure, **Service Bus**. Este servicio de mensajería es multiinquilino y en la nube de Azure. Envía información entre aplicaciones y servicios. Ofrece funcionalidades de mensajería estructurada de tipo FIFO (first in, first out).

Las colas permiten almacenar mensajes hasta que la aplicación receptora está disponible para recibirlos y procesarlos. Los mensajes en cola se ordenan y a su llegada se le asigna una marca de tiempo.

Figura 26: Tipo de cola FIFO que proporciona Service Bus.

(Fuente: https://docs.microsoft.com/es-es/azure/service-bus-messaging/service-bus-messaging-overview)

En nuestro caso se ha implementado esta tecnología para asegurar que el orden de los eventos de salida es igual al orden de los eventos de entrada, evitando que, por ejemplo, al usuario se le notifique antes del resultado de un reto sin haber recibido antes la notificación de que ha sido retado.

El siguiente paso es el **WebJob**. Este recibe una notificación de la cola si hay mensajes, selecciona y procesa uno a uno los mensajes.

9.4.4. WebJob

"WebJobs" es una característica de Azure que nos permite ejecutar un programa o script en el mismo contexto que una aplicación web, una API o una aplicación móvil. Es posible programar cuando se ejecuta el programa o script. En nuestro caso, el WebJob está programado para ejecutarse cada vez que la cola de mensajes tiene un elemento en ella.

Para el caso del bot, el WebJob cogerá el mensaje encolado y lo enviará al bot. Es necesario crear la comunicación con este a través de **Direct Line**, por lo que el canal debe de estar activo. Copiaremos la clave secreta que obtenemos dentro de la configuración del canal **Direct Line.** En el WebJob creamos la función encargada de generar el token, crear la conversación, crear el mensaje y mandarlo.

```
private static async Task StartBotConversation(string ptexto, string dni)
{
    // Obtención de token
    var tokenResponse = await new
DirectLineClient(directLineSecret).Tokens.GenerateTokenForNewConversationAsync();
    // Use token to create conversation
    var directLineClient = new DirectLineClient(tokenResponse.Token);
    var conversation = await
directLineClient.Conversations.StartConversationAsync();
    Activity userMessage = new Activity
    {
        From = new ChannelAccount(fromUser),
        Text = ptexto,
        Type = ActivityTypes.Event,
        Value = dni
    };
    await
directLineClient.Conversations.PostActivityAsync(conversation.ConversationId,
userMessage);
}
```

Figura 27: Función de conexión con bot.

Los eventos recibidos por el "WebJob" pueden ser de 3 tipos: **action#Started**, **action#Submitted** y **OutcomeEvent**. Dependiendo del tipo de evento el JSON creado tiene diferentes campos. Y el campo tipo indica el tipo de mensaje que deberá crear el bot.

Evento **action#Started:**

```
string jsont = "{\"tipo\": 1, \"jugador1\": \"" + nommot + "\", \"jugador2\": \""
+ oponente + "\"}";
```

Evento **action#Submitted:**

```
string jsont = "{\"tipo\": 0, \"nombre\": \"" + nommot + "\", \"acertadas\": \"" +
acertadas + "\", \"falladas\": \"" + falladas + "\"}";
```

Evento **OutcomeEvent:**

```
string jsont = "{\"tipo\": 2, \"jugador1\": \"" + nommot + "\", \"jugador2\": \""
+ oponente + "\", \"resultado\": \"" + ret.ganador + "\"}";
```

9.4.5. Módulo de motivación

Por último, el mensaje enviado mediante el WebJob es recibido por el controlador de nuestro bot. Como la actividad es de tipo evento, se descompone el JSON recibido. Mediante "switch" contemplamos los tres casos posibles (el campo "type" puede tener el valor 0, 1 o 2).

```
JObject json = JObject.Parse(message.Text);
//Selección del tipo de mensaje a enviar y contenido
switch ((int)json["tipo"])
{
```

Figura 28: Análisis del tipo de mensaje a enviar.

Si **tipo** es **0**:

El JSON recibido contiene los campos “Tipo”, “nombre”, “acertadas” y “falladas”. Con esos datos se obtiene el ratio de acierto con la función “CalcularRatio” (**figura 30**). El bot crea un mensaje en “Hero Cards” diferente dependiendo del ratio obtenido. En **figura 29** se observan los diferentes mensajes en “Hero Cards” para los distintos ratios de acierto.

```
case 0:
//obtención de los diferentes datos de mensaje de tipo 0
        nombre = (string)json["nombre"];
        acertadas = (string)json["acertadas"];
        falladas = (string)json["falladas"];

        ratio = CalcularRatio(acertadas, falladas); //calculo de ratio

        if (ratio >= 0 && ratio < 0.5)
        {
                titulo = $"¡No te preocupes {nombre}!";
                image = "http://cpcd.upct.es/imagenes_bot_teams/notepreocupes.jpg";
                texto = $"La próxima partida seguro que te saldrá mejor.";
        }
        else if (ratio >= 0.5 && ratio < 0.8)
        {
                titulo = $"¡Buena partida {nombre}!";
                image = "http://cpcd.upct.es/imagenes_bot_teams/buenapartida.jpg";
        }
        else if (ratio >= 0.8 && ratio < 1)
        {
                titulo = $"¡Genial {nombre}!";
                image = "http://cpcd.upct.es/imagenes_bot_teams/casipleno.jpg";
                texto = $"Has estado muy cerca del pleno.";
        }
        else if (ratio == 1)
        {
                titulo = $"¡Vaya Pleno {nombre}!";
                image = "http://cpcd.upct.es/imagenes_bot_teams/pleno.jpg";
                texto = $"Será muy difícil que te superen.";
        }
        attachment = GetCard(image, titulo, texto, acertadas, falladas);

        break;
```

Figura 29: Mensaje motivacional tipo 1

```
public static double CalcularRatio(string acertadas, string falladas)
{
        double res = 0;
        if (Convert.ToDouble(acertadas) == 0)
        {
                res = 0;
        }
```

```
        else
        {
                double total = Convert.ToDouble(acertadas) +
Convert.ToDouble(falladas);
                res = Convert.ToDouble(acertadas) / total;
        }
        return res;
}
```

Figura 30: Función que calcula el ratio acertadas-falladas

Si **Type** es **1**:

El JSON recibido contiene los campos “Tipo”, “jugador1” y “jugador2”. Con esta información, el bot crea y envía un mensaje de aviso al jugador retado. Así, este sabe que tiene una partida pendiente.

```
case 1:
        jugador1 = (string)json["jugador1"];
        jugador2 = (string)json["jugador2"];

        texto = $"¡Hola " + jugador2 + ", has sido retado por " + jugador1 + "!
¿Serás capáz de superarlo?";
        attachment = GetCard(texto);

        break;
```

Figura 31: Mensaje motivacional tipo 2

Si **Type** es **2**:

El JSON recibido contiene los campos “Tipo”, “jugador1”, “jugador2” y “resultado”. Dependiendo del valor del campo resultado, se notifica al usuario que inició el reto como terminó la partida. El dato **resultado** es 0 si se empata, 1 si el retador ha ganado y 2 si el retador ha perdido.

```
case 2:
        jugador1 = (string)json["jugador1"];
        jugador2 = (string)json["jugador2"];
        resultado = (string)json["resultado"];

        switch (resultado)
        {
                case "0":
                        titulo = $"Has empatado";
```

```
                        texto = $" " + jugador1 + " has empatado con " + jugador2 +
"."; 
                        break;
                case "1":
                        titulo = $"Has ganado";
                        texto = $" " + jugador1 + " has ganado contra " + jugador2 +
".";
                        break;
                case "2":
                        titulo = $"Has perdido";
                        texto = $" " + jugador1 + " has perdido contra " + jugador2 +
".";
                        break;
        }
        attachment = GetCard(titulo, texto);
        break;
```

Figura 32: Mensaje motivacional tipo 3

Una vez que se ha creado el mensaje en formato “Hero Card”. Se procede a crear la actividad que conecta el bot con el canal del usuario (si en la base de datos está registrado con más de un canal, el mensaje se enviará a cada uno de ellos).

```
MicrosoftAppCredentials.TrustServiceUrl(url);
connector = new ConnectorClient(new Uri(url));
ConversationResourceResponse conversation = new ConversationResourceResponse();

//CREACION DE MENSAJE
var sms = Activity.CreateMessageActivity();
sms.Attachments.Add(attachment);
sms.Type = ActivityTypes.Message;
sms.From = botAccount;
sms.Recipient = userAccount;
sms.ChannelId = channelId;

//Envio del mensaje por los diferentes canales
try
{
        switch (channelId)
        {
                case "msteams":
                        conversation =
connector.Conversations.CreateOrGetDirectConversation(botAccount, userAccount,
tenantId);
                        sms.Conversation = new ConversationAccount(id:
conversation.Id);
                        await
connector.Conversations.SendToConversationAsync((Activity)sms);
                        break;
                case "skype":
                        conversation = await
connector.Conversations.CreateDirectConversationAsync(botAccount, userAccount);
```

```
                    sms.Conversation = new ConversationAccount(id:
conversation.Id);
                    await
connector.Conversations.SendToConversationAsync((Activity)sms);
                    break;
             case "telegram":
                    conversation = await
connector.Conversations.CreateDirectConversationAsync(botAccount, userAccount);
                    sms.Conversation = new ConversationAccount(id:
conversation.Id);
                    await
connector.Conversations.SendToConversationAsync((Activity)sms);
                    break;
             case "facebook":
                    conversation = await
connector.Conversations.CreateDirectConversationAsync(botAccount, userAccount);
                    sms.Conversation = new ConversationAccount(id:
conversation.Id);
                    await
connector.Conversations.SendToConversationAsync((Activity)sms);
                    break;
      }
}
```

Figura 33: Creación y envío de la actividad mensaje a los diferentes canales.

10. Conclusiones

10.1. Conclusiones finales

En este proyecto se ha pretendido mostrar una visión global de la creación de un bot empezando por saber qué acciones va a realizar, la toma de decisiones que debemos tomar, desde cómo queremos que se muestre la información cuando un usuario interactúa con él hasta cómo y qué datos se van a almacenar datos del usuario. También el lenguaje que se va a utilizar, el alcance que va a tener nuestro bot, y la plataforma donde se va a desarrollar.

Se ha demostrado el potencial de los bots en la docencia. Con el envío de notificaciones a los participantes, donde se les animaba a seguir jugando o se les avisaba de los retos, se ha conseguido hacer más atractivo el concurso. Además, se demuestra que es posible facilitar la labor del docente con el seguimiento de alumnos o controlar video clases con tan solo una instrucción.

Gracias a la computación en la nube que ofrece Microsoft con Azure se ha evitado el retraso en los mensajes o que estos llegasen en el orden equivocado. Otra de las ventajas de alojar el bot en la nube, es la capacidad que tiene esta de escalabilidad si fuese necesario ante un aumento del flujo de tráfico.

Utilizar Bot Framework de Microsoft ha sido un acierto, consiguiendo con un solo desarrollo que el bot estuviese disponible en diferentes aplicaciones de mensajería.

Bot Framework dispone de un canal dedicado para la comunicación entre bot y aplicaciones externas. Con ese canal, se ha conseguido que el bot no fuese de tipo mensaje-respuesta, consiguiendo comunicarlo con la aplicación web de “Rétame y Aprendo” y dotándolo de la funcionalidad de envío de notificaciones.

10.2. Líneas futuras

Existen unos puntos en los que es posible un cambio en el desarrollo, y esto se debe a que después de la utilización del bot en un caso real como es el programa educativo *“Rétame y aprendo”* se han detectado posibles puntos a mejorar.

Uno de los primeros puntos a modificar en el prototipo de bot es la comprensión del lenguaje. Se podría estudiar y desarrollar el servicio cognitivo que nos ofrece Microsoft (LUIS) y así conseguir una mejor comprensión e interpretación de las necesidades del usuario. Esto se debe a que se vio que los usuarios además de utilizar los comandos de uso intentaban mantener una conversación y el bot solo detectaba ciertas palabras clave.

El desarrollo está destinado a mostrar varias funcionalidades en un solo bot. Separar cada funcionalidad en bots haría más sencillo e intuitivo su uso. En este caso serían tres bots diferentes: para alumnos participantes de “Rétame y Aprendo”, un bot para los docentes, y otro para el control remoto.

Otro posible cambio sería la optimización de la comprensión del bot para identificar comandos. Se considera que la detección de comandos mediante índices fue una solución a la falta de homogeneidad entre canales. Cambiar el formato de los comandos podría ser una solución.

Para otros desarrollos futuros, se podría desarrollar un bot de alerta a docentes con información relativa al seguimiento de los alumnos. Por ejemplo, el docente puede recibir un mensaje cuando el porcentaje de aprobados en una unidad ubicada en la plataforma e-learning esté por debajo del 50%. Pudiendo así, hacer un repaso de dicha unidad en clase.

Creación de un bot donde el alumno puede pedir información sobre unidades del temario (definiciones, conceptos, etc.) y devolver la información o hacer preguntas tipo test que el alumno debe responder. También, se podría diseñar un bot de avisos para el alumno donde se le notificará la subida de una nueva unidad, la fecha de un examen o notas. Envío de documentos con tareas al bot y que este fuese capaz de hacerlo llegar al docente.

10.3. Bibliografía

- Fco. Javier Ceballos Sierra, "*Microsoft C#. Curso de programación"*, Ra-Ma 2011
- Yolanda Cerezo López, Olga Peñalba Rodríguez, Rafael Caballero Roldán, *"Iniciación a la Programación en C#. Un enfoque práctico",* Delta Publicaciones 2007
- Kishore Gaddam, *"Building Bots with Microsoft Bot Framework"*, Packt Publishing 2017

10.4. Recursos Web

- La primera página web del mundo:
  https://www.deutschland.de/es/topic/economia/innovacion-tecnica/la-primera-pagina-web-del-mundo
- ¿Qué es un bot?
  https://es.wikipedia.org/wiki/Bot
- Documentación Azure Bot Service:
  https://docs.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-3.0
- Lenguaje C#:
  https://docs.microsoft.com/es-es/dotnet/framework/get-started/
- ¿Qué es un SDK?:
  https://www.xatakandroid.com/seguridad/que-es-un-sdk-y-por-que-algunos-pueden-suponer-una-amenaza-para-nuestra-privacidad
- Sensor Caliper:
  http://www.imsglobal.org/activity/caliper
- Servicio RESTful:
  https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional
- Servicio de cola de Azure:
  https://docs.microsoft.com/es-es/azure/service-bus-messaging/
- WebJob:
  https://docs.microsoft.com/es-es/azure/app-service/web-sites-create-web-jobs

- Azure Portal:
  https://www.tecon.es/que-es-microsoft-azure-como-funciona/
- PHP:
  http://php.net/manual/es/intro-whatis.php
  https://es.wikipedia.org/wiki/PHP

ANEXO: Instrucciones para la conexión con los diferentes canales

a. Skype

Skype es una aplicación de mensajería instantánea, teléfono y videollamadas que mantiene a los usuarios conectados. Con esta interfaz los usuarios podrán interactuar con bots.

El método para enlazar Skype es muy sencillo, solo es necesario activar el canal. Para distribuir el contacto de nuestro bot a los diferentes usuarios solo hay que facilitar el enlace haciendo clic derecho sobre el canal de Skype, pinchando en "copiar enlace". Ese enlace es el que podemos distribuir a cualquier usuario.

Dentro de la configuración de Skype, encontramos diferentes apartados de configuración:

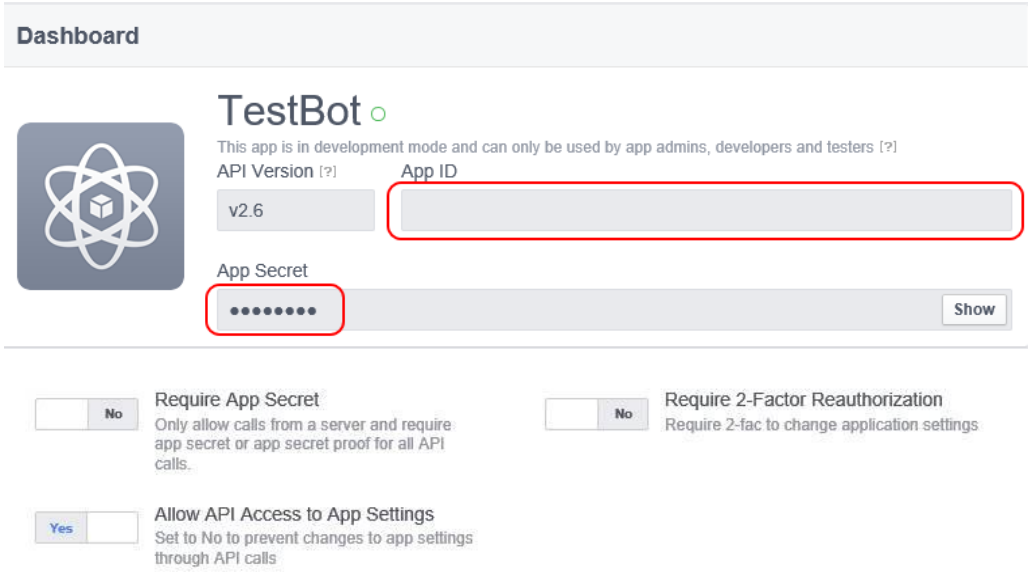
- **Web control:** Para embeber el bot en una web.
- **Messaging:** Para configurar como el bot envía y recibe mensajes en Skype.
- **Calling:** Para añadir funciones en las que el bot pueda llamar.
- **Groups:** Para permitir o no que le bot pueda entrar en grupos de Skype.
- **Publish:** Para permitir que el bot pueda ser utilizado por más de 100 usuarios diferentes. Para ello hay que publicarlo y que el equipo de Skype lo revise.

b. Facebook Messenger

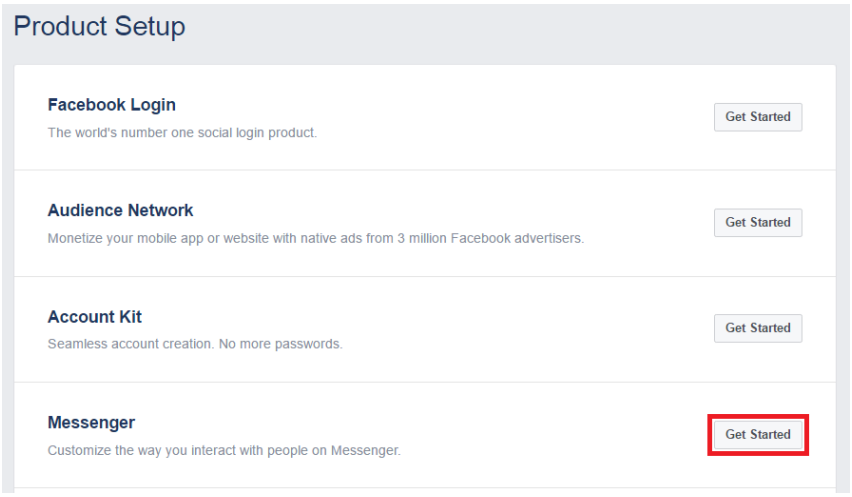
Para poder conectar nuestro bot con Facebook Messenger hay que seguir los siguientes pasos:

1. Crear Página de Facebook y apuntar ID de página.

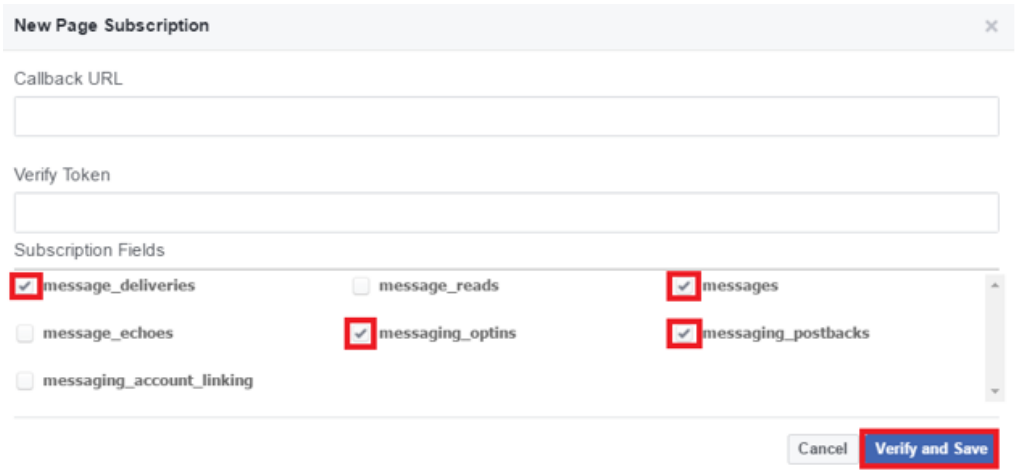
2. Crear “Aplicación de Facebook” entrando como usuario de Facebook en https://developers.facebook.com/
3. Guardar ID y clave secreta de la aplicación.
4. Activar la opción “Permitir el acceso de la API a la configuración de la aplicación” en Configuración/Opciones avanzadas/



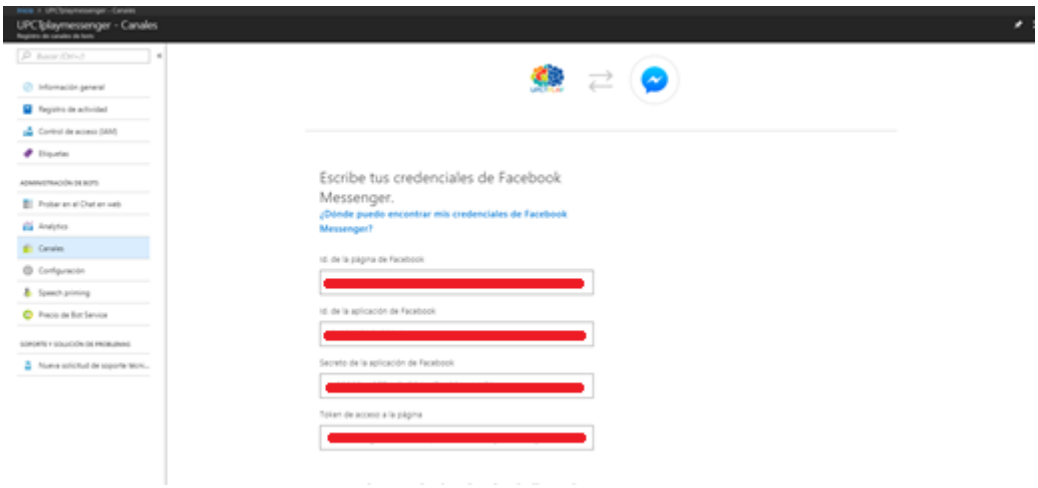
5. Activar Facebook Messenger en nuestra aplicación de Facebook y añadir producto/Messenger



6. Generar un identificador, eligiendo la página a la que lo queremos enlazar y guardar el identificador.
7. Habilitar Webhooks e introducir los datos que Azure nos aporta para conectar con Facebook.
8. Marcar las casillas de “message_deliveries”, “messages”, “messaging_options”, and “messaging_postbacks”.



9. Solicitar permisos a través del panel de configuración Para poder enviar y recibir mensajes mediante una página de Facebook. Los permisos necesarios son “pages_messaging”, “public_profile” y “user_friends”.
10. Introducir los datos que se piden en el Portal de Azure.



c. Skype for Business

Skype for Business es igual de sencillo de habilitar que Skype, solo debemos hacer clic en el icono de la aplicación y ya estará habilitado. La única diferencia entre Skype y Skype for Business es que el primero está orientado a pequeñas empresas o particulares y el segundo orientado a empresas más grandes. Permitiendo agregar 250 personas a reuniones en línea y ofrece seguridad de nivel empresarial.

d. Microsoft Teams

Esta aplicación sigue la misma mecánica de activación que Skype y Skype for Business simplemente es pulsar en el icono y el canal queda habilitado.

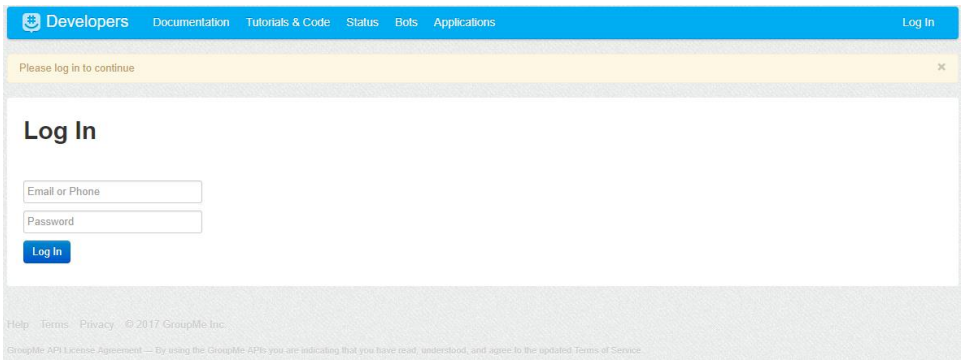
e. Email

Este canal solo lo podremos utilizar con una cuenta de correo de Office 365, los únicos datos necesario para enlazar nuestro bot es facilitar la cuenta de correo y la contraseña.

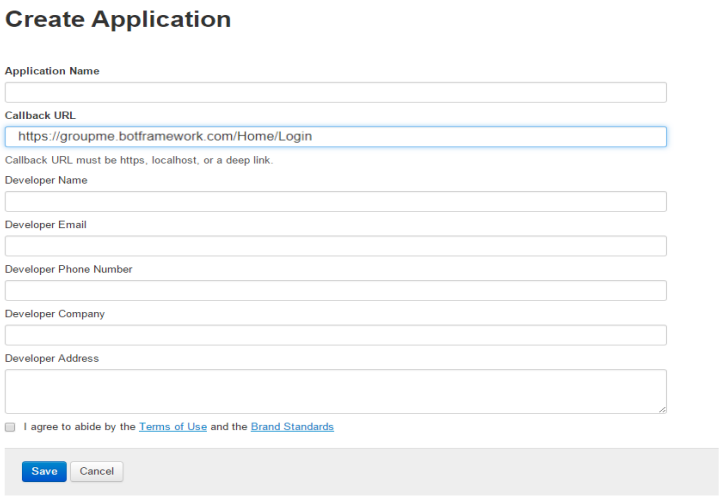
f. GroupMe

GroupMe es una App de mensajería instantánea para Smartphone de Microsoft. Para enlazar este canal a nuestro bot:

1. Crear una cuenta en GroupMe.
2. Crear una aplicación de GroupMe (https://dev.groupme.com/session/new) entrando con nuestra cuenta de usuario.

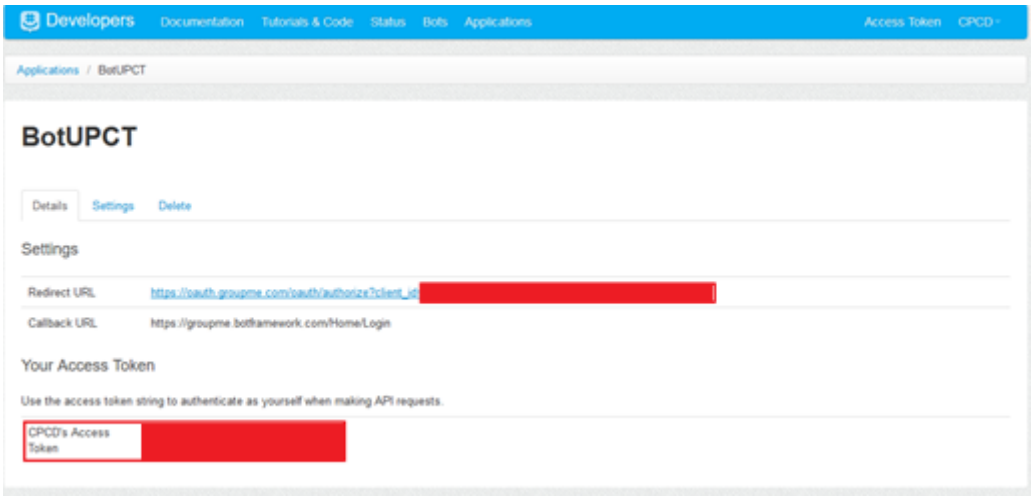


3. Crear la aplicación completando los campos de la siguiente imagen.



Una vez creada, en el panel de la aplicación nos aparece la información necesaria para enlazar en Bot Framework.

4. Copiar la clave que se encuentra detrás de “client_id=” en el campo “Redirect URL” y el Token del campo “Access Token” marcados en la imagen que se muestra a continuación y pegar esos datos en Bot Framework.

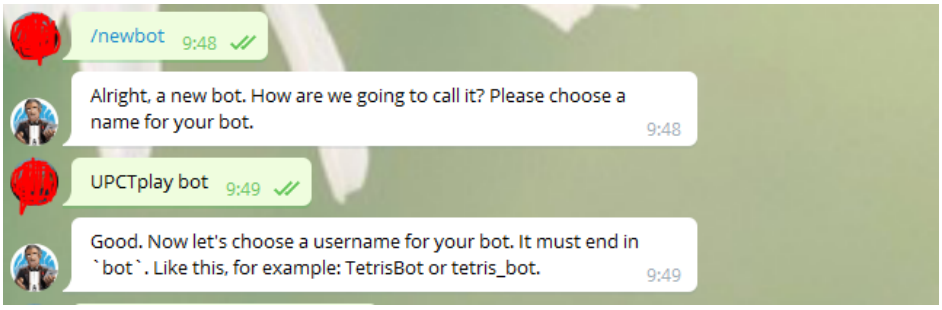


g. Telegram

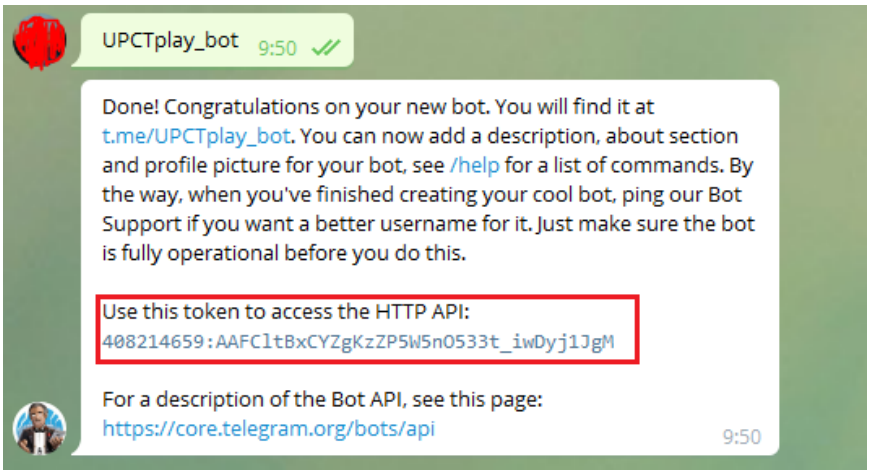
Telegram es una aplicación de mensajería enfocada al envío y recepción de mensajes de texto y multimedia. Inicialmente el servicio fue empleado para teléfonos móviles y luego dio el salto a multiplataforma.

Podemos configurar nuestro bot para que se comunique con el usuario que use esta aplicación. Para ello se debe:

1. Ser usuarios de esta aplicación de mensajería.
2. Buscar BotFather (bot creador de bots) en el buscador.
3. Con el comando "/newbot" empezar el procedimiento de creación de bot. Nos pedirá que introduzcamos el nombre del bot y seguidamente un nombre de usuario.



4. Nos devolverá un mensaje de bot creado con un token de acceso.



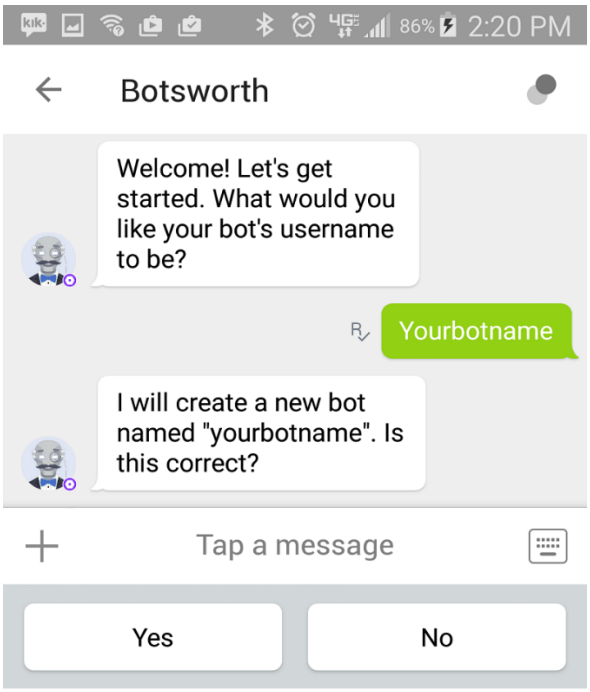
5. Copiar el token en el apartado correspondiente de Bot Framework.

h. Kik

Kik es una aplicación de mensajería instantánea gratuita para dispositivos móviles. Es similar al resto de aplicaciones conocidas en las que los usuarios pueden compartir mensajes, fotos, mensajes de voz y otros contenidos.

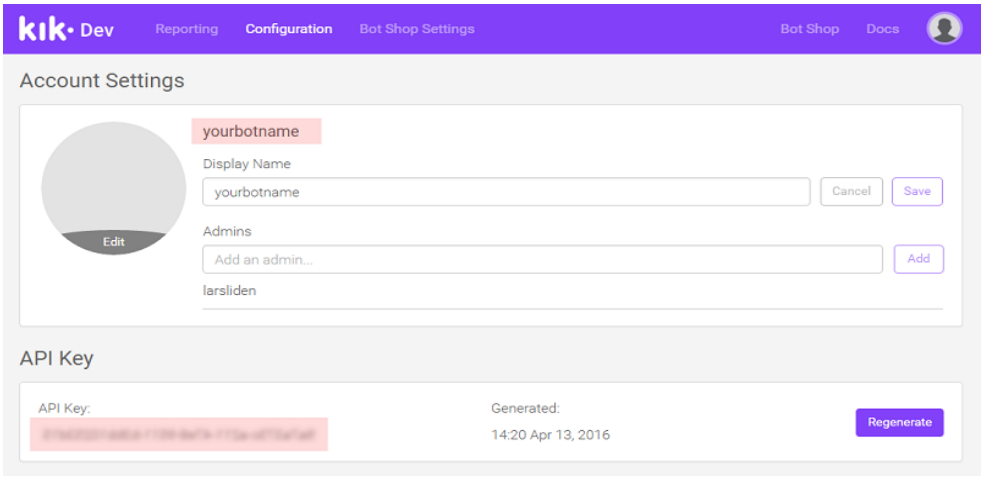
Para que nuestro bot pueda interactuar con los usuarios de Kik:

1. Crear con nuestro móvil una cuenta,
2. Acceder a la web de desarrolladores (https://dev.kik.com) con el usuario creado.
3. Escanear con la aplicación móvil de Kik la imagen que se muestra para abrir una conversación con el asistente de bots.



El asistente nos dará la bienvenida y nos preguntará por el nombre de nuestro bot que vamos a crear. Una vez creado, tendremos la opción de entrar a la configuración del bot creado.

4. Copiar el nombre del bot y “API key” y pegar en el apartado correspondiente de Bot Framework.

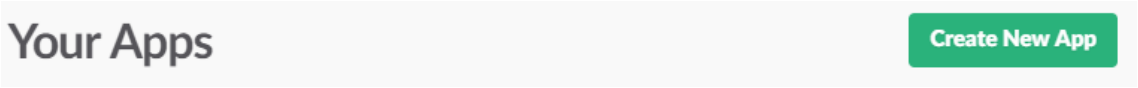


i. Slack

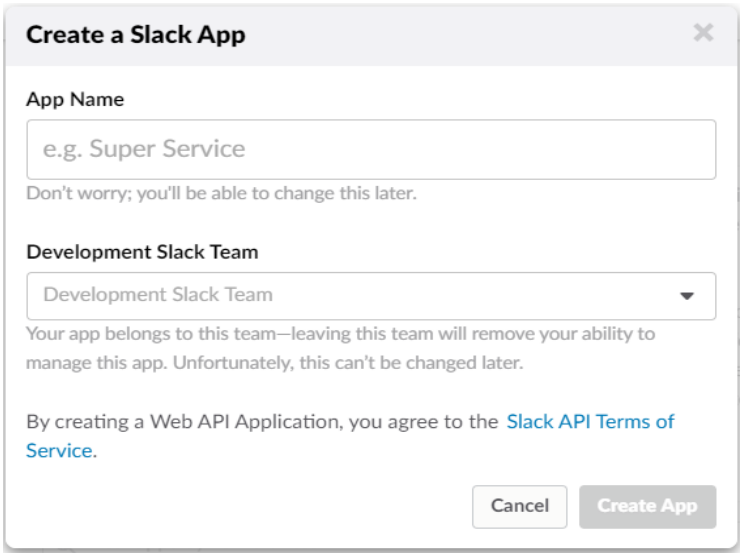
Herramienta de comunicación en equipo. Ofrece salas de chat organizadas por temas, así como grupos privados y mensajes directos e integra gran cantidad de servicios a terceros.

Para enlazar nuestro bot a Slack debemos tener una cuenta. Acceder a https://api.slack.com/apps y nos identificamos con usuario y contraseña de nuestra cuenta, una vez dentro:

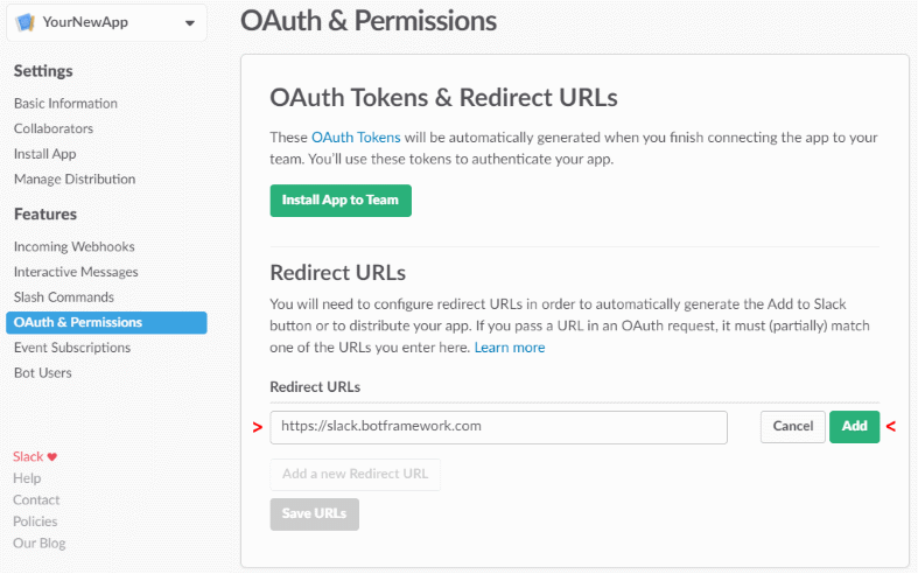
1. Pulsar “Create New App”.



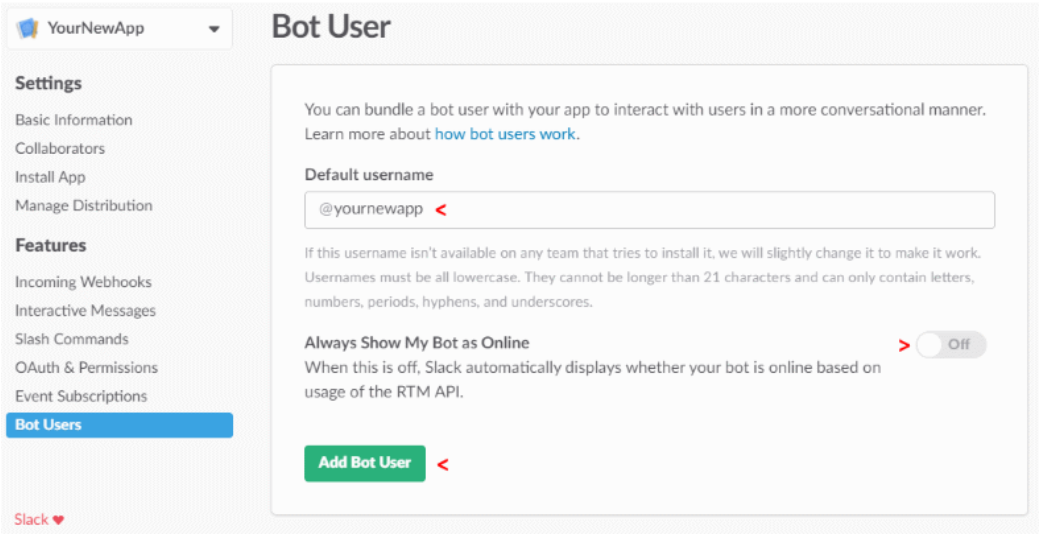
2. Introducir el nombre de la nueva App donde enlazaremos el bot y seleccionar el equipo de desarrollo de Slack al que pertenecerá. Cuando esté creada generará un Client ID y Client Secret.



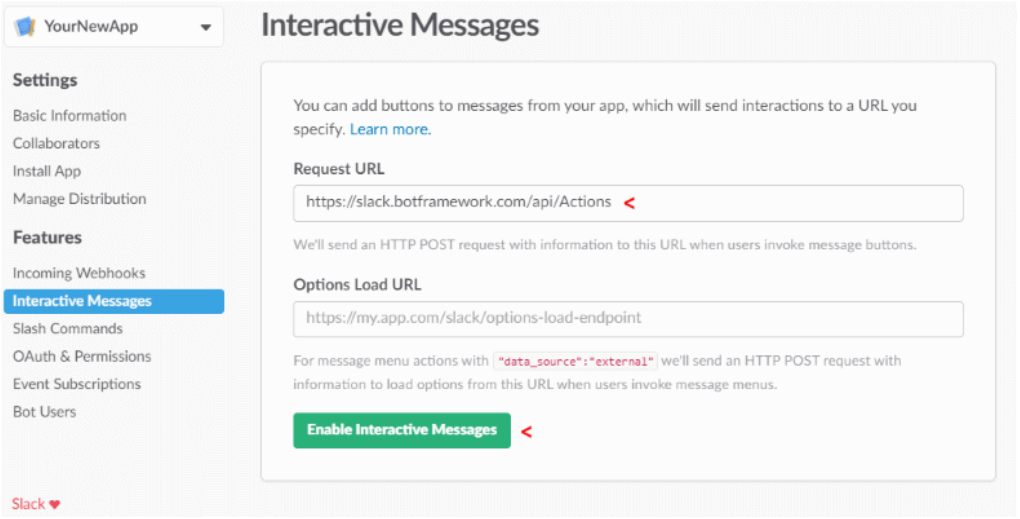
3. Añadir la URL de Redirección a Bot Framework, para ello, seleccionamos “OAuth & Permissions”, hacemos clic en “Add a new Redirect URL”, introducimos en el campo la URL (https://slack.botframework.com), le damos al botón de añadir y por último Save URLs.



4. Crear un usuario de Slack, accediendo a la pestaña “Bot Users” y haciendo clic en Add a Bot User. Añadimos la opción “Always Show My Bot as Online” y hacemos clic en “Add Bot User” y “Save Changes”.



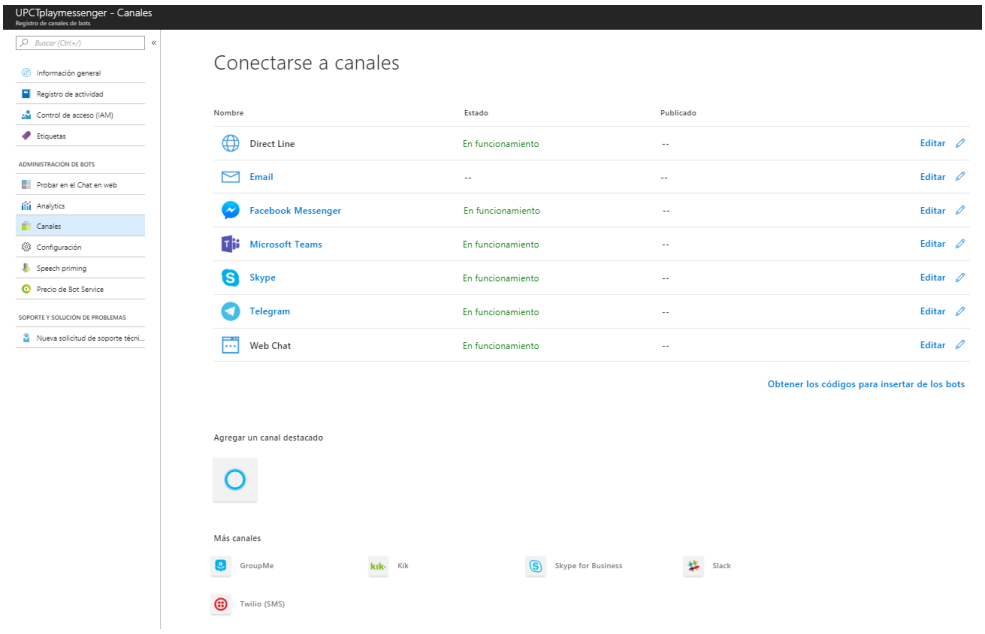
5. Añadir y configurar mensajes interactivos, en la pestaña “Interactive Messages” añadiendo la URL https://slack.botframework.com/api/Actions en el apartado “Request URL” y pulsar “Enable Interactive Messages” para guardar los cambios.



Por último en Basic Information copiamos el Client ID, Client Secret y el verificationToken y los pegaremos en el apartado de slack de Bot Framework portal.

j. Direct Line

Direct Line nos ofrece la opción de comunicar nuestro bot con una aplicación externa. Para añadir el canal Direct Line pulsamos en el botón de Direct Line.



A continuación, agregamos un nuevo sitio que representará la aplicación cliente externa que deseamos conectar a nuestro bot. Bot Framework genera unas claves secretas que deberá usar la aplicación cliente para autenticar las solicitudes de Direct Line para comunicarse con nuestro bot.