

Desarrollo de nuevos modelos de interacción usuario-ecommerce.

Integración de ecommerce en chatbot vía API REST

Memoria de Proyecto Final de Máster
Máster Universitario en Ingeniería Informática
Desarrollo de aplicaciones web

Autor: Rubén Rodríguez Paz

Consultor: Joan Giner Miguelez
Profesor: David García Solórzano

6 de enero de 2020

Créditos



Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-SinObraDerivada

[3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

“La mejor forma de predecir el futuro es implementarlo.”

David Heinemeier Hansson

Abstract

El uso de asistentes virtuales mediante mensajería o chatbots se ha popularizado en los últimos años debido a la irrupción de nuevos modelos de inteligencia artificial y el abaratamiento de los sistemas. Estos son utilizados para la resolución de problemas, informes de estado, triaje de usuarios, realización de tareas automatizadas e integración de acciones entre varios sistemas en los que el chatbot sirve como nexo. Precisamente entre estos usos, los chatbots están presentes en plataformas de comercio electrónico para dar soporte a los usuarios, realizar pedidos, etc.

Mediante este trabajo se desarrolla un chatbot que permite realizar la búsqueda, selección de productos y posterior compra en un ecommerce bajo la plataforma Magento 2. Para ello se hará uso de su API REST y se integrará en una aplicación NodeJS en la que se usará Microsoft Bot Framework como plataforma para la creación de este asistente virtual.

Palabras clave: ecommerce, comercio electrónico, chatbot, Magento, Microsoft Bot Framework, JavaScript.

Abstract (english version)

The use of messaging virtual assistants through or chatbots has become popular in recent years due to the emergence of new artificial intelligence models and the lower price of computing systems. These are used for troubleshooting, status reports, user triage, performing automated tasks and integrating actions between several systems in which the chatbot serves as a link. Precisely among these uses, chatbots are present in ecommerce platforms to support users, place orders, etc.

Through this project, a chatbot is developed that allows you to search, select products and later purchase in an ecommerce under the Magento 2 platform. To do this, you will use your REST API and integrate it into a NodeJS application in which Microsoft will be used Bot Framework as a platform for the creation of this virtual assistant.

Keywords: ecommerce, chatbot, Magento, Microsoft Bot Framework, JavaScript.

*A Laura Medel,
por animarme a comenzar esta aventura;
y por la ayuda, el aliento, la paciencia y su saber para terminarla.*

Índice

1. Introducción	11
2. Descripción	12
3. Objetivos	13
Principales	13
Secundarios	13
4. Marco teórico	14
Tipos de chatbot por usos y objetivos	16
Humano - Chatbot	16
Humano - Chatbot - Humano (fallback)	17
Humano - Chatbot - Humano	17
Tipos de chatbots por interacción con el usuario	17
Chatbots basados en menú-botones	17
Chatbots basados en el reconocimiento de palabras clave	17
Chatbots contextuales	17
5. Contenidos	18
6. Metodología	19
Dimensionamiento del proyecto	19
Planificación del proyecto	20
Instanciación de la plataforma de ecommerce	20
Creación del chatbot	20
Implementación de la lógica de negocio del proyecto	20
Pruebas	21
Exportación del chatbot a aplicaciones de chat	21
7. Arquitectura de la aplicación	22
Servidor	23
Cliente	24
8. Plataforma de desarrollo	25
Software	25
Magento 2	25
Microsoft Bot Framework	25
NodeJS	25
GitHub	26
Hardware	26
Digital Ocean	26
Microsoft Azure	26
9. Planificación	27
Hitos	27
Diagrama de Gantt	28

10. Proceso de desarrollo	29
Creación de la instancia de Magento 2	29
Instalación de PHP y complementos	29
Instalación de Apache	29
Instalación y configuración de MySQL	29
Creación de un proyecto de Magento 2	30
Configuración de Magento 2	31
Creación del chatbot	32
Desarrollo del chatbot	32
Figuras clave del desarrollo	33
Estado del chatbot	33
Perfil de usuario	33
Datos de conversación	34
Estructura del proyecto	34
package.json	35
index.js	35
bot.js	35
Flujo de ejecución	36
Gestión de estados en la función onMessage	37
Diálogos	37
Depuración	39
11. APIs utilizadas	41
/products/product/media	42
/products?searchCriteria=	42
/guest-carts	42
/guest-carts/cartId/items	42
/guest-carts/cartId/totals/	43
/guest-carts/cartId/shipping-information	43
/guest-carts/cartId/payment-information	43
12. Diagramas	44
13. Perfiles de usuario	45
Usuarios invitados	45
Usuarios registrados	45
14. Usabilidad/UX	46
Inserción de texto	47
Elección	47
15. Seguridad	48
Infraestructura de Magento 2	48
Magento 2	48
Infraestructura de MageChatbot	48
MageChatbot	48

16. Tests	50
17. Requisitos de instalación	52
18. Instrucciones de instalación/implantación	53
Configuración y creación de la infraestructura en Microsoft Azure	53
Conexión de MageChatbot con Telegram	56
19. Instrucciones de uso	58
20. Bugs	62
Imágenes en producción	62
POST to Bot timeout after 15s en Telegram	62
Caducidad de las API keys de Magento	62
21. Proyección a futuro	63
22. Conclusiones	64
23. Bibliografía	65
24. Glosario	71

Figuras y tablas

Índice de figuras

Figura 1: Resultados de la encuesta ante la pregunta “¿Se ha comunicado con negocios en los últimos 12 meses? (Audience, Drift, MyClever & Salesforce, 2018)	15
Figura 2: Resultados de la encuesta ante la pregunta respecto a los problemas con problemas con experiencias online (Audience, Drift, MyClever & Salesforce, 2018)	16
Figura 3: Esquema de la arquitectura propuesta. Elaboración propia.	23
Figura 4: Diagrama de Gantt con la planificación del proyecto. Elaboración propia.	28
Figura 5: Home de http://tfm.rubenr.es	30
Figura 6: Listado de productos de http://tfm.rubenr.es	31
Figura 7: Detalle de la ventana de integraciones en el área de administración de Magento 2	31
Figura 8: Estructura de ficheros del proyecto MageChatbot.	35
Figura 9: Ejemplo de mensaje condicional.	37
Figura 10: Ejemplo de mensaje general.	37
Figura 11: Flujo de ejemplo del proceso de checkout.	38
Figura 12: Ejemplo de log de debug en la consola de la aplicación.	39
Figura 13: Diagrama de estados de la aplicación. Elaboración propia.	44
Figura 14: Ejemplos de conversaciones en diferentes aplicaciones de mensajería (WhatsApp, Telegram, Facebook Messenger). (Facebook, 2019. Telegram, 2019)	46
Figura 15: Ejemplo de chatbot en Telegram. (Politibot, 2019)	47
Figura 16: Proceso de test y depuración en microsoft bot framework.	50
Figura 17: Listado de servicios configurados en Microsoft Azure.	51
Figura 18: Ejecución en consola de un despliegue de MageChatbot.	52
Figura 19: Ejemplo de creación de un chatbot para Telegram con BotFather.	53
Figura 20: Pantalla de configuración de integración con Telegram en Microsoft Azure.	53
Figura 21: Ejemplos de conversación inicial de MageChatbot en Telegram.	54
Figura 22: Ejemplo de conversación inicial de MageChatbot en el cliente web.	55
Figura 23: Ejemplo de información y resumen de MageChatbot en el cliente web.	56
Figura 24: Ejemplo del resumen del pedido de MageChatbot en el cliente web.	57
Figura 25: Resumen del pedido realizado con MageChatbot	60

Índice de tablas

Tabla 1: Tabla de hitos del proyecto.	26
---------------------------------------	----

1. Introducción

El concepto de asistente personal ha estado ligado históricamente a la inteligencia artificial (Göksel & Mutlu, 2016). Estos agentes de software ayudan a los usuarios de ordenadores o sistemas de computación a automatizar tareas y realizarlas con la mínima interacción entre hombre y máquina. En los últimos años, con la irrupción de los nuevos modelos de inteligencia artificial, la mejora en el rendimiento de los sistemas y el abaratamiento de su coste se ha popularizado un sistema de asistencia digital de tipo conversacional: los chatbots. Este tipo de asistentes realizan tareas en diferentes ámbitos, de modo automatizado, simulando una conversación con una persona y haciendo más natural la operación con sistemas informáticos. Los chatbots son utilizados en todo tipo de ámbitos para la resolución de problemas, informes de estado, triaje de usuarios, realización de tareas automatizadas e integración de acciones entre varios sistemas utilizando el chatbot como nexo entre estos (Nelson, 2017).

En paralelo al desarrollo de los chatbots, otra tecnología ha tomado gran relevancia tanto en la vida de los usuarios de internet como en la comunidad de desarrolladores: el comercio a través de internet (ecommerce) (Moore, 2018). El ecommerce es la compra y venta de bienes o servicios a través de internet. Dentro su paraguas se engloba desde las compras a través del móvil o el cifrado de pagos, hasta todo tipo de sistemas para compradores y vendedores en línea emulando, en muchos casos, procedimientos que se realizan en la adquisición de bienes o servicios como si de un medio físico se tratara. Con el desarrollo del ecommerce y el paso de los años se ha refinado una arquitectura de la información y una serie de procesos unificados que permiten al usuario comprar en diferentes ecommerce de modo natural y sin tener que aprender a utilizar la plataforma (Corrigan, 2019). La integración de chatbots en ecommerce permite cubrir algunas de las áreas que la plataforma no cubre como puede ser la resolución de problemas o la provisión de información sobre el catálogo ofertado, pero también se pueden integrar para trasladar la experiencia de compra al propio chatbot (Gupta & Borkar & De Mello & Patil, 2015).

Mediante este trabajo se desarrolla un chatbot que permite realizar la búsqueda, selección de productos y posterior compra en un ecommerce bajo la plataforma Magento 2. Para ello se hará uso de su API REST y se integrará en una aplicación NodeJS en la que se usará Microsoft Bot Framework como plataforma para la creación de este asistente virtual.

2. Descripción

Hoy en día, las plataformas de ecommerce contienen una amplia gama de productos y categorías en las que la información de los productos está repartida entre múltiples páginas. A su vez, cuentan con otro tipo de funcionalidades como listas de deseos, recomendaciones, procedimientos de venta cruzada y productos que pueden ser configurables en tiempo real. Esto ha hecho que navegar por algunas de ellas con el objetivo de obtener productos relevantes pueda ser un proceso lento y no intuitivo (Walde, 2018). Un usuario puede acceder a un ecommerce para buscar un producto concreto, navegar por categorías y subcategorías. Para ello, los ecommerce proveen de sistemas de búsqueda que filtran los resultados del catálogo o bien por palabras clave o por un sistema de atributos y pesos. En este último caso no todos los usuarios obtienen resultados relevantes y concluyentes (Sambhanthan & Good, 2013), resultando para el usuario un proceso poco intuitivo o complejo. Esta complejidad también se ve reflejada en la accesibilidad de los ecommerce, que en muchos casos no permite obtener resultados concluyentes para personas con problemas de accesibilidad (Sohaib & Kang, 2017).

A través de este TFM se desarrolla un chatbot que permite interactuar con un ecommerce mediante el uso de las API que provee esta plataforma. Este chatbot permite, mediante una conversación, simplificar la navegación del usuario por el abanico de páginas y acciones, así como listar productos y permitir su compra. A su vez, permite al usuario interactuar con este chatbot desde plataformas de mensajería instantánea (como WhatsApp o Telegram) y no desde el propio ecommerce.

Para ello, previamente se ha definido el tipo de tecnologías utilizadas tanto para la ejecución del ecommerce, como para el desarrollo del chatbot y la conexión entre los dos sistemas. En el lado del ecommerce se ha utilizado como plataforma Magento 2, una de las plataformas de comercio electrónico más usadas en la actualidad que destaca por su robustez y potencia de cara al desarrollo (Shadid, 2018).

Por otra parte, se ha definido cómo se va a desarrollar el chatbot y qué lenguajes o frameworks utiliza. En este caso ha utilizado Microsoft Bot Framework como plataforma de desarrollo. Un framework de desarrollo de chatbots para NodeJS con posibilidad de publicación del servicio en diferentes aplicaciones de mensajería instantánea mediante la plataforma de servicios de Microsoft, Microsoft Azure (Microsoft, s.f.g).

3. Objetivos

A través de este TFM se pretenden alcanzar los siguientes objetivos tanto principales como secundarios.

Principales

- Implementar un nuevo modelo de interacción entre usuario y ecommerce a través del desarrollo de un chatbot que pueda conectarse a una plataforma de ecommerce vía API.
- Facilitar la realización de diferentes acciones por parte de usuario a través de un sistema en el que el chatbot permita ejecutar un flujo básico de compra en el ecommerce (búsqueda, detalle, carrito y compra).

Secundarios

- Permitir que el código sea ejecutado en el entorno de producción a través de la implementación de la infraestructura necesaria.
- Facilitar el acceso al chatbot mediante un cliente web, para que este sea accesible para el usuario.
- Facilitar el uso del ecommerce a través de otros canales, posibilitando el acceso al chatbot vía plataformas de mensajería instantánea (WhatsApp y Telegram).

4. Marco teórico

Para hablar de chatbots hay que remontarse a los inicios de la inteligencia artificial en los que Alan Turing definió el juego de “imitación” al detallar el funcionamiento del test conocido posteriormente como el Test de Turing (Turing, 1950). Este juego pretendía determinar si un ordenador podía imitar el comportamiento humano hasta el punto de no ser diferenciado de un humano en un test doble ciego (Almanson, Ebtesam & Hussain, Farookh, 2019).

El primer chatbot, llamado ELIZA, fue desarrollado en 1966 (Yúbal, 2017). Este sistema mantenía una conversación con los usuarios a través de un terminal. Aunque era un sistema bastante complejo, se basaba en estrategia de devolver las preguntas recibidas como otra pregunta. Este comportamiento se mantiene hoy en día cuando un chatbot no consigue “entender” lo que el usuario dice, hasta que la respuesta del usuario puede ser identificada dentro del mapa de conocimiento del robot de conversación. En los años posteriores se desarrollaron múltiples tecnologías que dotaron a los chatbots de gran potencia. Entre ellas destacan el *Semantic Analysis*, *Sentiment Analysis*, *Natural Language Processing (NLP)*, *Natural Language Understanding (NLU)* o el Machine Learning.

En 2018, empresas como Drift, SurveyMonkey, Salesforce, Audience y Myclever realizaron un estudio sobre el uso de chatbots en la web (Audience, Drift, MyClever & Salesforce, 2018). En este estudio realizaron una serie de preguntas a 1051 personas de entre 18 y 64 años en Estados Unidos con el objetivo de conocer la situación de los chatbots en el mercado, su uso y proyección futura. Esta encuesta abría con la siguiente pregunta: “¿Cómo te has comunicado con empresas en el último año?”

Business Communication Channels

How have you communicated with businesses in the past 12 months?

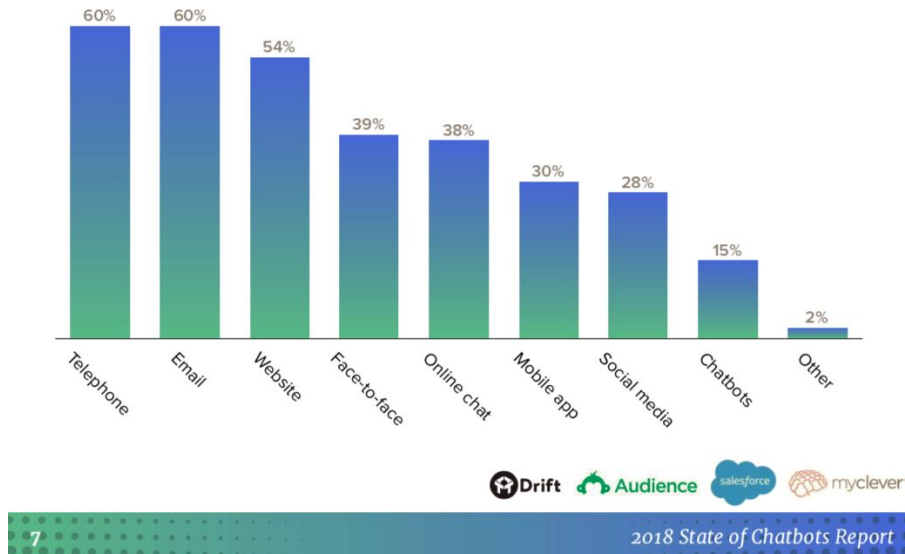


Figura 1: Resultados de la encuesta ante la pregunta "¿Se ha comunicado con negocios en los últimos 12 meses? (Audience, Drift, MyClever & Salesforce, 2018)

En la figura 1 se puede ver como el 15% de los encuestados ha utilizado chatbots en el último año y el 38% un chat online, que también puede contener un chatbot. Si se tiene en cuenta que se puede establecer conversaciones con medios de mensajería en teléfonos hace que se pueda inferir que uso de los chatbots puede estar por encima del 50% (Drift, 2018). Con esta información se puede ver como las comunicaciones con chatbots van en aumento, por lo que su uso e implementación tiene sentido si solucionan problemas reales. Este estudio dirige las preguntas hacia ese tipo de análisis, las relaciona con las dificultades en su uso y las preferencias y ventajas que más valoran los encuestados.

Ante la pregunta "¿Qué frustraciones has experimentado en el último año?", el 34% de los encuestados determina que las webs son difíciles de navegar. Por la tipología de web, las tiendas online o e-commerce entran dentro de las de mayor complejidad (Cowderoy, 2001).

Problems With Traditional Online Experiences

What frustrations have you experienced in the past month?

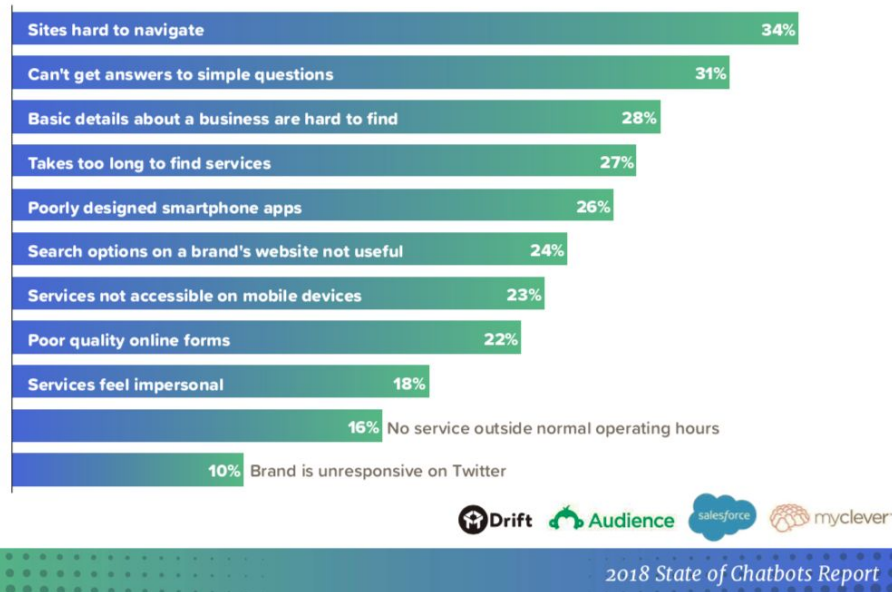


Figura 2: Resultados de la encuesta ante la pregunta respecto a los problemas con experiencias online (Audience, Drift, MyClever & Salesforce, 2018)

Teniendo en cuenta este tipo de problemas y las frustraciones de los usuarios, el uso de los chatbots se puede enmarcar dentro de los siguientes casos (Drift, 2018):

- Obtener una respuesta rápida.
- Resolución de reclamaciones y problemas.
- Obtener información más detallada.
- Reserva de servicios.
- Pago de servicios.
- Compra de productos o servicios.

Tipos de chatbot por usos y objetivos

Teniendo en cuenta los casos de uso posibles de un chatbot y su objetivo podemos dividir las soluciones que implementan chatbots en los siguientes tipos (Devaney, 2016):

Humano - Chatbot

Este tipo de soluciones se caracterizan por la falta de participación de otro humano. O bien aprovechan el uso de sistemas de procesamiento de lenguaje natural (NPL) para resolver preguntas, o bien la interacción

con el chatbot como medio para automatizar procesos. En cualquier caso, el éxito de este tipo de sistemas depende del dominio del tema y la capacidad de aprendizaje del chatbot.

Humano - Chatbot - Humano (fallback)

Este caso es similar al anterior pero con la diferencia de que la conversación se dirige a un humano cuando el chatbot llega a un punto de no retorno o no tiene capacidad de actuar ante las preguntas del usuario. En ese punto, un humano retoma la conversación para asistir al usuario.

Humano - Chatbot - Humano

Este tipo de chatbots se utilizan cuando no se pretende reemplazar a un humano para interactuar con los usuarios, si no como un medio de categorización o ayuda en primera instancia. Estos chatbots permiten recolectar la información previa necesaria para gestionar la interacción o categorizar al usuario para dirigirlo al humano responsable de responder la consulta o atender su petición.

Tipos de chatbots por interacción con el usuario

Si se tiene en cuenta la forma de interactuar con el usuario, los chatbots pueden dividirse en 3 categorías (Phillips, 2018):

Chatbots basados en menú-botones

Es el caso más común en el mercado y están basados en un listado de acciones predeterminadas donde el usuario puede elegir entre ellas, recorriendo un árbol de decisión o un diagrama de flujo. Son suficientes para manejar un gran tipo de procesos pero, no cuentan con NLP por lo que no son óptimos para predecir o optimizar los recorridos del árbol o diagrama.

Chatbots basados en el reconocimiento de palabras clave

A diferencia de los chatbots basados en menú-botones, estos chatbots integran inteligencia artificial (AI) para determinar cómo responder de modo correcto al usuario. Por ello, pueden reconocer lo que usuario escribe y responder apropiadamente (o al menos intentarlo).

Chatbots contextuales

Este tipo de chatbots son los más complejos de todos, ya que integran Machine Learning e inteligencia artificial (AI). Permiten recordar conversaciones con un usuario específico y aprender con el paso del tiempo. A diferencia de los chatbots basados en el reconocimiento de palabras clave estos pueden mejorar progresivamente y aprender de lo que el usuario dice, por lo que una previa no comprendida puede aprenderse para ocasiones futuras.

5. Contenidos

Atendiendo al contexto definido previamente y a los objetivos fijados para ese proyecto, la solución propuesta está basada en una aplicación en NodeJS que permite implementar un chatbot usando Microsoft Bot Framework. Esta aplicación permite conectarse mediante una API REST a un ecommerce con Magento 2 y realizar el proceso básico de compra para usuarios anónimos. Para ello ha sido necesario:

- Desplegar una instancia de Magento 2 en la nube
- Diseñar una arquitectura mediante la cual se pueda configurar la instancia de Magento 2 que emplea el chatbot mediante un fichero de configuración.
- Proveer de los métodos necesarios para operar con la API REST mediante la aplicación en NodeJS. Para ello el diseño es fácilmente extensible y utiliza los principios de modularidad propia de cualquier aplicación orientada a objetos.

Debido al carácter piloto del proyecto se ha trabajado como si se estuviera desarrollando un MVP (mínimo producto viable). La aplicación atiende a los objetivos mínimos fijados en la propuesta del proyecto pero ha sido diseñada para cumplir con todas las funcionalidades descritas en cualquier ecommerce convencional. Para cumplir con los objetivos del proyecto, la aplicación permite:

- Realizar búsquedas de productos en el ecommerce.
- Elegir entre los productos de la tienda y agregarlos al *carrito de la compra*.
- Realizar la compra de dichos productos.

A su vez, la aplicación provee de los mecanismos necesarios para que el usuario pueda conocer el estado de su compra, mostrando un resumen y la posibilidad de ofrecer información de ayuda en el caso en el que el usuario se encuentre perdido o no sepa como utilizar dicha aplicación.

Finalmente, y utilizando las herramientas provistas por Microsoft Azure, la aplicación se ha desplegado en la nube. Para ello ha sido necesario configurar una serie de recursos que permiten ejecutar el chatbot en los sistemas de Microsoft Azure. Mediante el uso del asistente de Microsoft azure se conecta con clientes de mensajería como WhatsApp o Telegram y con un cliente web.

6. Metodología

Como se ha mencionado previamente, el objetivo de este TFM consiste en el desarrollo de una aplicación que implemente un chatbot que permita conectarse con una plataforma de ecommerce Magento 2 y, mediante el chatbot, permitir a un usuario interactuar con el ecommerce de un modo conversacional.

Para responder a la definición del problema y a los objetivos propuestos, el proyecto responde a las siguientes etapas:

- Dimensionamiento preciso del proyecto
- Planificación del proyecto
- Instanciación de la plataforma de ecommerce
- Creación del chatbot
- Implementación de la lógica de negocio del proyecto
- Pruebas
- Exportación del chatbot a aplicaciones de chat

Dimensionamiento del proyecto

El primer paso para realizar el proyecto ha sido su dimensionamiento de forma precisa. El proyecto está formado por las siguientes tareas:

- Creación de servidor VPS y configuración para cumplir con los requisitos de instalación de Magento 2
- Instalación de Magento 2 en dicho servidor, junto con datos de prueba
- Creación de una aplicación en NodeJS e integración de Microsoft Bot Framework
- Desarrollo o importación de conector API REST con Magento 2
- Desarrollo de lógica de negocio de la aplicación. La aplicación debe ser capaz de buscar en el catálogo de Magento 2, añadir al carrito y realizar el proceso de compra
- Creación de infraestructura necesaria para alojar el chatbot
- Creación de cliente web para poder testear el funcionamiento
- Configuración de bot en al menos una plataforma de mensajería y configuración de la misma para habilitar el funcionamiento con el chatbot previamente desarrollado

Hay una serie de acciones que se excluyen del desarrollo, ya que el objetivo del proyecto es desarrollar un MVP y no la funcionalidad completa:

- Integración o creación de sistema de reconocimiento del lenguaje mediante AI
- Desarrollo de arquitectura escalable tanto para el ecommerce como para el chatbot
- Creación de arquitectura con certificado HTTPS
- Integración de Machine Learning en el chatbot

Planificación del proyecto

El siguiente paso, una vez definido el alcance el proyecto previamente ha sido organizada en hitos de entrega mediante un diagrama de Gantt como se desarrolla posteriormente en el apartado 9.

Instanciación de la plataforma de ecommerce

Tras la planificación, una de las etapas e hitos principales ha sido la creación de una instancia de Magento 2 con la que interactúa el bot. Para ello, se ha hecho uso de un VPS alojado en Digital Ocean. Esta instancia en la nube contiene una configuración básica LAMP con la que se ha instalado Magento 2 posteriormente. Tras la instalación del mismo, se ha creado el dominio `tfm.rubentr.es` y se ha apuntado a dicha instancia del servidor para un fácil acceso desde el navegador.

Creación del chatbot

El siguiente paso ha sido la creación del bot. Para ello, se ha desarrollado una aplicación basada en NodeJS, se ha definido la arquitectura necesaria para el desarrollo, se han importado los módulos necesarios para operar con Microsoft Bot Framework y se han fijado las reglas para que el código sea legible, ordenado y consistente a lo largo del proyecto.

Implementación de la lógica de negocio del proyecto

Una vez definido el stack de trabajo con una arquitectura sólida, ha sido el momento de comenzar el desarrollo de la lógica de negocio. En primer lugar, la aplicación debía ser capaz de conectarse con la instancia de Magento 2 creada previamente y hacer uso de los diferentes *endpoints* que provee. En segundo lugar, se han ido desarrollando las funcionalidades para llevar a cabo todas las posibilidades en el flujo de conversación que se ha definido posteriormente.

Pruebas

Tras el desarrollo de lógica de negocio se ha realizado un proceso completo y detallado de testing. Para ello se ha utilizado tanto Bot Framework Emulator como Mocha¹ como plataforma de testing. Para facilitar las pruebas y mantener un código consistente y bien formado, se ha hecho uso de ESLint² con el fin de identificar patrones problemáticos.

Exportación del chatbot a aplicaciones de chat

Tras la validación de las funcionalidades y objetivos de la aplicación, se ha publicado el chatbot en servicios de mensajería instantánea: Telegram y WhatsApp. Para ello se ha hecho uso de Microsoft Azure, ya que provee de un sistema de publicación sencillo y cubre las necesidades de este proyecto.

¹ Mocha (s.f.). Página Web Corporativa. Mocha. Recuperado el 4 de noviembre de 2019 de <https://mochajs.org>

² ESLint (s.f.). Página Web Corporativa. ESLint. Recuperado el 4 de noviembre de 2019 de <https://eslint.org>

7. Arquitectura de la aplicación

La arquitectura de la aplicación está dividida en dos partes claramente diferenciadas. Por un lado, el ecommerce desde el que se consultará la información sobre los productos y se realizará la compra. Para el propósito de este proyecto actuará a modo de “servidor”. Por otro, el chatbot en sí, que actuará de “cliente”, se encargará de interactuar con el usuario y de mostrar los resultados que provengan de la API. Hay que tener en cuenta que tanto Magento 2, como cualquier aplicación en NodeJS, y en concreto MageChatbot, cuenta con su propia arquitectura interna.

Magento posee una arquitectura modular basada en MVVM que divide la parte de cliente y servidor. En todos los casos, se encuentra desarrollado en PHP como lenguaje nativo, pero con la particularidad de que cuenta con dos “clientes” accesibles desde el navegador: El *storefront* para el uso de los clientes y el *admin panel* para la gestión de la tienda, configuración y analítica (Matveev, 2018).

En las últimas versiones de Magento 2 se ha hecho un fuerte hincapié en desacoplar las funcionalidades tanto del *storefront* como del *admin panel* para dar entrada a una arquitectura modular. En ella, el *storefront* puede ser intercambiado por otro tipo de soluciones como aplicaciones en NodeJS que implementan un frontend con ReactJS o Vue.js utilizando su API REST o GraphQL como canal de comunicación (Ubertheme, 2019).

Por otra parte, debido a que NodeJS es un entorno de desarrollo y no un framework de propósito concreto, se pueden utilizar múltiples arquitecturas para diseñar una aplicación. A su vez, hay diferentes patrones que se pueden utilizar para estructurar el código. Todos estos patrones comparten como principios básicos la modularidad, la separación de responsabilidades y la reutilización de código mediante el uso de módulos y librerías externas (Volodymyr, 2018).

El siguiente gráfico resume la arquitectura planteada para este proyecto:

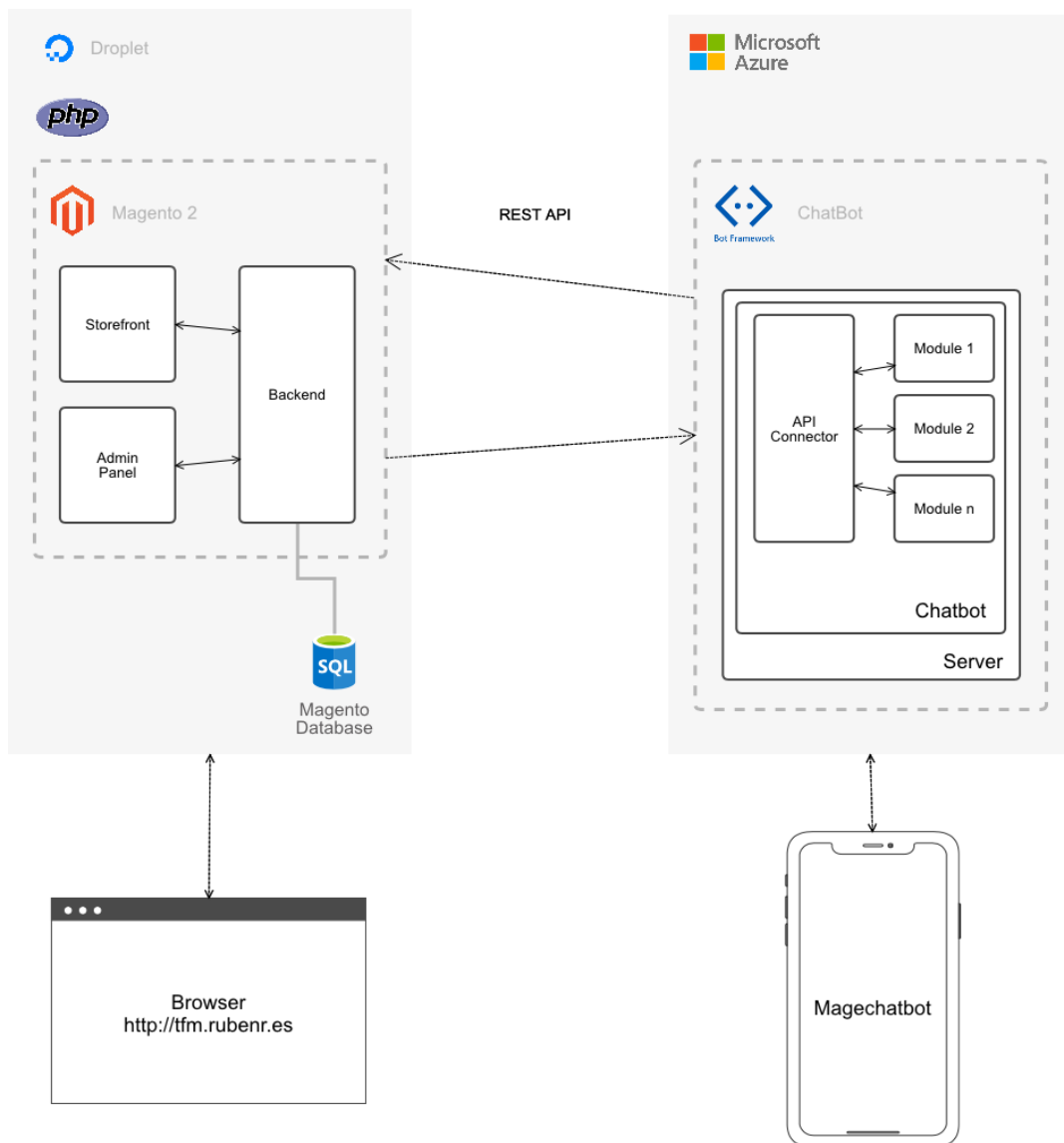


Figura 3: Esquema de la arquitectura propuesta. Elaboración propia.

Servidor

En nuestra aplicación, el servicio que actúa de servidor es una instancia de Magento 2.3 alojada en un VPS (*droplet*) de Digital Ocean. Este VPS actuará como si fuese un servidor físico, con la ventaja de ser administrable mediante un panel en la web de Digital Ocean.

Para que el *droplet* pueda ejecutar una instancia de Magento 2 es necesario que cumpla una serie de requisitos entre técnicos como se detalla posteriormente. Este droplet está formado por un stack básico LAMP, Ubuntu Linux, Apache, MySQL y PHP. Para operar sobre el *droplet*, Digital Ocean provee de una IP fija y acceso mediante SSH. Tras esto, se puede instalar Magento en el *droplet*.

Para facilitar el acceso al panel de administración y evitar el uso de la IP se ha creado el dominio <http://tfm.rubenr.es> y se ha hecho apuntar a la IP del droplet y al puerto 80 (puerto por defecto de Apache). Con Magento funcionando en el servidor se pueden realizar la configuración desde el *wizard*, la instalación de los datos de ejemplo y habilitar el acceso mediante API a Magento.

Ciente

El chatbot que actuará de cliente está formado por una aplicación en NodeJS con Microsoft Bot Framework para manejar la conversación y ofrecer resultados. Teniendo en cuenta los principios de modularidad y extensión³ la aplicación está dividida en:

- Módulo para la creación del servidor al que permitirán conectarse los clientes de mensajería
- Módulo para la gestión propia del bot, que importará Microsoft Bot Framework
- Módulo conector para conectarse con la API de Magento y gestionar las diferentes peticiones que se harán, tanto de consulta como de inserción de datos
- Diferentes módulos para la gestión de la información del producto, carrito, etc.

Para la exportación o instalación de la aplicación se ha hecho uso de los servicios de Microsoft Azure, que posee las herramientas necesarias para la creación de un servidor virtual, la instalación de la aplicación en NodeJS y la posterior publicación del chatbot en los servicios de mensajería instantánea (WhatsApp y Telegram, para este proyecto).

³ Magento (s.f.). Extensibility and modularity. Recuperado el 22 de octubre de <https://devdocs.magento.com/guides/v2.3/architecture/extensibility.html>

8. Plataforma de desarrollo

La plataforma de desarrollo gira en torno a la aplicación NodeJS, cliente que implementa Microsoft Bot Framework. A lo largo de este apartado se describen las herramientas y tecnologías empleadas en este proyecto

Software

Magento 2

Magento es una plataforma de ecommerce open-source con la que se pueden llevar a cabo todo tipo de proyectos relacionados con la venta en internet. Está basado en PHP e incorpora como tecnologías en el desarrollo HTML, CSS, LESS, Javascript, RequireJS, KnockoutJS, Grunt y Composer (Magento, 2019.a). Esta plataforma permite de un modo flexible crear desde sencillas tiendas administrables desde su backend a complejos marketplaces con múltiples integraciones para grandes empresas. Entre las características de esta plataforma destacan la posibilidad de personalizar el diseño, el soporte multi-idioma, la gestión de clientes, las múltiples formas de pago y envío, el control de stock, la gestión de comentarios, su CMS y sus herramientas de marketing y SEO⁴. Todas estas características y su modularidad la han hecho una de las plataformas más utilizadas y mejor valoradas (Magento, 2019.b).

Microsoft Bot Framework

Microsoft Bot Framework⁵ es una plataforma para el desarrollo de experiencias conversacionales con IA. Su gran potencia permite crear implementaciones de tipo empresarial. Este framework se puede integrar en aplicaciones con NodeJS de un modo sencillo para construir aplicaciones que cuenten con sistemas de conversación o IA. Junto con este framework y para realizar pruebas de la aplicación, Microsoft también tiene Bot Framework Emulator. Un emulador con el que probar las aplicaciones en tiempo real sin estar desplegadas en un entorno concreto, una aplicación de mensajería o una web.

NodeJS

NodeJS es un entorno en tiempo de ejecución para la capa de servidor basado en JavaScript⁶. Este entorno está basado en el motor de ejecución de JavaScript de Google V8 y permite crear aplicaciones altamente escalables posibilitando la ejecución de una aplicación compleja utilizando un solo lenguaje de programación. Asimismo, surge a la vez que el paradigma de programación basado en microservicios.

⁴ Magento (s.f.). Meaningful Commerce Experiences. Adobe | Magento Commerce. Recuperado el 4 de noviembre de 2019 de

<https://magento.com/products/magento-commerce/features-meaningful-commerce-experiences>

⁵ Microsoft (s.f.). Microsoft Bot Framework. Microsoft. Recuperado el 4 de noviembre de 2019 de <https://dev.botframework.com/>

⁶ OpenJS Foundation (s.f.). Página web corporativa. Node.js Recuperado el 15 de diciembre de 2019 de <https://nodejs.org/es/>

GitHub

GitHub es una plataforma para alojar proyectos usando el sistema de control de versiones Git⁷. Como en nuestro caso, este sistema se suele utilizar para almacenar el código fuente de programas de ordenador. En este proyecto, se ha utilizado el código para llevar un control, almacenar y poder desplegar posteriormente la aplicación cliente hecha con Microsoft Bot Framework. Para ello se ha creado un repositorio público llamado MageChatbot⁸ con el código fuente de la aplicación.

Hardware

Digital Ocean

Digital Ocean es un proveedor de infraestructuras cloud que ofrece múltiples servicios como si de un SaaS se tratara⁹. Para este proyecto se ha hecho uso de un *droplet*: una máquina virtual que opera como VPS y puede ser multipropósito. Con él se ha creado un servidor LAMP e instalado en este una instancia de Magento 2.

Microsoft Azure

Microsoft Azure es el nombre que recibe todo el conglomerado de servicios cloud de Microsoft. Permite tanto uso y almacenamiento de servicios de terceros como integración sencilla de servicios propios de Microsoft, como Microsoft Bot Framework¹⁰.

⁷ Github (s.f.).Página web corporativa. GitHub Recuperado el 2 de diciembre de 2019 de <https://github.com>

⁸ Rodríguez, R.(s.f.). MageChatbot GitHub. Recuperado el 2 de enero de 2020 de <https://github.com/rubenRP/magechatbot>

⁹ Digital Ocean (s.f.). Página Web Corporativa. Digital Ocean. Recuperado el 10 de octubre de 2019 de <https://www.digitalocean.com>

¹⁰ Microsoft (s.f.).Página web corporativa. Microsoft Azure. Recuperado el 27 de septiembre de 2019 de <https://azure.microsoft.com/es-es/>

9. Planificación

El proceso de planificación del proyecto está marcado por 3 fechas en las que se hacen entregas parciales de contenido y una entrega final en la que se presenta la totalidad el proyecto. Teniendo en cuenta estas fechas, el proyecto se divide en una serie de hitos equilibrando la carga de trabajo entre períodos y aportando valor en cada iteración, como se puede ver en la tabla 1.

Hitos

Nombre	Fecha del hito	Descripción
Fase 1	1/10/19	<ul style="list-style-type: none">● Contexto y justificación● Objetivos del Trabajo● Enfoque y método● Planificación del Trabajo
Fase 2	30/10/2019	<ul style="list-style-type: none">● Inicio del desarrollo● Documentación● Revisión de objetivos y planificación
Fase 3	8/12/2019	<ul style="list-style-type: none">● Continuación del desarrollo● Documentación● Revisión de objetivos y planificación
Fase 4	6/1/2020	<ul style="list-style-type: none">● Finalización del proyecto● Finalización de memoria● Elaboración de la presentación● Realización de la presentación (video)

Tabla 1: Tabla de hitos del proyecto.

Diagrama de Gantt

El siguiente diagrama de Gantt muestra una visión más detallada de la Tabla 1, las fechas de los hitos y las dependencias entre tareas.

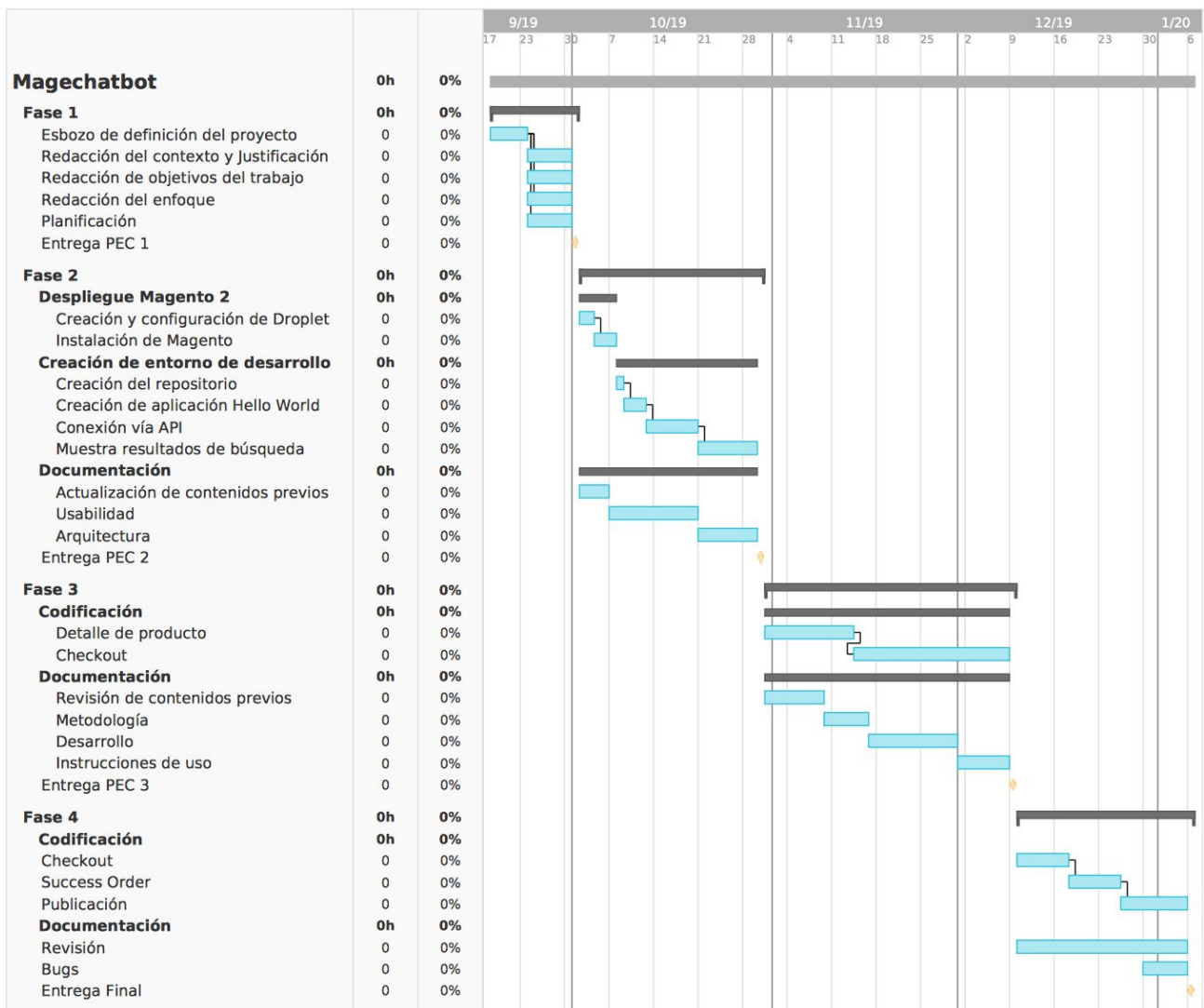


Figura 4: Diagrama de Gantt con la planificación del proyecto. Elaboración propia.

10. Proceso de desarrollo

Creación de la instancia de Magento 2

Para poder interactuar con Magento 2 primero es necesario crear una instancia de este framework. El modo más habitual es desplegando este software en un servidor web para que pueda ser accedido a través de un navegador. Para ello, el servidor web debe contar con una serie de requisitos de instalación concretos (Magento, 2019.a) como se define en el apartado 17 de este documento.

Un modo ágil para crear un servidor que cuente con estos requisitos es el uso de un VPS. Este sistema permite particionar un servidor físico en varios servidores virtuales que pueden tener diferentes configuraciones y se comportan como servidores reales (Rouse, 2017). Esta compartición facilita el escalado y la ampliación de recursos en caso de ser necesario, como un abaratamiento de costes y mayor optimización de los equipo. Para este caso se ha decidido utilizar Digital Ocean como proveedor de VPS.

Una vez creada la cuenta de Digital Ocean, se ha procedido a la creación de un Droplet (VPS) con Ubuntu 18.04 como SSOO, 2GB de RAM y 25GB de almacenamiento. Una vez creado el droplet se ha generado una IP pública que en este caso es 165.22.23.0. Para facilitar el uso del servidor una vez instalado Magento y hacerlo más amigable se ha creado el dominio <http://fm.rubenr.es> y se ha apuntado a esta IP y al puerto 80 (habitualmente el puerto utilizado con Apache). De este modo se podrá navegar por la tienda online desde esa dirección.

Para proceder a la instalación de los servicios necesarios para instalar Magento 2 se ha accedido al VPS por SSH y se ejecutan los siguientes comandos:

Instalación de PHP y complementos

```
sudo apt-get install php7.2 php7.2-cli php7.2-common php7.2-json php7.2-opcache  
php7.2-mysql php7.2-mbstring php7.2-zip php7.2-fpm
```

Instalación de Apache

```
sudo apt-get install apache2
```

Instalación y configuración de MySQL

```
sudo apt-get install mysql-server5.7
```

```
mysql -u root -p
CREATE DATABASE magento;
CREATE USER userRuben@localhost IDENTIFIED BY 'password';
CREATE USER userMagento@localhost IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON magento.* TO userRuben@localhost IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON magento.* TO userMagento@localhost IDENTIFIED BY 'password';
```

Creación de un proyecto de Magento 2

```
composer create-project --repository=https://repo.magento.com/
magento/project-community-edition magento
composer install
```

Llegado a este punto, al abrir el enlace <http://tfm.rubenr.es> se accede al Wizard para la instalación de Magento 2. En este proceso se han fijado los valores de la base de datos creada previamente y los datos de la cuenta de administración. A su vez, y para facilitar el posterior uso de la web, se han instalado los datos de prueba. Este proceso genera un catálogo de prueba con varios productos, usuarios, pedidos, como si de una tienda de moda se tratara.

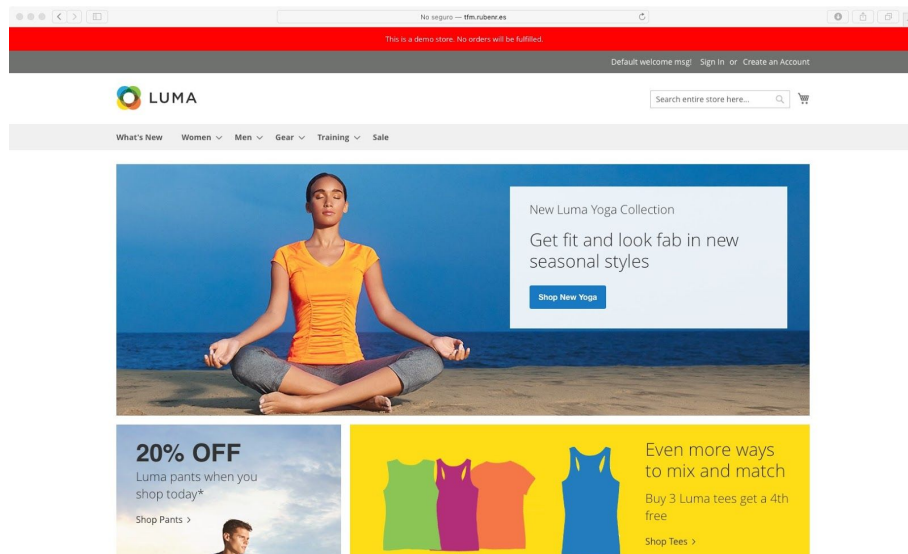


Figura 5: Home de <http://tfm.rubenr.es>

Desarrollo de nuevos modelos de interacción usuario-e-commerce. Integración de e-commerce en chatbot vía API REST Máster Universitario en Ingeniería Informática

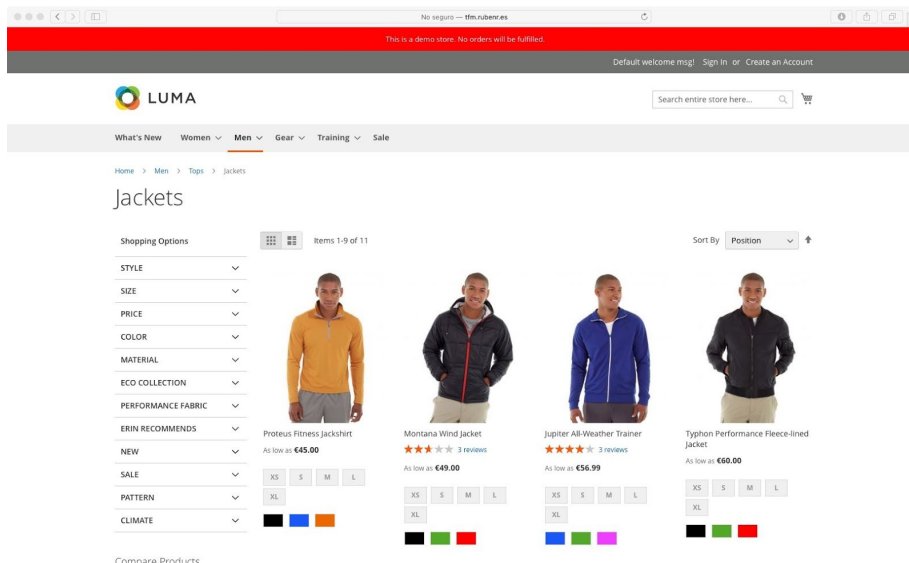


Figura 6: Listado de productos de <http://tfm.rubentr.es>

Configuración de Magento 2

Como parte principal del proyecto, ha sido necesario habilitar Magento para que se pueda interactuar desde su API REST, como se detalla en el apartado 11. Esto se ha realizado mediante lo que es conocido como “integraciones”¹¹. Con este sistema se pueden habilitar diferentes integraciones que permiten diferentes habilidades y el acceso a diferentes recursos. En nuestro caso, será necesario el acceso al catálogo (lectura), usuarios, carrito y checkout (lectura y escritura). Tras crear la integración se ha provisto de una *consumer key*, *consumer secret*, *access token* y *access token secret*. Estos datos son necesarios en nuestro chatbot para poder operar con Magento.

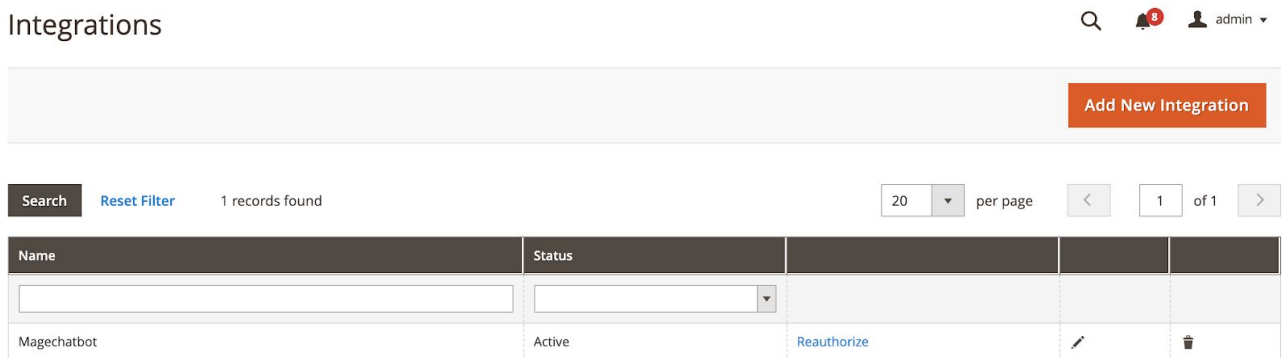


Figura 7: Detalle de la ventana de integraciones en el área de administración de Magento 2

¹¹ Magento (s.f.). REST API reference. Magento DevDocs. Recuperado el 22 de octubre de 2019 de <https://devdocs.magento.com/redoc/2.3/>

Creación del chatbot

Para crear un chatbot con Microsoft Bot Framework se puede optar por dos lenguajes de programación: C# y JavaScript. Para este proyecto se ha decidido optar por crear un chatbot con JavaScript e integrarlo dentro de una aplicación con Node.js. Para ello, los requisitos necesarios de la aplicación son los siguientes¹²:

- Visual Studio Code
- Node.js
- Yeoman
- Git
- Bot Framework Emulator.

Para generar el entorno de desarrollo se ha hecho uso de Yeoman, un sistema que permite generar una estructura de ficheros de modo automático siguiendo plantillas. Mediante una serie de preguntas y respuestas se han creado todos los ficheros necesarios para crear un chatbot con el propósito definido y a la vez se han generado las configuraciones necesarias para el posterior despliegue del chatbot dentro de los servicios de Microsoft Azure. Para nuestro caso, se ha elegido la estructura básica, ya que la integración de API's no es una entre las contempladas por el Framework¹³.

Los pasos a seguir para generar la estructura de un chatbot básico que repite lo que escribe el usuario (chatbot de eco) con JavaScript son los siguientes:

```
npm install -g windows-build-tools
npm install -g npm
npm install -g yo generator-botbuilder
yo botbuilder
```

Desarrollo del chatbot

Una vez generado el bot de eco se comienza con el desarrollo del chatbot. Debido al carácter conversacional del proyecto y que puede ser utilizado por múltiples usuarios de modo concurrente hay tres

¹² Microsoft (s.f.). Tutorial de creación de un bot básico. Microsoft Azure. Recuperado el 2 de octubre de 2019 de <https://docs.microsoft.com/es-es/azure/bot-service/bot-builder-tutorial-basic-deploy>

¹³ Microsoft (s.f.). BotBuilder-Samples. GitHub. Recuperado el 4 de octubre de 2019 de <https://github.com/microsoft/BotBuilder-Samples>

figuras en el desarrollo claves a tener en cuenta: perfil de usuario, datos de la conversación y estado del chatbot.

Figuras clave del desarrollo

Estado del chatbot

El diálogo es uno de los conceptos principales en el desarrollo de un chatbot. Como en las conversaciones entre personas, este diálogo puede ser lineal e independiente entre mensajes o bien estar basado en la información previamente establecida en la conversación. Bajo esta idea se fundamenta el estado del chatbot. Para algunos bots, no es necesaria esta gestión para mantener la simplicidad. Para otros, como es el caso del proyecto, es necesaria la gestión de estados para que el bot mantenga una conversación útil.

El ciclo de compra de un ecommerce puede ser representado mediante un diagrama de estados. Este diagrama puede ser exportado conceptualmente al chatbot para que así se puedan gestionar todas las casuísticas del proyecto. Típicamente y simplificando estos estados se resumen en búsqueda, visualización del producto, añadir al carrito y proceso de pago (checkout). Aun así, hay múltiples variaciones entre estos estados básicos y muchos componentes extra que pueden añadirse a este diagrama como la gestión de usuarios, favoritos o sistemas de comparación categorización.

Perfil de usuario

Como se ha comentado previamente, un chatbot por defecto carece de estado. Independiente de la gestión de estado, la mera ejecución del chatbot por un usuario implica la creación de un nuevo contexto para dicho usuario. Este contexto puede utilizar información del usuario para manejar el comportamiento del bot. Un ejemplo de información almacenada en el perfil de usuario puede ser la hora de conexión, el equipo, o como es en nuestro caso, un id de customer, o un id de carrito. Estos datos son individuales para cada usuario y únicos, tanto a nivel de chatbot como a nivel de Magento. La compartición de este tipo de claves implicaría una violación de privacidad y seguridad. Microsoft Bot Framework provee de herramientas para la creación de perfiles individuales de usuario una vez comienza la creación de un nuevo contexto. Esto se realiza mediante la importación de la clase `UserState`.

```
const { BotFrameworkAdapter, MemoryStorage, ConversationState, UserState } =  
require('botbuilder');
```

Este objeto ha sido inicializado en el constructor de nuestro chatbot y asigna un perfil de usuario cada vez que se agrega un nuevo miembro al chatbot

```
this.userState = new UserState(memoryStorage);
```

```
this.userProfileAccessor = userState.createProperty(USER_PROFILE_PROPERTY);
```

Para añadir información relevante al estado del usuario se opera sobre el objeto en uno de los handles básicos que ofrece Bot Framework, como puede ser onMessage (el usuario escribe un mensaje).

```
this.onMessage(async (turnContext, next) => {  
  // Get the state properties from the turn context.  
  const userProfile = await this.userProfileAccessor.get(turnContext, {});  
  
  userProfile.productQuery = turnContext.activity.text;  
}
```

Datos de conversación

Junto a la figura de perfil de usuario existe la figura de datos de conversación (conversationData). Esta clase se comporta de un modo similar a userState, pero su objetivo es diferente. En este caso, el objetivo de esta clase es almacenar información relativa al estado del chatbot en esa conversación y no necesariamente relativa al usuario. Dado que en MageChatbot se basa en estados, se almacena el estado del proceso de compra en el que se encuentra el usuario.

```
constructor(conversationState, userState) {  
  super();  
  // Create the state property accessors for the conversation data and user profile.  
  this.conversationDataAccessor = conversationState.createProperty(  
    CONVERSATION_DATA_PROPERTY  
  );  
}
```

```
this.onMessage(async (turnContext, next) => {  
  const conversationData = await this.conversationDataAccessor.get(  
    turnContext,  
    {  
      botState: 1  
    }  
  );  
}
```

Estructura del proyecto

La estructura inicial del proyecto contiene en la raíz un archivo *package.json* donde se especifica la información sobre el proyecto, las dependencias de otros paquetes, un archivo *index.js* que pone en marcha el servidor que ejecuta el chatbot, un fichero *bot.js* con el cuerpo del chatbot y su lógica base y una serie de ficheros bajo la carpeta *modules* que contiene lógica de negocio, gestión de las llamadas a la API de Magento de un modo modular.

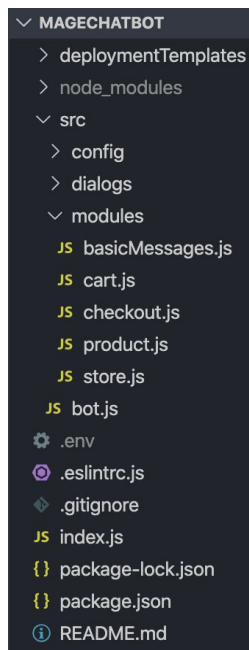


Figura 8: Estructura de ficheros del proyecto MageChatbot.

package.json

Es el archivo principal de configuraciones del proyecto. Se encuentra en la raíz del mismo para ser compatible con gestores de módulos como NPM y Yarn. En él se refleja tanto el nombre del proyecto, como su descripción, scripts, autor, licencia y dependencias.

index.js

Es el archivo principal de la aplicación, pone en marcha el servidor mediante Restify y lanza el chatbot mediante la importación del fichero de lógica de chatbot. De este modo se mantiene aislado la funcionalidad del servidor, que se encarga de “escuchar” las peticiones entrantes y el fichero del bot que se ocupa de la lógica de negocio del mismo.

bot.js

Este fichero contiene la clase MageChatbot que se encarga de gestionar la funcionalidad básica del chatbot y el movimiento entre estados del mismo. Esta funcionalidad se describe en detalle a continuación.

Flujo de ejecución

La ejecución del chatbot se puede dividir en tres estados básicos realizados por parte del usuario. Estos estados son parte de Bot Framework y caracterizan someramente las acciones de un usuario de cualquier chatbot. Estas son:

- **onMembersAdded**. Esta función es llamada cuando un usuario inicia el chatbot. Es útil para mostrar información de bienvenida o para fijar valores a estados iniciales del usuario, como si de un constructor se tratara. En el caso de este chatbot la función se encuentra dentro del constructor por lo que no puede ser tratada como tal.

```
this.onMembersAdded(async (context, next) => {  
    await BasicMessages.sendWelcomeMessage(context);  
    await BasicMessages.sendSuggestedActions(context);  
    await next();  
});
```

- **onMessage**. Esta función es llamada cuando un usuario escribe un mensaje o selecciona dentro de las posibilidades ofrecidas por el chatbot. En el caso de MageBot la gestión de los estados y las diferentes posibilidades dentro del flujo de compra se desarrolla dentro de esta función.
- **onDialog**. Esta función es llamada cuando el usuario se encuentra dentro de un diálogo. Un diálogo es una secuencia de preguntas y respuestas que al finalizar se tratan a la vez. En el caso de Magebot se utiliza para consultar los datos necesarios para la compra, dirección de envío, datos de usuario, etc. Esto permite simplificar el tratamiento de mensajes en la función onMessage, ya que los datos para el pago se envían a la vez mediante la Api.

```
this.onDialog(async (turnContext, next) => {  
    // Save any state changes. The load happened during the execution of the Dialog.  
    await this.conversationState.saveChanges(turnContext, false);  
    await this.userState.saveChanges(turnContext, false);  
  
    // By calling next() you ensure that the next BotHandler is run.  
    await next();  
});
```

```
});
```

Gestión de estados en la función onMessage

Para gestionar las diferentes posibilidades del diagrama dentro de la función onMessage, se ha hecho uso de dos declaraciones switch. Todas estas posibilidades ejecutan las acciones correspondientes a su estado y modifican este dirigiendo al estado siguiente en función de lo decidido por el usuario. Este estado se encuentra almacenado en el objeto conversationData.botState.

Las acciones del usuario pueden ser de dos tipos: condicionales y generales. Las acciones condicionales son el resultado de ofrecer al usuario varias opciones dentro de un estado. Por ejemplo, la búsqueda de productos termina en una elección condicional donde se pregunta al usuario si desea añadir el producto al carrito.

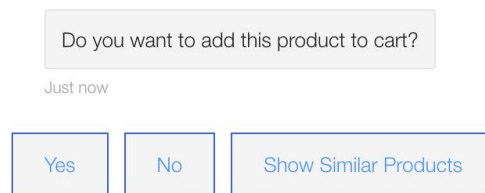


Figura 9: Ejemplo de mensaje condicional.

Las acciones generales se suelen ejecutar tras una acción condicional y suelen implicar una llamada a la API y una muestra de resultados. Estas acciones se gestionan mediante el segundo switch.

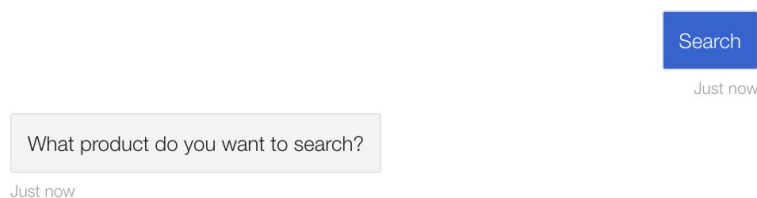


Figura 10: Ejemplo de mensaje general.

Diálogos

Los diálogos son un concepto utilizado en Microsoft Bot Framework y proporcionan una forma útil de administrar conversaciones con los usuarios. Son estructuras de un bot que actúan como funciones en su

programa; cada diálogo está diseñado para realizar una tarea específica en un orden concreto¹⁴. En el caso de MageChatbot se utiliza para administrar el proceso de compra, ya que son una serie de preguntas-respuestas en las que se guarda la información relativa al pedido y acaba con el envío de la información a la API para proceder a finalizar el pedido.

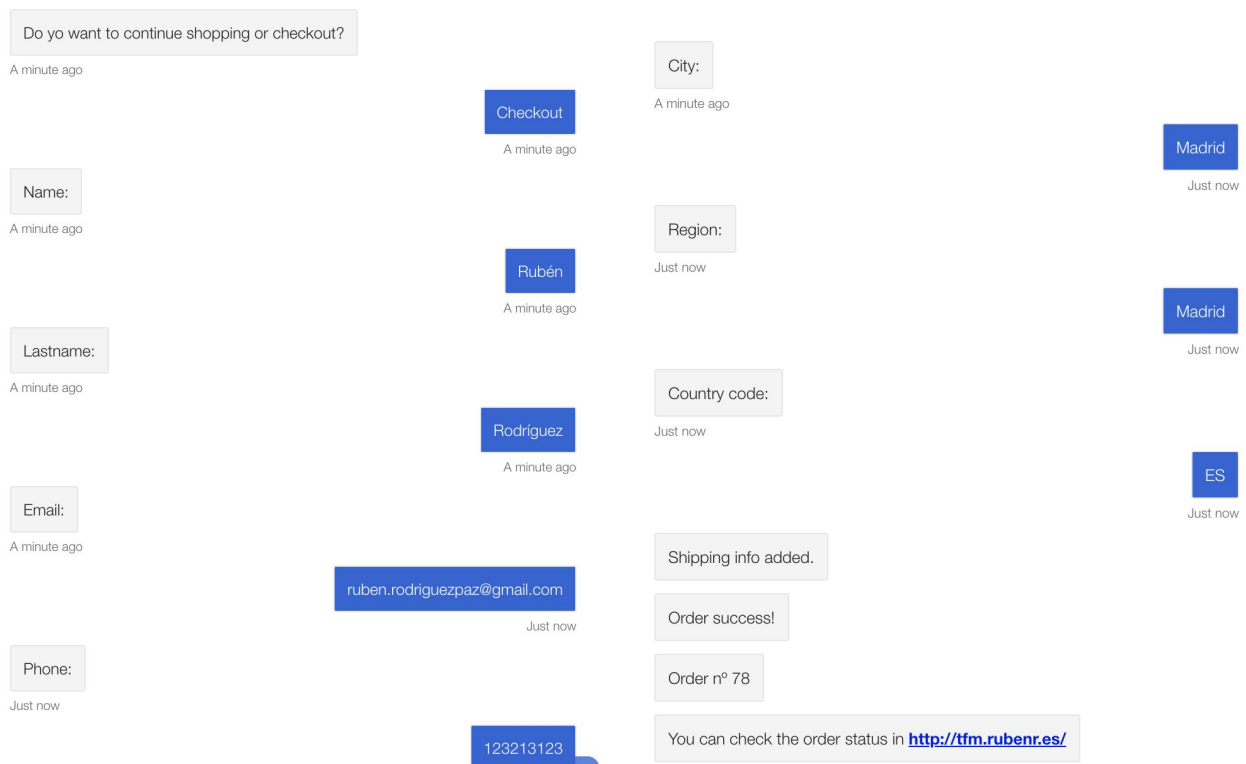


Figura 11: Flujo de ejemplo del proceso de *checkout*.

Una de las ventajas del uso de diálogos en el proceso de *checkout* es que se extrae toda la funcionalidad fuera del bloque de lógica del fichero *bot.js*. En este caso, el diálogo relativo al proceso de pago se encuentra en *dialogs/shippingDialog.js*. La estructura resumida del diálogo es la siguiente:

```
class ShippingDialog extends ComponentDialog {
  constructor(userState) {
    super("shippingDialog");

    this.userProfile = userState.createProperty(USER_PROFILE);

    this.addDialog(new TextPrompt(NAME_PROMPT));

    this.addDialog(
```

¹⁴ Microsoft (s.f.). Biblioteca de Diálogos. Microsoft Azure. Recuperado el 15 de noviembre de 2019 de <https://docs.microsoft.com/es-es/azure/bot-service/bot-builder-concept-dialog>

```
new WaterfallDialog(WATERFALL_DIALOG, [
  this.nameStep.bind(this),
  this.lastnameStep.bind(this),
])
);

this.initialDialogId = WATERFALL_DIALOG;
}
async nameStep(step) {
  return await step.prompt(NAME_PROMPT, "Name:");
}
async confirmStep(step) {
  step.values.country = step.result;
  return await step.endDialog(step.values);
}
```

En el fragmento previo se puede observar un constructor en el que se define el tipo de pasos o preguntas a realizar (*steps*) y su orden junto con la función *nameStep* que se encarga de almacenar la información del nombre del usuario y la función *confirmStep*, que devuelve el objeto *steps.values* con toda la información recogida a la función *onMessage* de *bot.js* donde ha sido invocada. Posteriormente se toma la información y se envía al módulo *checkout.js*.

Depuración

Para analizar la evolución del chatbot y los parámetros almacenados durante la ejecución se han dispuesto de varios logs de consola para poder trazar los pasos por los que se van pasando y los valores del texto introducido.

```
User text: Ruben
Case 5
User text: Rodriguez
Case 5
User text: ruben.rodriguezpaz@gmail.com
Case 5
User text: 213213123
Case 5
User text: Fake Street 123
Case 5
User text: 28981
Case 5
User text: Parla
Case 5
User text: Madrid
Case 5
User text: ES
Case 5
SetPayment and order
{ instanceId: '098c37c6-e59d-9a51-9a48-1799c78ceec4',
  name: 'Ruben',
  lastname: 'Rodriguez',
  email: 'ruben.rodriguezpaz@gmail.com',
  phone: '213213123',
  street: 'Fake Street 123',
  postcode: '28981',
  city: 'Parla',
  region: 'Madrid',
  country: 'ES' }
error: API call failed: "email" is required. Enter and try again.
```

Figura 12: Ejemplo de log de debug en la consola de la aplicación.

A su vez, también se muestran en el terminal los errores detallados en el caso de que haya un problema en la llamada a la API de Magento 2. De este modo queda constancia del error sin notificar al usuario con información demasiado concreta, lo que podría suponer un problema de seguridad. Para poder ver realizar la depuración de la aplicación en Microsoft Azure se ha habilitado el *logging* de la aplicación, de modo que se guarda tanto la información relativa a la ejecución de la instancia de Azura como la información definida previamente en MageChatbot.

11. APIs utilizadas

Para hacer posible la conexión entre la aplicación que gestiona el chatbot y la tienda online se ha hecho uso de la API REST de Magento. Esta API provee de los métodos necesario para realizar búsquedas, listar productos, ofrecer detalle de los mismos, añadir al carrito y proceder al pago. A su vez también existen métodos para gestionar usuarios registrados, realizar registros e incluso añadir productos a una lista de deseos. La referencia de los *endpoints* ofrecidos por Magento 2 se puede consultar en la web para desarrolladores de Magento 2¹⁵. A su vez, para cada instancia de Magento 2 existe un entorno Swagger para poder ver tanto los *endpoints* genéricos de Magento como los propios de cada proyecto. En nuestro caso, ya que no se realizarán desarrollos adicionales o personalizados en Magento 2, los *endpoints* provistos han sido iguales a los de la web de referencia de Magento. En cualquier caso, el listado de todos los *endpoints* de nuestro proyecto se pueden consultar en la ruta `/swagger` de cualquier proyecto con Magento 2¹⁶ siempre que la aplicación no se encuentre en “modo producción”.

Para hacer uso de cualquier método provisto por la API de Magento 2, es necesario un paso previo de autenticación. Magento 2 permite diferentes tipos de autenticación mediante API; en este caso se hará uso de autenticación con OAuth. Para eso, ha sido necesario el uso del endpoint `/integration/customer/token` para obtener un token de sesión. En esta llamada se han enviado las keys y secretos provistos en Magento 2 al crear la integración para el chatbot. La respuesta ha sido un hash que ha sido almacenado al ejecutar la aplicación y que es enviado en cada llamada a la API posterior.

Para la integración de la API de Magento 2 con MageChatbot se ha hecho uso del módulo de NPM `magento2-rest-client`¹⁷. Un módulo desarrollado por la compañía Divante creado para la integración de Magento 2 con su producto Vue Storefront¹⁸: aplicación PWA basada en VueJS que provee de una capa de frontend desacoplada de la proporcionada por Magento2. Por la robustez de este módulo y la propia complejidad y duración en desarrollo de un conector sólido se ha decidido hacer uso de este. En cualquier caso y de modo genérico, se podría hacer uso del módulo Axios¹⁹ para la comunicación de la aplicación NodeJS con cualquier API.

¹⁵ Magento (s.f.). REST API reference. Magento DevDocs. Recuperado el 22 de octubre de 2019 de <https://devdocs.magento.com/redoc/2.3/>

¹⁶ Rodriguez, R. (s.f.). Magento Community. Swagger. Recuperado el 2 de enero de 2020 de <http://tfm.rubenr.es/swagger>

¹⁷ DivanteLtd (s.f.) Magento 2 REST client. GitHub. Recuperado el 2 de octubre de 2019 de <https://github.com/DivanteLtd/magento2-rest-client>

¹⁸ DivanteLtd (s.f.) Vue Storefront. GitHub. Recuperado el 2 de octubre de 2019 de <https://github.com/DivanteLtd/vue-storefront>

¹⁹ Axios (s.f.). Axios: GitHub. Recuperado el 2 de septiembre de 2019 de <https://github.com/axios/axios>

De todos los *endpoints* listados en Swagger y en la documentación para desarrolladores de Magento 2, los que se han usado para la aplicación han sido los siguientes:

/products/product/media

Este endpoint se encarga de obtener las imágenes de un producto. Para ello es necesario enviar el SKU del producto. La petición devuelve como respuesta exitosa un array con las imágenes del producto requerido.

/products?searchCriteria=

Endpoint encargado de realizar búsquedas de productos. Es uno de los endpoint más utilizados en Magento 2 y de los más complejos ya que, para realizar una búsqueda, es necesario el uso de los *searchCriteria*²⁰, una concatenación de restricciones para filtrar una colección de productos. Por la arquitectura de Magento 2, los productos forman parte de una identidad genérica que puede ser filtrada por atributos (nombre, talla, color, sku...). Para la búsqueda empleada en MageChatbot se ha hecho uso de varios filtros para responder a la búsqueda del usuario, tanto por nombre, sku como descripción del producto.

/guest-carts

Este endpoint se encarga de crear un nuevo carrito para un nuevo usuario, para ello no será necesario enviar ningún parámetro adicional, ya que se trata de una compra para usuarios invitados. La llamada a este endpoint crea una nueva fila en la tabla *quotes* de Magento 2 a la que se pueden asignar productos. La respuesta correcta de esta llamada devuelve un parámetro *cartId* que se almacena para poder asignar productos si el usuario añade un producto al carrito.

/guest-carts/cartId/items

Para la acción de añadir productos al carrito de cada usuario se hace uso de este endpoint. Para ello, es necesario enviar tanto el parámetro *cartId* mencionado previamente dentro de un objeto de tipo *cartItem* que contiene la información necesaria del producto.

```
const cartItem = {
  sku: cartSku,
  qty: 1,
  quote_id: cartId
};
```

Por simplicidad, no se gestionan los incrementos de producto en el chatbot, pero se pueden añadir de uno en uno, sumándose a los existentes en carrito.

²⁰ Magento (s.f.). Search for product with the /search endpoint. Magento DevDocs. Recuperado el 4 de noviembre de 2019 de

<https://devdocs.magento.com/guides/v2.3/rest/search-endpoint.html>

/guest-carts/cartId/totals/

Este endpoint se encarga de obtener los totales del carrito (subtotal, impuestos, total) dado un *cartId*. En MageChatbot se usa cada vez que se añade un producto al carrito, a modo de resumen y recordatorio del estado actual del carrito, ya que no se cuenta como es habitual en los ecommerce de un icono con el valor de los productos que se encuentran en el carrito.

/guest-carts/cartId/shipping-information

El proceso de checkout por API en Magento 2 se divide en 2 etapas: (1) métodos y dirección de envío (*shipping information*) y (2) método de pago y dirección de facturación (*payment information*). El endpoint *shipping-information* se encarga de capturar la información de envío. Por efectos de simplicidad, como ocurre en múltiples ecommerce, se usa un solo método de envío.

/guest-carts/cartId/payment-information

La captura de los datos de facturación y método de pago se realiza mediante el endpoint *payment-information*. Tanto la captura de los datos de envío como los de pago componen un pedido (*Order* en Magento 2). El endpoint *payment-information* captura la información de pago y finaliza el pedido, devolviendo un *orderId* cuando el resultado del pedido es exitoso.

12. Diagramas

El diseño de la aplicación se ha hecho en base a un análisis de las funciones necesarias de un chatbot, los pasos necesarios para operar con Magento 2 y, a su vez, al cumplimiento de los objetivos fijados. Para estructurar la petición de información en el chatbot y todos los posibles flujos que puede realizar el usuario en la aplicación se ha utilizado un diagrama de estados (figura 13). Este diagrama define todas las acciones que el usuario puede realizar y los estados que recorrerá en la aplicación. Estos estados tienen acciones asociadas como realizar consultas a Magento, guardar información en el perfil de usuario o analizar la respuesta del cliente para moverse entre estados.

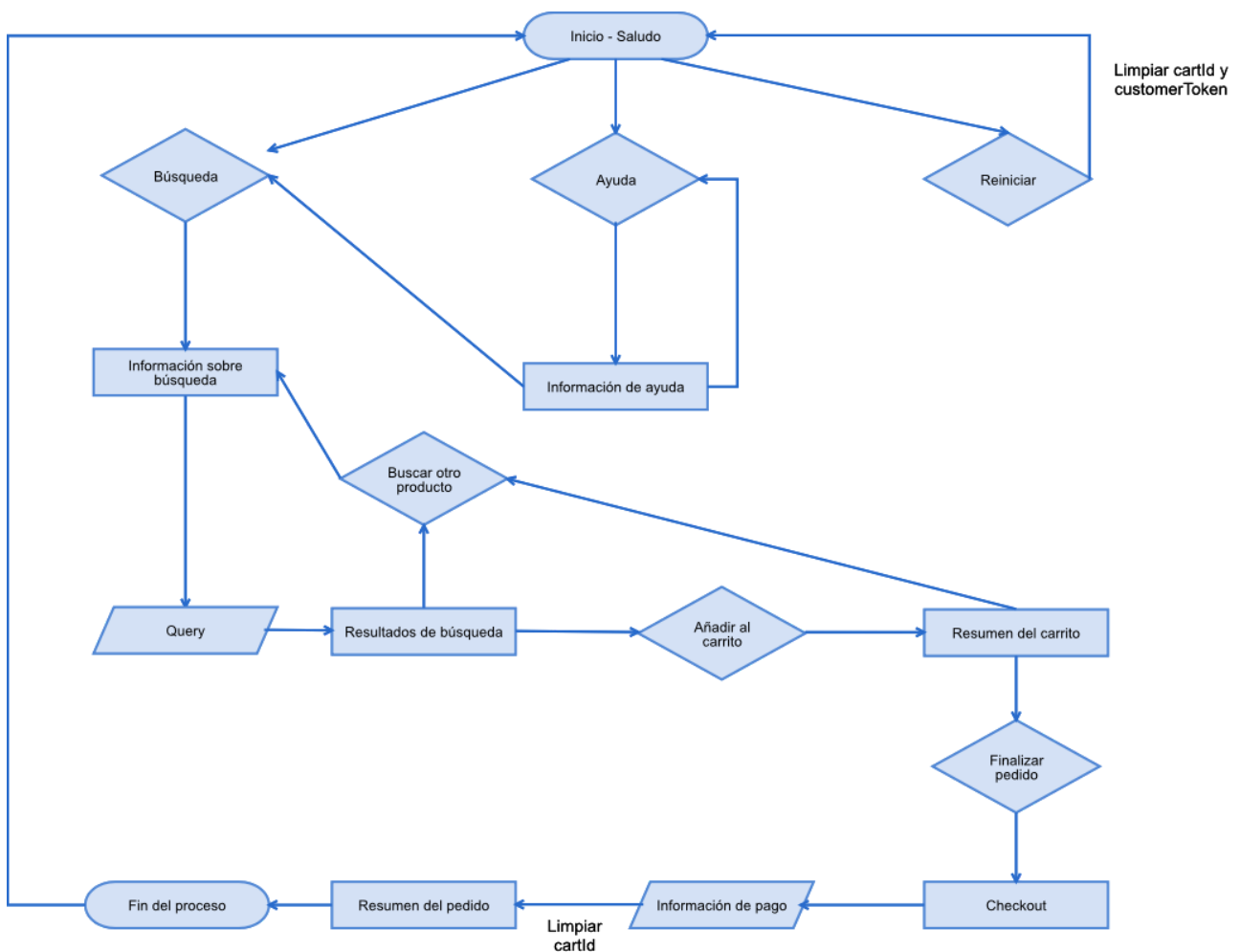


Figura 13: Diagrama de estados de la aplicación. Elaboración propia.

A su vez, este diagrama ha permitido, en la fase de desarrollo, dividir las funcionalidades por módulos atendiendo a su propósito. Así tendremos un módulo de mensajes generales *BasicMessages*, un módulo para ofrecer la información relativa al producto *Product*, un módulo con las acciones relativas al carrito *Cart* y un módulo con la gestión del proceso de compra y pago *Checkout*.

13. Perfiles de usuario

Habitualmente en todas las plataformas de ecommerce existen dos perfiles de usuario bien definidos: usuarios invitados y usuarios registrados (Resnick, 2018).

Usuarios invitados

Estos usuarios no están almacenados en la base de datos, por tanto no se tiene ninguna información previa del mismo. En este caso se permite la compra (hay tipos de tiendas en las que no) pero, para ello, tienen que rellenar un formulario de facturación, envío y el tipo de pago preferido, etc. Es habitual que, en los procesos de compra, se ofrezca al usuario registrarse para almacenar los datos y que sean usados como datos por defecto para próximas compras (Charlton, 2018).

Usuarios registrados

Estos usuarios han rellenado un formulario de registro. La información de registro inicial es variable, por lo que hay casos que ya se guarda la información de facturación y otros en las que la información almacenada es mínima (correo electrónico y contraseña). Este tipo de usuarios pueden guardar varias direcciones de facturación, direcciones de envío, métodos de pago preferidos y artículos en la lista de deseos.

Para este tipo de usuarios la información a cumplimentar en el proceso de compra suele ser menor que la de los usuarios invitados. A su vez, Magento permite al gestor o propietario de la tienda asignar estos usuarios a otro tipo conjunto de roles llamado "Grupos de cliente" que agrupan usuarios por tipología, modelo de negocio, gasto realizado, etc., facilitando tanto las métricas y el análisis de las ventas como la gestión de ofertas y descuentos para los usuarios de dicho grupo de clientes (Kovacevic, 2019).

De cara a la gestión interna de los usuarios, la asignación de ofertas u otros criterios de negocio existen figuras adicionales como grupos de clientes o segmentos de clientes, pero no aplican para el propósito de este proyecto.

En el caso de nuestra aplicación y como parte de los objetivos principales se han usado usuarios invitados. Como parte de los objetivos secundarios se tratará la gestión de usuarios registrados, inicio de sesión, registro y compra de dichos usuarios.

14. Usabilidad/UX

Por la tipología de aplicación, los patrones UX seguidos y todo lo relacionado a la usabilidad de la aplicación viene fijado por las posibilidades y la arquitectura visual que definen las aplicaciones de mensajería para móviles, como WhatsApp o Telegram. De hecho, el uso de clientes web son simples implementaciones de la interfaz móvil sobre una web, por lo que aunque a niveles de usabilidad cumple, no están pensadas para su uso estricto en la web (Intercom, 2016). Estas aplicaciones parten de unos principios similares en los que prima el uso de la pantalla táctil, la necesidad de reducir el aprendizaje de las funcionalidades de la aplicación a la mínima expresión y la posibilidad de ofrecer comunicación individual y en grupos (Nizam, 2018). Esta es una de las razones por la que la apariencia entre los diferentes clientes de mensajería sea similar (figura 14).

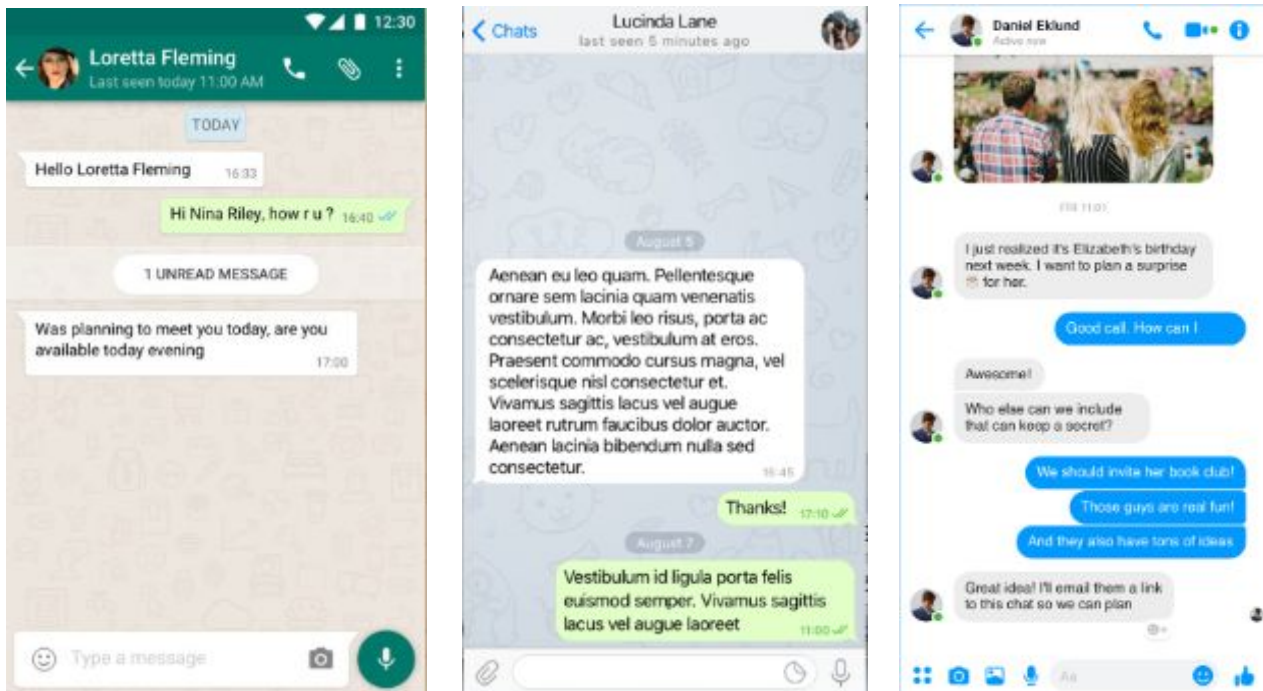


Figura 14: Ejemplos de conversaciones en diferentes aplicaciones de mensajería (WhatsApp, Telegram, Facebook Messenger). (Facebook, 2019. Telegram, 2019)

A los usos habituales de los clientes de mensajería en los que se puede intercambiar texto, imágenes, audios, diferente contenido multimedia, y tras la aparición de los primeros chatbots para clientes de mensajería móvil, se les sumó otro tipo de caso de comunicación: la elección entre una serie de posibilidades con las que el usuario podía interactuar (The Engine Room, s.f.) (Figura 15). Esto permitía a los chatbots filtrar la conversación entre los diferentes casos de uso que tenían contemplados.

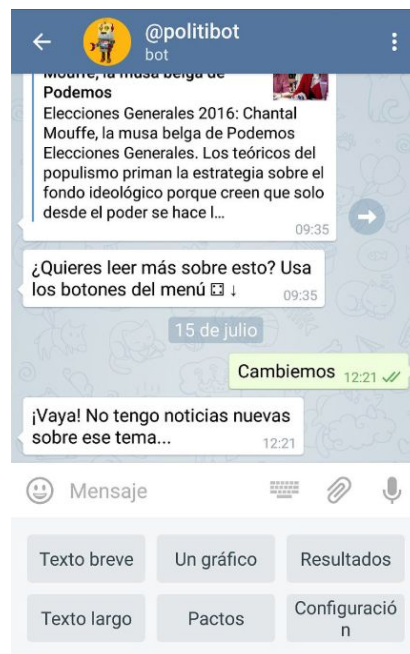


Figura 15: Ejemplo de chatbot en Telegram. (Politibot, 2019)

Teniendo esto en cuenta, hay dos tipos de interacciones que el usuario puede realizar en MageChatbot: inserción de texto y elección.

Inserción de texto

Esta interacción suele venir precedida de una pregunta previa, como puede ser la búsqueda de productos o cuando el usuario desea finalizar el pedido y es necesario rellenar la dirección de facturación, etc. En cualquiera de los casos, el análisis del texto introducido contiene el atajo “help”, que dirige automáticamente al usuario a la información de ayuda del bot, en caso de que necesite crear otro pedido y reiniciar el bot o añadir más productos al carrito. Interacción entre cosas.

Elección

Esta interacción suele ir precedida de la impresión de información por parte del chatbot y permite al usuario decidir qué desea hacer y así poder navegar entre los diferentes estados del chatbot. Estos mensajes se capturan inicialmente en la función *onMessage* y en función del valor, modifican el estado siguiente para que se pueda continuar con la acción del chatbot en el segundo *switchcase* de la función *onMessage*.

15. Seguridad

Teniendo en cuenta la arquitectura de la aplicación, la seguridad de la misma depende de 4 factores: la infraestructura donde se aloja Magento 2, Magento 2, la infraestructura donde se aloja el chatbot y el chatbot.

Infraestructura de Magento 2

Este área cubre todo lo relacionado con la configuración del servidor donde se aloja Magento 2, estado de las actualizaciones, puertos abiertos, módulos habilitados, etc., siendo en el caso de este proyecto el *droplet* de Digital Ocean. La configuración de este VPS en el proyecto es básica y no contempla picos de usuarios, alta concurrencia o alta disponibilidad, ya que no es objeto de este proyecto y no se encuentra en el alcance.

Magento 2

Este área cubre lo relacionado con la arquitectura del software, patrones de desarrollo, almacenamiento de contraseñas, control de perfiles de usuario y roles, etc

No se encuentra en el alcance de este proyecto.

Infraestructura de MageChatbot

El área de la infraestructura de Microsoft Azure cubre toda la configuración de los grupos de recursos, recursos creados y su configuración, sistemas, actualizaciones, puertos etc..., .en el caso de este proyecto la configuración es básica y está basada en la documentación de Microsoft. Esta configuración no contempla (o no refleja) control de disponibilidad, intrusión o actividades de seguridad y a su vez no se encuentra dentro del alcance.

MageChatbot

Respecto a la seguridad relativa al chatbot existen una serie de datos que deben ser protegidos ya que son parte del usuario. En el caso de MageChatbot se trata de los token de carrito, usuario y la información relativa a la compra. Esta información es almacenada en el "Perfil de Usuario" de Microsoft Bot Framework²¹. Según la documentación de Microsoft, la información del objeto dentro del framework está cifrada y se almacena en el *memory storage* de la aplicación. Esto no exime que la aplicación deba cumplir con la legislación relativa a la GDPR²².

²¹ Microsoft (s.f). Save user and conversation data. Microsoft Azure. Recuperado el 4 de noviembre de 2019 de Recuperado de <https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-howto-v4-state>

²² Microsoft (s.f.) General data protection regulation. Microsoft | Bot Framework. Recuperado el 10 de diciembre de 2019 de <https://blog.botframework.com/2018/04/23/general-data-protection-regulation-gdpr/>

Por otra parte, toda la información relativa a las comunicaciones vía API con Magento y, por la arquitectura de la aplicación están tokenizadas. Al comenzar un diálogo con el chatbot se genera un token único que identifica al usuario de forma anónima en las comunicaciones.

16. Tests

Para el proceso de pruebas de la aplicación se ha decidido utilizar un entorno de desarrollo local, tanto para Magento 2 como para MageChatbot. De ese modo se pueden aislar los problemas derivados de la infraestructura como se detalla en el apartado 20. Para poder probar la aplicación correctamente se ha hecho uso de Microsoft Framework Emulator, un emulador de mensajería especialmente pensado para usar con Microsoft Bot Framework²³.

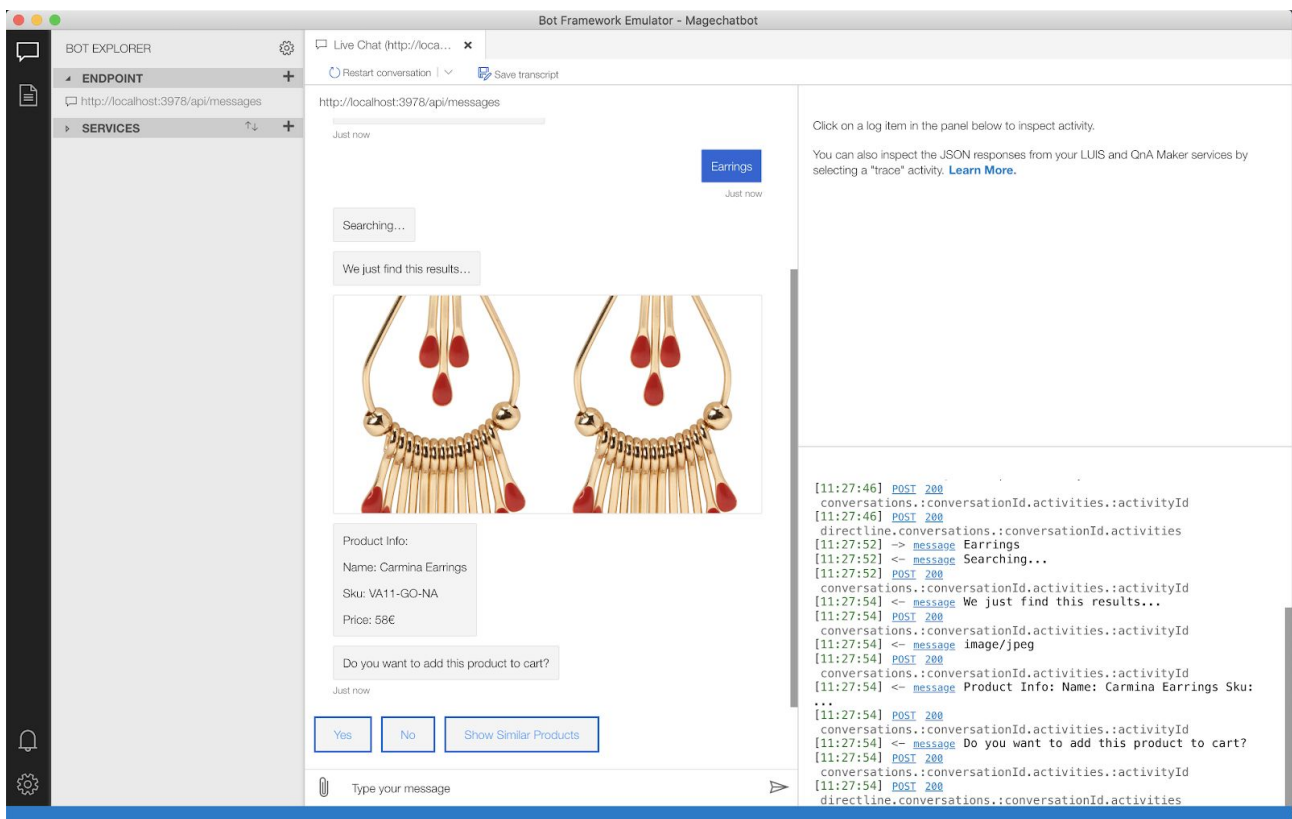


Figura 16: Proceso de test y depuración en microsoft bot framework.

Para el proceso de testing se han recorrido todas las posibilidades del diagrama de flujo de la aplicación y se han validado los resultados tanto con lo mostrado en consola (como se detalla en el capítulo 10) como los resultados de cada pedido dentro del panel de administración de Magento.

Para la realización de test unitarios del chatbot se ha hecho uso de Mocha²⁴ ya que viene predefinido como framework de testing en Microsoft Bot Framework. Para ello, se ha procedido a la instalación del módulo

²³ Microsoft (s.f). Microsoft Bot Framework. Microsoft. Recuperado el 4 de noviembre de 2019 de <https://dev.botframework.com/>

²⁴ Mocha (s.f.). Página Web Corporativa. Mocha. Recuperado el 4 de noviembre de 2019 de <https://mochajs.org>

mediante NPM y a la generación del código básico y la configuración necesaria para realizar los test unitarios con Mocha. Microsoft Bot Framework provee de un paquete para el testing de bots²⁵ llamado *bootbuilder-testing* con funciones para la realización de comparaciones, *assert* y validaciones en el bot, como se puede ver en el siguiente ejemplo:

```
const sut = new ShippingDialog();
const testClient = new DialogTestClient('msteams', sut);

let reply = await testClient.sendActivity('Checkout');
assert.strictEqual(reply.text, 'Name:');
reply = await testClient.sendActivity('Joe');
```

Para este proyecto se han creado los test básicos pero no se ha cubierto toda la funcionalidad dado alcance definido previamente.

²⁵ Microsoft (s.f). How to unit test bots. Microsoft Bot Framework. Recuperado el 2 de diciembre de <https://docs.microsoft.com/en-us/azure/bot-service/unit-test-bots?view=azure-bot-service-4.0&tabs=javascript>

17. Requisitos de instalación

Para poder interactuar con Magento 2 primero es necesario crear una instancia de este framework. El modo más habitual es desplegando este software en un servidor web para que pueda ser accedido a través de un navegador. Para ello, el servidor web debe contar con una serie de requisitos de instalación concretos. Para la versión a instalar (2.3.3) el servidor deberá contar como mínimo con (Magento, 2019 a.):

- Linux como SSOO
- 2Gb de RAM
- Composer
- Apache 2.4
- MySQL 5.6
- PHP 7.2

En el caso de este proyecto, la instancia de MageChatbot ha sido desplegada bajo los servicios de Microsoft Azure. Este modo de despliegue permite, de un modo más transparente y sencillo, crear los recursos necesarios para que el chatbot pueda funcionar en la web. En cualquier caso, para el funcionamiento del chatbot son necesarios los siguientes requisitos mínimos de hardware (Microsoft s.f. d.):

- Linux como SSOO
- 2Gb de RAM
- NodeJS > 10.10
- NPM
- Yarn

18. Instrucciones de instalación/implantación

Para poder interactuar con el chatbot en entornos reales, este se ha desplegado en un servidor que cumple los requisitos previamente mencionados. Una vez desplegado e iniciado el software se ha conectado a un cliente de mensajería para facilitar su uso, ya que es la puerta de entrada del usuario común.

Para esto, se ha optado por utilizar como infraestructura los servicios cloud de Microsoft Azure. Estos servicios cloud ofrecen varias ventajas para los desarrollos hechos con Microsoft Bot Framework²⁶:

- Permiten desplegar la aplicación en la nube de Azure con facilidad y sin coste. A su vez poseen una documentación muy detallada sobre la configuración de la instancia de Azure y los despliegues de software.
- Permiten conectar la instancia que contiene el chatbot con sistemas de mensajería mediante un asistente.

Configuración y creación de la infraestructura en Microsoft Azure

Las operaciones con Microsoft Azure se realizan desde el cliente de Azure para el terminal. Primero ha sido necesario iniciar sesión como usuario registrado de Microsoft Azure:

```
az login
```

Este comando abre la página de login de Microsoft Azure. Tras iniciar sesión se ha creado un registro de aplicación de Azure. Esto permite usar azure AD para autenticar usuarios ya que el chatbot hace uso de mensajes autenticados. En nuestro caso está basado en el nombre del chatbot.

```
az ad app create --display-name "Magechatbot" --password "password"  
--available-to-other-tenants
```

Este comando devuelve como respuesta un JSON con información relativa a la instancia creada. Entre los atributos generados destaca el `appId`, que se ha usado posteriormente para crear los despliegues.

Una vez generado el `appId` se puede crear la infraestructura necesaria para desplegar la aplicación. Para ello Microsoft hace uso de plantillas ARM. Estas plantillas se generan al crear un chatbot con Yeoman y

²⁶ Microsoft (s.f.). Tutorial de creación de un bot básico. Microsoft Azure. Recuperado el 2 de octubre de 2019 de <https://docs.microsoft.com/es-es/azure/bot-service/bot-builder-tutorial-basic-deploy>

definen todos los recursos que son necesarios crear para tener una aplicación funcional con Microsoft Bot Framework²⁷. Para este caso ha sido necesario crear un grupo de recursos que administre todos los recursos necesarios. Dentro de este grupo de recursos se ha creado un plan de servicio de aplicaciones, un servicio de aplicaciones y un registro de canales de bot. También ha sido necesario definir el área geográfica donde se desea crear este grupo de recursos.

```
az deployment create --name "MagechatbotDep" --template-file  
"template-with-new-rg.json" --location "centralus" --parameters appId="appID"  
appSecret="password" botId="Magechatbot" botSku=F0 newAppServicePlanName="AzureCloud"  
newWebAppName="Magechatbot" groupName="MagechatbotGr" groupLocation="centralus"  
newAppServicePlanLocation="centralus"
```

Tras ejecutar el comando en el panel de Azure se puede ver cómo ya se han creado los recursos necesarios y la infraestructura está preparada para realizar despliegues.

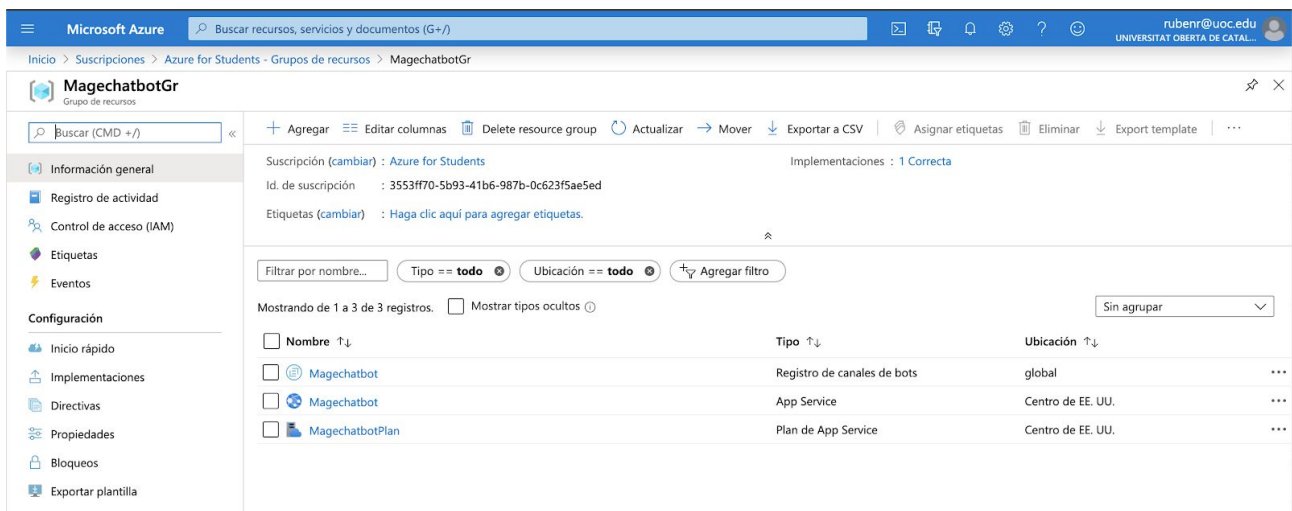


Figura 17: Listado de servicios configurados en Microsoft Azure.

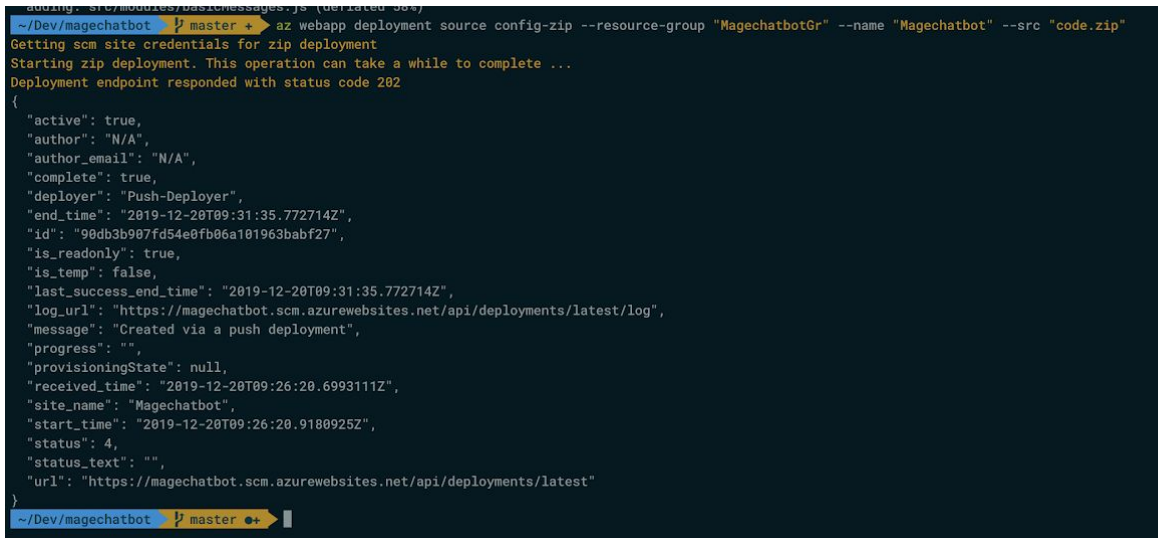
Para realizar despliegues en Microsoft Azure se ha generado un archivo de configuración web.config a nivel de aplicación mediante la consola de azure.

```
az bot prepare-deploy --code-dir "." --lang Javascript
```

²⁷ Microsoft (s.f.). Create a bot using Bot Framework SDK for JavaScript. Microsoft Azure. Recuperado el 2 de octubre de 2019 de <https://docs.microsoft.com/bs-latn-ba/azure/bot-service/javascript/bot-builder-javascript-quickstart?view=azure-bot-service-4.0>

Tras esto, ya se pueden realizar despliegues en la infraestructura de Azure mediante:

```
az webapp deployment source config-zip --resource-group "MagechatbotGr" --name  
"Magechatbot" --src "code.zip"
```



```
~/Dev/magechatbot master *  
az webapp deployment source config-zip --resource-group "MagechatbotGr" --name "Magechatbot" --src "code.zip"  
Getting scm site credentials for zip deployment  
Starting zip deployment. This operation can take a while to complete ...  
Deployment endpoint responded with status code 202  
{  
  "active": true,  
  "author": "N/A",  
  "author_email": "N/A",  
  "complete": true,  
  "deployer": "Push-Deployer",  
  "end_time": "2019-12-20T09:31:35.772714Z",  
  "id": "90db3b987fd54e0fb06a101963babf27",  
  "is_readonly": true,  
  "is_temp": false,  
  "last_success_end_time": "2019-12-20T09:31:35.772714Z",  
  "log_url": "https://magechatbot.scm.azurewebsites.net/api/deployments/latest/log",  
  "message": "Created via a push deployment",  
  "progress": "",  
  "provisioningState": null,  
  "received_time": "2019-12-20T09:26:20.6993111Z",  
  "site_name": "Magechatbot",  
  "start_time": "2019-12-20T09:26:20.9180925Z",  
  "status": 4,  
  "status_text": "",  
  "url": "https://magechatbot.scm.azurewebsites.net/api/deployments/latest"  
}
```

Figura 18: Ejecución en consola de un despliegue de Magechatbot.

Conexión de MageChatbot con Telegram

Una vez ejecutado el primer despliegue correctamente se puede conectar el chatbot a los clientes de mensajería en el recurso *Bot Channel Registration*. Aunque existe un asistente para conectar las instancias de chatbot de Microsoft Azure con los diferentes clientes de mensajería o aplicaciones de comunicación, cada uno de ellos tiene un método diferente. Para el caso de Telegram, la conexión se realiza mediante *Botfather*²⁸, un chatbot para generar chatbots. Permite, mediante conversación, crear, administrar y modificar tu propio chatbot.

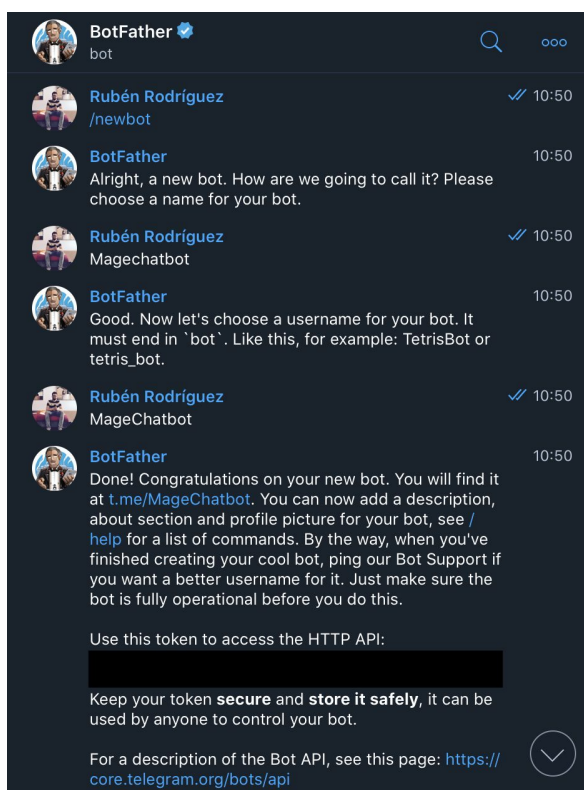


Figura 19: Ejemplo de creación de un chatbot para Telegram con BotFather.

Al generar un nuevo chatbot se ofrece un token para acceder a la API de comunicación de Telegram. Esta clave será la utilizada en el asistente de Azure para conectar con Telegram de modo transparente, por lo que no ha sido necesario crear un middleware que opere con la API de Telegram²⁹:

²⁸ Telegram (s.f.). Bots: An introduction for developers. Telegram. Recuperado el 2 de diciembre de 2019 de <https://core.telegram.org/bots>

²⁹ Telegram (s.f.). Telegram APIs. Telegram. Recuperado el 4 de noviembre de 2019 de <https://core.telegram.org/api>

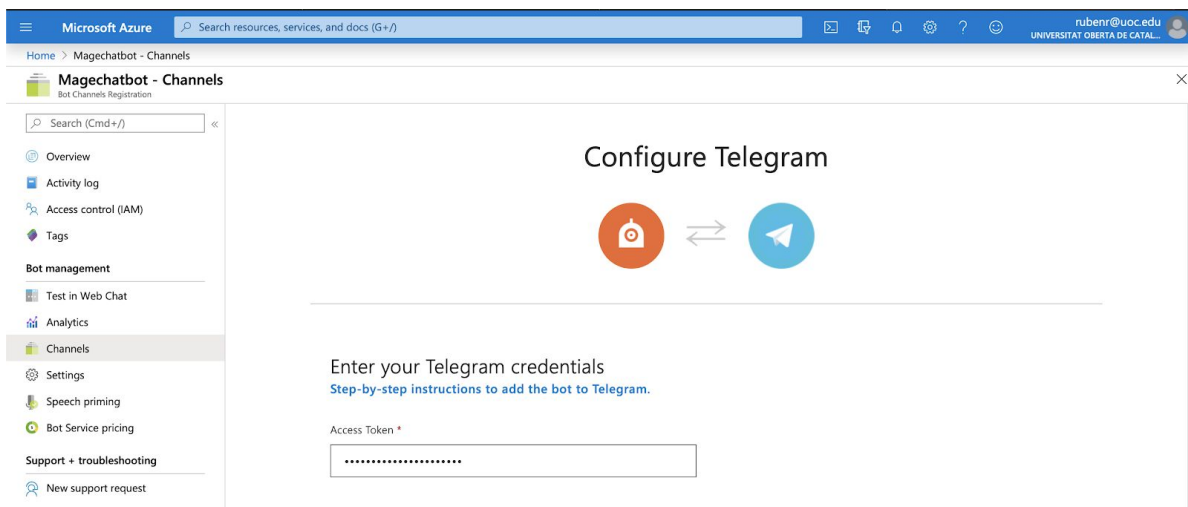


Figura 20: Pantalla de configuración de integración con Telegram en Microsoft Azure.

Tras esto, se puede probar el chatbot desde Telegram en la búsqueda de canales o mediante el enlace <https://t.me/Magechatbot>

19. Instrucciones de uso

El modo de uso de MageChatbot está basado en el funcionamiento de clientes de correo y su uso es similar tanto para el canal de Telegram como para el cliente web. Se puede usar MageChatbot mediante el canal creado de Telegram <https://t.me/MageChatbot> (Figura 21) o mediante el chatbot web alojado en <http://tfm.rubenr.es/magechatbot> (Figura 22). Una vez iniciado el bot y tras el saludo inicial, el chatbot te invita a realizar las acciones que desees: buscar un producto, reiniciar el chatbot o mostrar información de ayuda

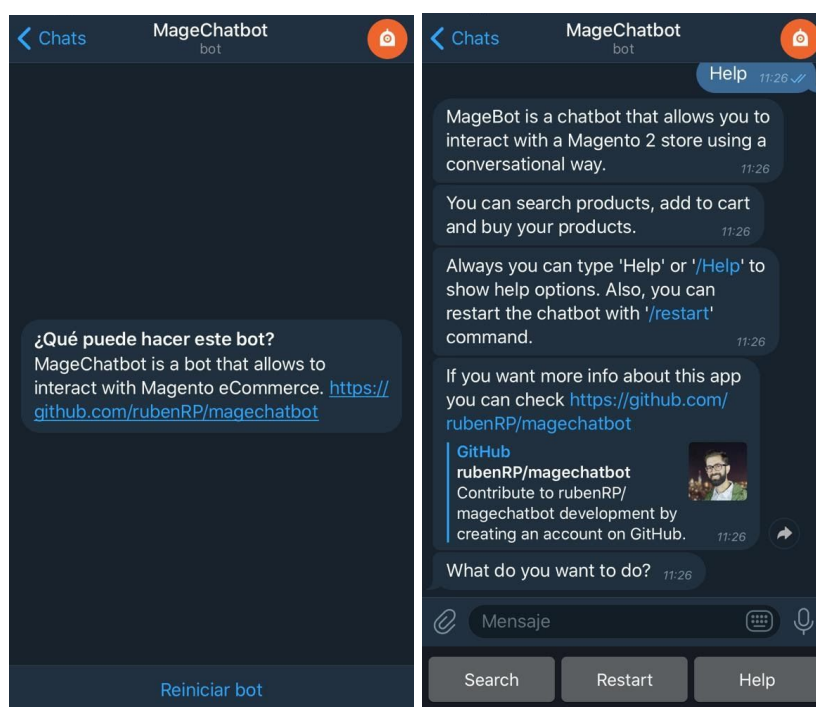


Figura 21: Ejemplos de conversación inicial de MageChatbot en Telegram.

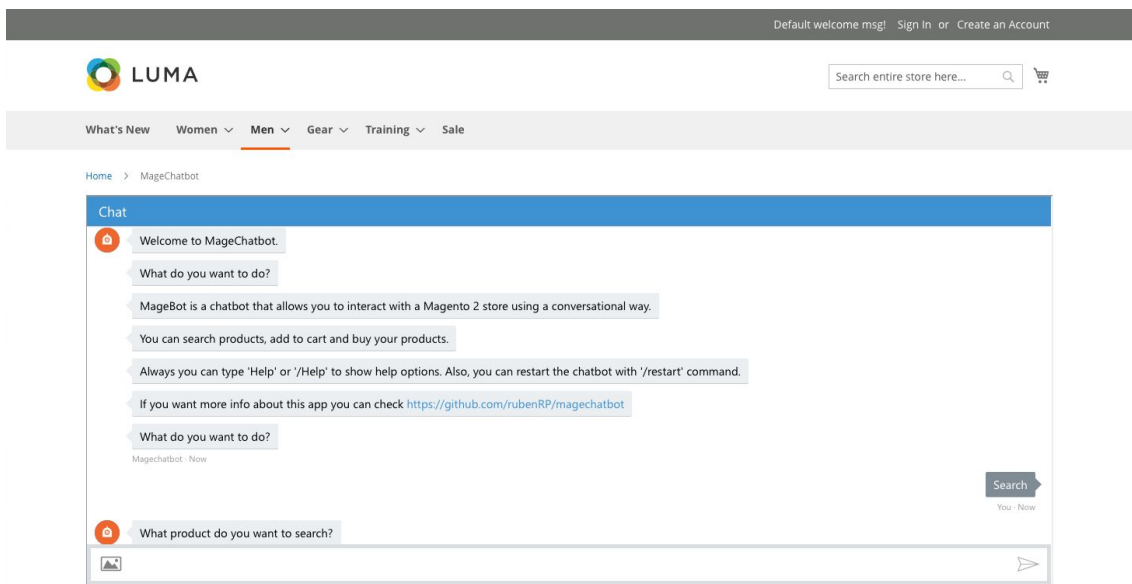


Figura 22: Ejemplo de conversación inicial de MageChatbot en el cliente web.

Cuando el usuario encuentra el producto deseado sólo tiene que añadirlo al carrito. Una vez añadido un producto al carrito se muestra el resumen del pedido y si se quiere continuar con el proceso de compra o seguir buscando productos (Figura 23). Este paso se puede repetir las veces que se desee.

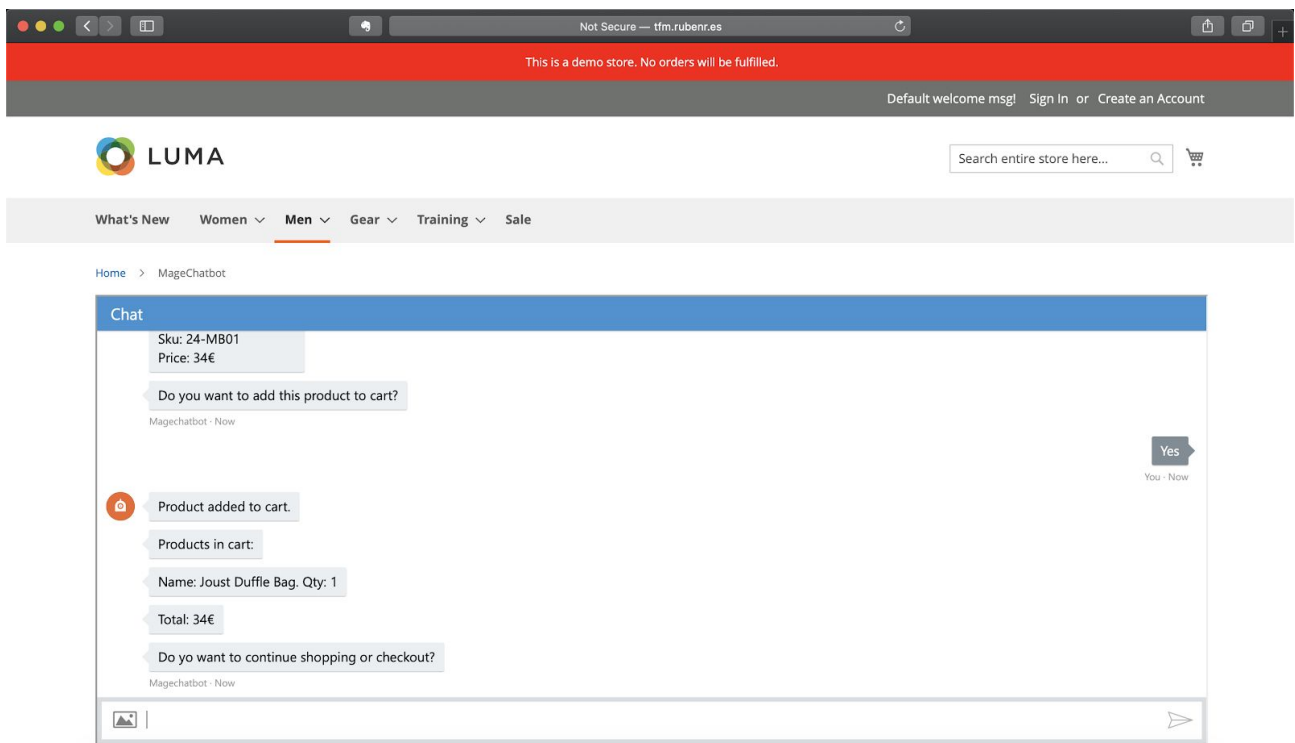


Figura 23: Ejemplo de información y resumen de MageChatbot en el cliente web.

Una vez añadidos todos los productos se puede seguir con el proceso de pago. En este paso se solicitan los datos de envío (Figura 24). Para el MVP se han simplificado los pasos y sólo se ha habilitado tanto el envío gratis a España como los gastos contrareembolso.

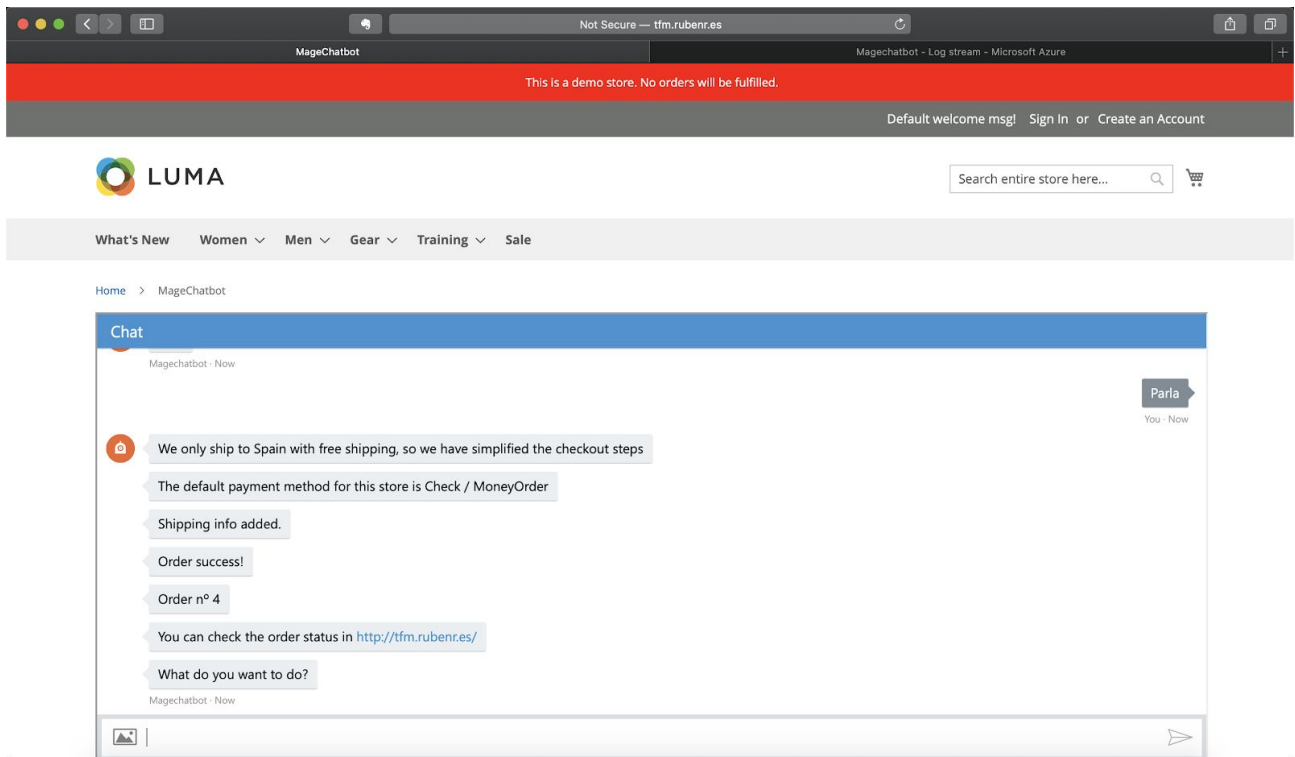


Figura 24: Ejemplo del resumen del pedido de MageChatbot en el cliente web.

Una vez rellenados todos los datos correctamente se procede a la creación del pedido. El estado del pedido se puede revisar en el área de administración de Magento 2 (Figura 25).

The screenshot shows the MageChatbot interface for order #000000004. The interface includes a sidebar with navigation options and a main content area displaying order details.

Order ID: #000000004

Billing Address: Rubén Rodríguez, Fake Street 123, Parla, Madrid, 28981, Spain, T: 12312313

Shipping Address: Rubén Rodríguez, Fake Street 123, Parla, Madrid, 28981, Spain, T: 12312313

Payment & Shipping Method:

Payment Information: Bank Transfer Payment. The order was placed using EUR.

Shipping & Handling Information: Free Shipping - Free €0.00

Items Ordered:

Product	Item Status	Original Price	Price	Qty	Subtotal	Tax Amount	Tax Percent	Discount Amount	Row Total
Joust Duffle Bag	Ordered	€34.00	€34.00	Ordered 2	€68.00	€0.00	0%	€0.00	€68.00

SKU: 24-MB01

Figura 25: Resumen del pedido realizado con MageChatbot

20. Bugs

Durante el desarrollo de la aplicación se han detectado los siguientes problemas. En su mayoría, derivados del tipo de servidor utilizado y del cliente de mensajería que se ha decidido implementar.

Imágenes en producción

El servidor de Digital Ocean que aloja la instancia de Magento 2 no tiene https, por lo que tanto en el canal de Telegram como en el cliente web no se pueden visualizar las imágenes del producto. En el entorno local se ha verificado que funciona correctamente con una instancia. Esto es reportado en la documentación de Microsoft Bot Framework³⁰

POST to Bot timeout after 15s en Telegram

Para proteger Telegram, la aplicación bloquea las peticiones que duran más de 15 segundos. Debido a que la API de Telegram se conecta con la aplicación en Microsoft Azure y esta con la API de Magento, se generan colas de peticiones que duran más de 15 segundos y por tanto se bloquean las peticiones. Esto imposibilita obtener resultados de búsqueda en Telegram con la infraestructura actual. Estos problemas han sido documentados en Github³¹ ³²y Stack Overflow³³. Aunque las soluciones temporales propuestas pasan por eliminar la condición *await* de todas las peticiones asíncronas de tipo *async/await*, en el caso de MageChatbot no es posible ya que interactúa con la API de Magento y es necesario esperar a la respuesta para seguir operando dentro de la aplicación.

Caducidad de las API keys de Magento

A lo largo del desarrollo ha sido necesario regenerar las keys de integración de Magento 2 a pesar de no tener periodo de caducidad. La solución ha sido regenerar las claves, modificarlas en el fichero de configuración de MageChatbot y forzar otro despliegue. Se desconoce el origen del problema, pero al encontrarse en el área de Magento 2, está fuera del alcance del proyecto.

³⁰ Microsoft (s.f.). Incorporación de elementos multimedia a los mensajes. Microsoft Azure. Recuperado el 10 de octubre de 2019 de <https://docs.microsoft.com/es-es/azure/bot-service/bot-builder-howto-add-media-attachments?view=azure-bot-service-4.0&tabs=javascript>

³¹ Nwhitmont(s.f.). [Feature Request] User-configurable Direct Line timeout duration. botframework-sdk. GitHub Recuperado el 2 de diciembre de 2019 de <https://github.com/microsoft/botframework-sdk/issues/3220>

³² RodrigoRVieira (s.f.). [Question] How to suppress and/or increase the 15 seconds timeout message while posting to our messages endpoint?. botframework-sdk. GitHub Recuperado el 2 de diciembre de 2019 de <https://github.com/microsoft/botframework-sdk/issues/3122>

³³ Tamarasaraman, S. (2019), POST to Bot timed out after 15s. Stack Overflow. Recuperado el 15 de diciembre de 2019 de <https://stackoverflow.com/questions/58409906/post-to-bot-timed-out-after-15s>

21. Proyección a futuro

Una de las decisiones tomadas para el desarrollo del proyecto ha sido que este fuera Open Source, esta es una de las razones por las que ha sido publicado en GitHub. De este modo, la comunidad de desarrolladores puede usar el código, modificarlo o contribuir para hacerlo más completo. Dentro de los pasos a seguir, destacan los siguientes:

- Incorporación de envío en el proceso de *checkout*.
- Incorporación de métodos de pago al proceso de *checkout*.
- Posibilitar los envíos a diferentes países en el *checkout*.
- Añadir la posibilidad de tener diferentes direcciones de envío y facturación
- Visualización y selección de productos configurables (productos con talla y color, por ejemplo).
- Acceso como usuario registrado
- Incorporación de funcionalidad de favoritos (*wishlist*).

Otra posible rama para el desarrollo del proyecto es la incorporación de sistemas para el reconocimiento natural del lenguaje. Para este propósito Microsoft provee de sistemas para complementar y evolucionar el bot mediante LUIS (*Language Understanding*)³⁴. Esos sistemas pueden proveer de un valor añadido a la aplicación y una mejora en la usabilidad de la misma.

Para cumplir con estos objetivos se crearán tickets en GitHub para que los desarrolladores puedan colaborar en el desarrollo dando forma a una versión final y estable del chatbot. Mediante la gestión de *pull request* e *issues* se dará continuidad al proyecto en la forma en la que los desarrolladores decidan. La evolución del proyecto se podrá ver en la página de GitHub de MageChatbot

<https://github.com/rubenrp/magechatbot>.

³⁴ Microsoft (s.f.). Use multiple LUIS and QnA Models. Microsoft Bot Framework. Recuperado el 2 de octubre de 2019 de <https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-tutorial-dispatch?view=azure-bot-service-4.0&tabs=cs>

22. Conclusiones

Como se ha comentado a lo largo de diferentes secciones del documento, el objetivo principal de este proyecto ha sido el desarrollo de un chatbot que permite conectarse a una instancia de Magento 2 mediante el uso de REST API permitiendo la compra de modo conversacional.

Atendiendo a los objetivos inicialmente, y tras el desarrollo del proyecto se puede establecer que:

- Se ha implementado un nuevo modelo de interacción entre usuario y ecommerce a través del desarrollo de un chatbot que puede conectarse a una plataforma de ecommerce vía API.
- Se ha desarrollado una aplicación que facilita la realización de diferentes acciones por parte de usuario a través de un sistema en el que el chatbot permite ejecutar un flujo básico de compra en el ecommerce (búsqueda, detalle, carrito y compra).
- Se ha creado la infraestructura necesaria para implementar el chatbot en un entorno de producción
- Se ha integrado dicho chatbot con una plataforma de mensajería instantánea y en un cliente web, para que este sea accesible para el usuario

Teniendo en cuenta los objetivos previos, se pueden establecer las siguientes conclusiones como resultado del desarrollo del proyecto:

- Es posible la interacción mediante API con un ecommerce a través del despliegue de una instancia de Magento 2 y el uso de un *droplet* de Digital Ocean.
- La creación de un chatbot funcional que incluya integraciones complejas se simplifica con el uso de un framework como Microsoft Bot Framework.
- Es posible la integración de una API como la de Magento 2 en un chatbot mediante un conector genérico o un módulo de NodeJS.
- Los hechos anteriormente citados permiten generar la lógica necesaria para para realizar búsquedas por el catálogo de Magento 2, añadir al carrito y realizar la compra de productos.
- Es posible hacer accesible al usuario un chatbot en clientes de mensajería instantánea como Telegram mediante la integración de servicios de Microsoft Azure.

23. Bibliografía

Almansor, E. & Hussain, F. (2019). Survey on Intelligent Chatbots: State-of-the-Art and Future Research Directions. 10.1007/978-3-030-22354-0_47.

Arunasalam, S. & Good, A. (2013). Implications for Improving Accessibility to E-Commerce Websites in Developing Countries - A Study of Hotel Websites.

Audience, Drift, MyClever & Salesforce, (2018). 2018 State of Chatbots Report. Recuperado el 4 de noviembre de 2019 de <https://www.drift.com/wp-content/uploads/2018/01/2018-state-of-chatbots-report.pdf>

Axios (s.f.). Axios: GitHub. Recuperado el 2 de septiembre de 2019 de <https://github.com/axios/axios>

Charlton, G. (2018). 17 tips for a smooth ecommerce checkout. SaleCycle. Recuperado el 16 de septiembre de 2019 de <https://www.salecycle.com/blog/strategies/17-tips-for-a-smooth-ecommerce-checkout-process/>

Corrigan, K. (2019). Checkout Page. Oberlo. Recuperado el 16 de septiembre de 2019 de <https://www.oberlo.com/ecommerce-wiki/checkout-page>

Cowderoy, A. (2001). Measures of Size and Complexity for Web-Site Content. Recuperado el 7 de noviembre de 2019 de <https://pdfs.semanticscholar.org/8786/e588e3718abae5fd4df9e226d955978a0296.pdf>

Devaney, E. (2016), Infographic: Rise of the Chatbots. Drift. Recuperado el 10 de octubre de 2019 de <https://www.drift.com/blog/chatbots-infographic/>

Digital Ocean (s.f.). Página Web Corporativa. Digital Ocean. Recuperado el 10 de octubre de 2019 de <https://www.digitalocean.com>

DivanteLtd (s.f.) Magento 2 REST client. GitHub. Recuperado el 2 de octubre de 2019 de <https://github.com/DivanteLtd/magento2-rest-client>

DivanteLtd (s.f.) Vue Storefront. GitHub. Recuperado el 2 de octubre de 2019 de <https://github.com/DivanteLtd/vue-storefront>

ESLint (s.f.). Página Web Corporativa. ESLint. Recuperado el 4 de noviembre de 2019 de <https://eslint.org>

Facebook (2019). Facebook Messenger. Facebook. Recuperado el 4 de noviembre de 2019 de <https://es-la.facebook.com/messenger/>

Github (s.f.).Página web corporativa. GitHub. Recuperado el 2 de diciembre de 2019 de <https://github.com>

Göksel, N. & Mutlu, M. (2016). On the track of Artificial Intelligence: Learning with Intelligent Personal Assistants. International Journal of Human Sciences. 13. 592. 10.14687/ijhs.v13i1.3549.

Gupta, S. & Borkar, D. & De Mello, C. & Patil, S. (2015). An E-Commerce Website based Chatbot: (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6.

Intercom, (2016). 8 principles of bot design. Medium. Recuperado el 24 de octubre de 2019 de <https://medium.com/intercom-inside/8-principles-of-bot-design-51f03df1d84c>

Jest (s.f.). Página Web corporativa. Jest - delightful JavaScript Testing. Recuperado el 10 de septiembre de 2019 de <https://jestjs.io/>

Kovacevic, J. (2019). Magento 2 Customer Groups Configuration. Inchoo. Recuperado el 4 de noviembre de 2019 de <https://inchoo.net/magento-2/magento-2-customer-groups/>

Magento. (2019). Technology stack. Magento DevDocs. Recuperado el 22 de octubre de 2019 de <https://devdocs.magento.com/guides/v2.3/architecture/tech-stack.html>

Magento. (2019). Adobe (Magento) Named a Leader in 2019 Magic Quadrant for Digital Commerce. Adobe | Magento Commerce. Recuperado el 22 de octubre de 2019 de <https://magento.com/resources/adobe-named-leader-2019-magic-quadrant-digital-commerce>

Magento (s.f.). Extensibility and modularity. Recuperado el 22 de octubre de <https://devdocs.magento.com/guides/v2.3/architecture/extensibility.html>

Magento (s.f.). Meaningful Commerce Experiences. Adobe | Magento Commerce. Recuperado el 4 de noviembre de 2019 de <https://magento.com/products/magento-commerce/features-meaningful-commerce-experiences>

Magento (s.f.). REST API reference. Magento DevDocs. Recuperado el 22 de octubre de 2019 de <https://devdocs.magento.com/redoc/2.3/>

Magento (s.f.). Search for product with the /search endpoint. Magento DevDocs. Recuperado el 4 de noviembre de 2019 de <https://devdocs.magento.com/guides/v2.3/rest/search-endpoint.html>

Matveev, A. (2018). Magento 2 Module-based Architecture. Belvg. Recuperado el 4 de noviembre de 2019 de <https://belvg.com/blog/magento-2-module-based-architecture.html>

Microsoft (s.f.). Biblioteca de Diálogos. Microsoft Azure. Recuperado el 15 de noviembre de 2019 de <https://docs.microsoft.com/es-es/azure/bot-service/bot-builder-concept-dialog>

Microsoft (s.f.). BotBuilder-Samples. GitHub. Recuperado el 4 de octubre de 2019 de <https://github.com/microsoft/BotBuilder-Samples>

Microsoft (s.f.). Create a bot using Bot Framework SDK for JavaScript. Microsoft Azure. Recuperado el 2 de octubre de 2019 de <https://docs.microsoft.com/bs-latn-ba/azure/bot-service/javascript/bot-builder-javascript-quickstart?view=azure-bot-service-4.0>

Microsoft (s.f.). General data protection regulation. Microsoft | Bot Framework. Recuperado el 10 de diciembre de 2019 de <https://blog.botframework.com/2018/04/23/general-data-protection-regulation-gdpr/>

Microsoft (s.f.). How to unit test bots. Microsoft Bot Framework. Recuperado el 2 de diciembre de <https://docs.microsoft.com/en-us/azure/bot-service/unit-test-bots?view=azure-bot-service-4.0&tabs=javascript>

Microsoft (s.f.). Incorporación de elementos multimedia a los mensajes. Microsoft Azure. Recuperado el 10 de octubre de 2019 de <https://docs.microsoft.com/es-es/azure/bot-service/bot-builder-howto-add-media-attachments?view=azure-bot-service-4.0&tabs=javascript>

Microsoft (s.f.). Microsoft Bot Framework. Microsoft. Recuperado el 4 de noviembre de 2019 de <https://dev.botframework.com/>

Microsoft (s.f.). Página web corporativa. Microsoft Azure. Recuperado el 27 de septiembre de 2019 de <https://azure.microsoft.com/es-es/>

Microsoft (s.f.). Save user and conversation data. Microsoft Azure. Recuperado el 4 de noviembre de 2019 de <https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-howto-v4-state>

Microsoft (s.f.). Tutorial de creación de un bot básico. Microsoft Azure. Recuperado el 2 de octubre de 2019 de <https://docs.microsoft.com/es-es/azure/bot-service/bot-builder-tutorial-basic-deploy>

Microsoft (s.f.). Use multiple LUIS and QnA Models. Microsoft Bot Framework. Recuperado el 2 de octubre de 2019 de <https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-tutorial-dispatch?view=azure-bot-service-4.0&tabs=cs>

Mocha (s.f.). Página Web Corporativa. Mocha. Recuperado el 4 de noviembre de 2019 de <https://mochajs.org>

Moore K. (2018). Ecommerce 101 + The History of Online Shopping: What The Past Says About Tomorrow's Retail Challenges. Big Commerce. Recuperado el 4 de noviembre de 2019 de <https://www.bigcommerce.com/blog/ecommerce/>

Nelson, R. (2017). 10 Case Studies on Chatbots. Overthink Group. Recuperado el 19 de septiembre de 2019 de <https://overthinkgroup.com/chatbot-case-studies/>

Nizam, N. (2018). What UX Should Learn From WhatsApp. DZone. Recuperado el 4 de noviembre de 2019 de <https://dzone.com/users/3222906/nazreenzm.html>

Nwhitmont(s.f.). [Feature Request] User-configurable Direct Line timeout duration. botframework-sdk. GitHub. Recuperado el 2 de diciembre de 2019 de <https://github.com/microsoft/botframework-sdk/issues/3220>

OpenJS Foundation (s.f.). Página web corporativa. Node.js. Recuperado el 15 de diciembre de 2019 de <https://nodejs.org/es/>

Phillips, C. (2018). The 3 Types of Chatbots & How to Determine the Right One for Your Needs. Chatbots Magazine. Recuperado el 4 de diciembre de 2019 de <https://chatbotsmagazine.com/the-3-types-of-chatbots-how-to-determine-the-right-one-for-your-needs-a4df8c69ec4c>

Politibot (s.f.) Politibot. Recuperado el 6 de octubre de 2019 de <https://politibot.io>

Resnick, N. (2018). How to turn anonymous visitors into known customers. Sourcify. Recuperado el 4 de noviembre de 2019 de <https://www.sourcify.com/turn-anonymous-visitors-known-customers/>

RodrigoRVieira (s.f.). [Question] How to suppress and/or increase the 15 seconds timeout message while posting to our messages endpoint?. botframework-sdk. GitHub Recuperado el 2 de diciembre de 2019 de <https://github.com/microsoft/botframework-sdk/issues/3122>

Rodríguez, R.(s.f.). MageChatbot GitHub. Recuperado el 2 de enero de 2020 de <https://github.com/rubenRP/magechatbot>

Rodriguez, R. (s.f.). Magento Community. Swagger. Recuperado el 2 de enero de 2020 de <http://fm.rubenr.es/swagger>

Rouse, M. (2017). Virtual private server (VPS) or virtual dedicated server (VDS). Search Server Virtualization. Recuperado el 16 de diciembre de 2019 de <https://searchservervirtualization.techtarget.com/definition/virtual-private-server>

Sambhanthan, A., & Good, A. (2013). Implications for Improving Accessibility to E-Commerce Websites in Developing Countries: A Subjective Study of Sri Lankan Hotel Websites. ArXiv, abs/1302.5198.

Shadid, S. (2018). Top Ecommerce Platforms Market Share in 2018. Cloud Ways Recuperado el 15 de diciembre de 2019 de <https://www.cloudways.com/blog/top-ecommerce-platforms/>

Sohaib, O. & Kang, K.. (2017). E-Commerce Web Accessibility for People with Disabilities. 10.1007/978-3-319-52593-8_6.

Tamilarasan, S. (2019), POST to Bot timed out after 15s. Stack Overflow. Recuperado el 15 de diciembre de 2019 de <https://stackoverflow.com/questions/58409906/post-to-bot-timed-out-after-15s>

Telegram (s.f.). Bots: An introduction for developers. Telegram. Recuperado el 2 de diciembre de 2019 de <https://core.telegram.org/bots>

Telegram (s.f.). Telegram APIs. Telegram. Recuperado el 4 de noviembre de 2019 de <https://core.telegram.org/api>

The Engine Room (s.f.).Messaging Apps. The Engine Room Library. Recuperado el 15 de diciembre de 2019 de <https://library.theengineroom.org/messaging-apps/>

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59, 433–460

Uberdev (2019). The current state of GraphQL in Magento 2.3. Ubertheme. Recuperado el 17 de octubre de 2019 de https://www.ubertheme.com/magento-news/magento_2_graphql_api/

Volodymyr, T. (2018). Building a Stable Node.js Project Architecture. *Best Practices for Node.js Development* (2018). Apiko. Recuperado el 17 de septiembre de 2019 de <https://apiko.com/blog/node-js-architecture-tips/>

Walde, A. (2018). The state of accessibility in e-commerce. *Medium*. Recuperado el 17 de septiembre de 2019 de <https://medium.com/@lsnrae/the-state-of-accessibility-in-e-commerce-fc97b9bad3a2>

WhatsApp Inc.(2019). Página web Corporativa. WhatsApp. Recuperado el 4 de noviembre de 2019 de <https://www.whatsapp.com>

Yúbal, F (2017). Así era ELIZA, el primer bot conversacional de la historia. *Xataka*. Recuperado el 4 de octubre de 2019 de <https://www.xataka.com/historia-tecnologica/asi-era-eliza-el-primer-bot-conversacional-de-la-historia>

24. Glosario

Chatbot: Programa informático diseñado para simular una conversación con usuarios humanos, especialmente a través de internet.

API Endpoint: Describe un punto por el cual una API permite comunicarse con otro software.

Framework: Esquema o estructura que se establece y que se aprovecha para desarrollar y organizar un software determinado.

REST: Estilo de arquitectura para diseñar aplicaciones en red.

SEO: *Search Engine Optimization*. Conjunto de acciones orientadas a mejorar el posicionamiento de un sitio web.

SSH: *Secure Shell*. Protocolo para el acceso remoto a un servidor.

Swagger: Software que ayuda a los desarrolladores a diseñar y documentar servicios web RESTful.

VPS: *Virtual Private Server*. Es un modo de particionar un servidor físico dando lugar a varios servidores “virtuales” que se comportan como si de un servidor físico se tratara.