Grado en Ingeniería Informática

Trabajo Final de Grado

Desarrollo de chatbots en la plataforma de Facebook

Autor:
Adrian Ionut harabagiu

Supervisor:
Fernando Gregori Peréz
Tutor académico:
Rafael Berlanga Llavori

Fecha de lectura: 14 de Septiembre de 2017
Curso académico 2016/2017

Agradecimientos

Primero me gustaría agradecer a los compañeros de la empresa Paynopain por su colaboración. Me habéis apoyado muchísimo y siempre me habéis ayudado cuando lo necesitaba. Particularmente me gustaría nombrar a mi supervisor en Paynopain, Fernando Gregori Peréz y Juanjo Chust. Me gustaría agradeceros el tiempo que me habéis dedicado y todas esas veces que me habéis ayudado, sacándome más de un apuro con la realización de este proyecto. Además, me gustaría dar las gracias a mi tutor Rafael Berlanga Llavorí por transmitirme su entusiasmo por los chatbots y por ofrecerme esta oportunidad de aprender más sobre los chatbots con Node.js. También me gustaría agradecer a mis padres por sus sabios consejos y su comprensión. Siempre habéis estado ahí para mí. Finalmente, a mis amigos que me han hecho olvidar esos días complicados y desconectar de la rutina.

Resumen

Este proyecto de fin de grado, está basado en la investigación y el desarrollo de un chatbot que permita buscar y comprar productos, escribiendo en un simple chat de cualquier aplicación de mensajería y que el chatbot responda con los resultados como si fuera un usuario más. En este caso la aplicación de mensajería propuesta por la empresa ha sido Facebook y como lenguaje de programación he utilizado Node.js. Para conseguir llevar a cabo este proyecto he tenido que implementar una función básica que pueda entender las peticiones del usuario, conectar la aplicación con diferentes APIs (Application Programming Interface) y usar la estructura hexagonal para el futuro mantenimiento y testeo del proyecto.

Palabras clave

API REST, chatbot, Facebook, Node.js

Keywords

API REST, chatbot, Facebook, Node.js

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El contexto de las practicas ha sido en la empresa PaynoPain Solutions ubicada en el Espaitec de la Universidad Jaume I de Castellón. Es una empresa basada en el desarrollo tecnológico de servicios, aplicaciones informáticas y soluciones de pago móvil. Desde el principio de la estancia he podido observar que es una empresa que busca nuevas alternativas tecnológicas a los servicios actuales. Este proyecto es un claro ejemplo de una innovación tecnológica aparecida recientemente, que son los ChatBots. Estos chatbots son programas informáticos con los que es posible mantener una conversación a través del chat de aplicaciones de mensajería como: Telegram, Facebook, Skype etc. En este proyecto se va a diseñar un chatbot que intenta solucionar las necesidades de la gente a la hora de efectuar compras desde terminales pequeños, de una forma simple, rápida y segura mediante la aplicación de Messenger de Facebook que ya está instalada en millones de terminales. Como el chatbot está conectado a una página de Facebook, los millones de usuarios de esta red social pueden acceder a dicha página de una empresa, empezar a chatear con el chatbot y efectuar las compras necesarias sin salir de la red social, Facebook.

1.2. Objetivos del proyecto

Para lograr la meta fijada en el desarrollo del proyecto se han planteado los siguientes objetivos de trabajo.

- Objetivo general: Desarrollar un chatbot que pueda entender las peticiones escritas de los usuarios y devuelva una lista con los productos que coincidan con las indicaciones del usuario. Además, se propone también ofrecer la opción de compra de los productos elegidos por el usuario directamente desde la aplicación de Messenger de Facebook.

- Objetivos específicos:

    1. Estudio de la documentación del lenguaje de programación Node.js y de las funciones de los diferentes APIs usadas en el proyecto (Facebook API, Catalogo API, PayLands API).
    2. Instalación y configuración del framework Node.js, usando la plantilla propuesta por Facebook para este lenguaje de programación.
    3. Instalación y configuración de GIT, como herramienta de control de versiones conectada con GitLab(plataforma ofrecida por la empresa para alojar las versiones del proyecto), para controlar los cambios hechos en el código del proyecto.
    4. Enlazar el chatbot con la plataforma de Facebook, para conseguir un canal de entrada de peticiones y de salida de resultados de dichas peticiones.
    5. Adaptación de la estructura hexagonal, que se explica en el apartado 2.1 del proyecto, para agilizar el mantenimiento posterior del chatbot.
    6. Conectar el chatbot con un API catálogo, ofrecido por la empresa, para poder buscar productos.
    7. Desarrollo de una función básica para entender las peticiones de los usuarios y poder asignarlas a diferentes funciones programables.
    8. Desarrollo del carrito de compra y la base de datos en MongoDB donde guardar tanto el carrito temporal como los pedidos de los usuarios.
    9. Conectar el chatbot con la API de pago, Paylands para hacer las compras directamente desde el chatbot.

1.3. Estructura de la memoria

En el capítulo 2 se muestran las diferentes tecnologías usadas en el proyecto. El capítulo 3 describe la metodología, planificación y el seguimiento que se ha realizado del proyecto. El capítulo 4 presenta el análisis del sistema(casos de uso, requisitos de datos y requisitos de usuario), el diseño de la arquitectura del sistema y el diseño de la interfaz. En el capítulo 5 encontramos los detalles de implementación, la verificación y la validación del proyecto. En el capítulo 6 se describen las conclusiones personales, académicas o profesionales y las posibles extensiones del proyecto. Finalmente acabamos con una bibliografía donde están las citas o menciones a diferentes fuentes y un anexo con diferentes capturas del funcionamiento del chatbot.

Capítulo 2

Descripción del proyecto

Como se ha mencionado en la introducción, el proyecto planteado por la empresa con-sistió en la implementación de un chatbot para consultar catálogos y realizar compras desde Messenger mediante conversaciones. El proyecto ha seguido la metodología de trabajo Modelo-Vista-Controlador (MVC), más concretamente la arquitectura hexagonal, la cual se describirá en una sección posterior ya que es una arquitectura utilizada en la empresa PaynoPain.

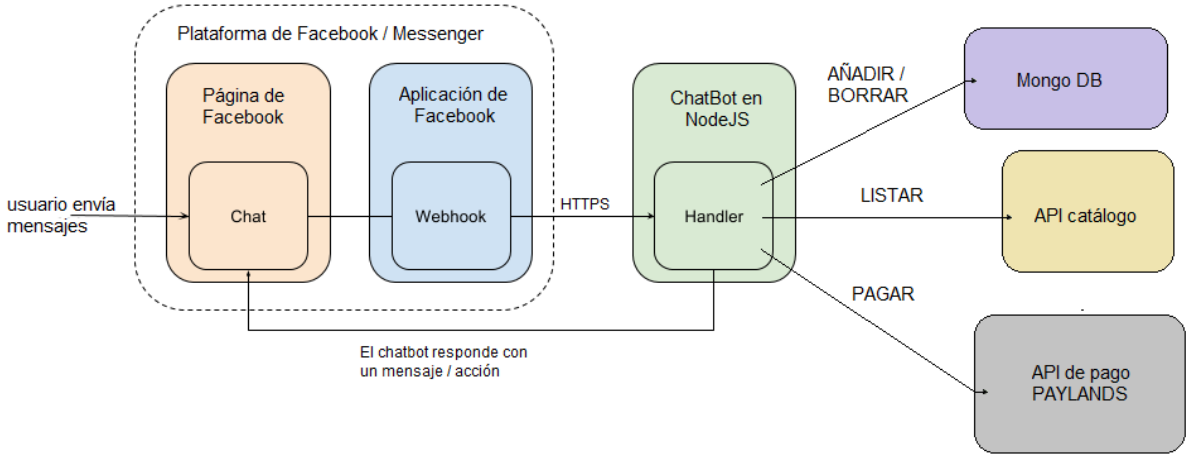


Figura 2.1: Modelado del sistema

Tal como se aprecia en la Figura 2.1(fuente: [10]), el usuario escribe un mensaje en el chat de Facebook. Ese mensaje se envía a través del webhook por un túnel HTTPS gracias a la herramienta NGrok, hasta mi API donde esta implementado el chatbot. Una vez en la API, si el mensaje corresponde a la acción de 'añadir' o 'borrar' un producto al carrito se ejecutarán las funciones de borrado o inserción del producto en la base de datos de MongoDB. Si el mensaje corresponde a la acción de 'pagar' se efectúa la redirección a la pagina web de plataforma de pago, donde se introducirán los datos de la tarjeta y se procederá al pago. Dependiendo si la transacción se efectúa correctamente o con errores se muestra un mensaje tanto en la pagina web de la plataforma de pago como en el chat de Facebook del usuario. Finalmente si el

mensaje corresponde a un texto escrito por el usuario, se obtiene el nombre del producto deseado mediante la función basicAI(texto) y se busca dicho producto en la API catálogo. Después la API catálogo devuelve una lista con los productos que coinciden con el producto buscado en formato JSON y luego se modifica a mensaje estructurado según las especificaciones del API de Facebook y se envía de vuelta al chat del usuario.

2.1. La arquitectura de diseño Hexagonal

Para mejorar el futuro mantenimiento del proyecto se utiliza la estructura hexagonal. Tal como se aprecia en la Figura 2.2 (fuente:[5]), dicha estructura efectúa una división entre el frontend que ve el usuario y el backend de la aplicación. Es una estructura derivada del patrón MVC que internamente se divide en casos de uso, gateways, entidades, etc. con el objetivo de separar, la lógica de los casos de uso y los accesos (insertar, modificar, borrar) tanto al backend como a las APIs externas. Además, es esencial para la validación, ya que permite hacer tests de la lógica y de los accesos por separado. Para poder implementar las entidades utilizadas en la estructura hexagonal, es necesario utilizar tipado y objetos basados en clases, ya que está especialmente diseñado para el desarrollo en Java. Para este proyecto, al utilizar JavaScript como lenguaje de desarrollo se requirió adaptar el código a Typescript y ECMAScript6 tal como se cita en [7].

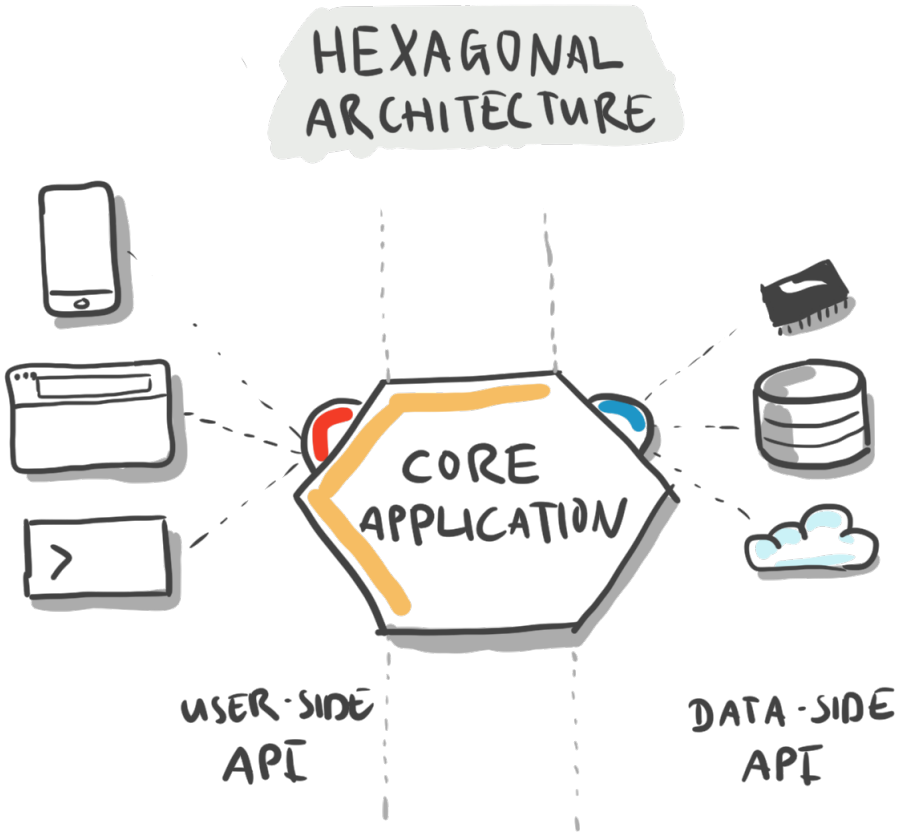


Figura 2.2: Diagrama Estructura Hexagonal

2.2. Tecnología utilizada en el proyecto

En cuanto a la situación inicial del proyecto, hay varias tecnológicas que son requeridas por la empresa PaynoPain como el uso de Node.js y Facebook Messenger. Node.js es el lenguaje más adoptado para la implementación de chatbots. Por otro lado, a plataforma de Facebook Messenger goza de gran popularidad entre los consumidores de chatbots.

Un requisito básico de un chatbot es que debe procesar muchas peticiones a la vez, desde muchos usuarios. Es por ello que el uso de lenguajes asíncronos como Node.js junto con el uso de 'promesas' [4], es la solución ampliamente utilizada en este tipo de implementaciones.

El framework Node.js permite la instalación de múltiples paquetes de software ofrecidos en el repositorio NPM. Concretamente en este proyecto se utilizaron los paquetes: mcrypt, express, http, request, entre otros.

La librería MCrypt es clave en este proyecto, ya que es necesaria para codificar las peticiones enviadas por POST desde mi API al API catálogo y para decodificar las respuestas devueltas que contendrán los productos que estábamos buscando. El algoritmo de cifrado requerido por la API catálogo es Rijndael-256.

La librería express y http se utiliza para crear un servidor web HTTP (Hypertext Transfer Protocol), para poder conectar con la API de Facebook. La librería request se utiliza tanto para conectar con APIs externas como para obtener o enviar datos a dichas APIs.

Para poder cumplir los requerimientos de la empresa decido utilizar una plantilla de ejemplo para la creación de chatbots en la plataforma de Messenger en lenguaje Node.js [8]. Con dicha plantilla consigo enviar tanto mensajes en formato texto como estructurados (lista, carrusel, botones, etc.) a un chat de Messenger que está enlazado a una página de Facebook.

Además, uso las APIs externas ofrecidas por la empresa PaynoPain, como la API CATÁLOGO que se utiliza para buscar y devolver al usuario diversas categorías o productos y la API DE PAGO (Paylands) que ofrece una plataforma web donde facilitar los datos del usuario y de la tarjeta de crédito con la que se efectuará el pago.

Para la conexión entre las APIs corporativas y el chat de Facebook se requiere usar una URI(Uniform Resource Identifier) HTTPS (Hypertext Transfer Protocol Secure). Para solucionar este problema uso la herramienta NGrok que crea un túnel desde el servidor Node.js local a un link HTTPS público.

Para el backend se ha utilizado una base de datos NoSql llamada MongoDB. El uso de dicha base de datos proporciona al proyecto una mejora en la escalabilidad, la disponibilidad y en el rendimiento. Además, permite un cambio rápido de esquemas al evolucionar la aplicación.

Capítulo 3

Planificación del proyecto

3.1. Metodología

La metodología usada por la empresa PaynoPain ha sido la *metodología ágil de software SCRUM* y también ha sido la que he usado para la gestión de este proyecto. Tal como observamos en la siguiente fuente [9], esta metodología sirve para gestionar el desarrollo de software, cuyo principal objetivo es maximizar el retorno de la inversión para la empresa. Se basa en construir primero la funcionalidad de mayor valor para el cliente y en los principios de inspección continua, adaptación, auto-gestión e innovación.

La forma de trabajar de la empresa se resume en la herramienta web 'Teamwork', una aplicación web que sirve para monitorizar/añadir tareas a un determinado proyecto con una estimación de tiempo determinada, porcentaje de realización, comentarios, etc. Esta herramienta me ha permitido exportar las tareas del proyecto a un diagrama de Gantt como veremos en las figuras 3.3, 3.4, 3.5 y 3.6.

Según he podido observar en esta empresa se toman muy en serio la preparación previa y el análisis de costes a la hora de empezar un proyecto tanto interno como externo a la empresa. En la preparación previa se empieza planificando las diferentes tareas a realizar entre los diferentes miembros del equipo (interfaz, backend, frontend, etc.). Luego el equipo se esfuerza en completar las tareas en el tiempo estimado previamente establecido entre el jefe de proyecto y el programador. Durante el periodo del proyecto se añaden diversas tareas o se modifican las tareas iniciales, dependiendo de los requerimientos del proyecto o de los clientes.

Las estimaciones temporales de las tareas pueden aumentar debido a complicaciones encontradas durante la programación de la aplicación o refactorizaciones de código necesarias para el futuro mantenimiento tal como me ha sucedido con este proyecto en varias ocasiones.

3.2. Planificación

En la planificación inicial había estimado el proyecto en 6 tareas principales que presento a continuación.

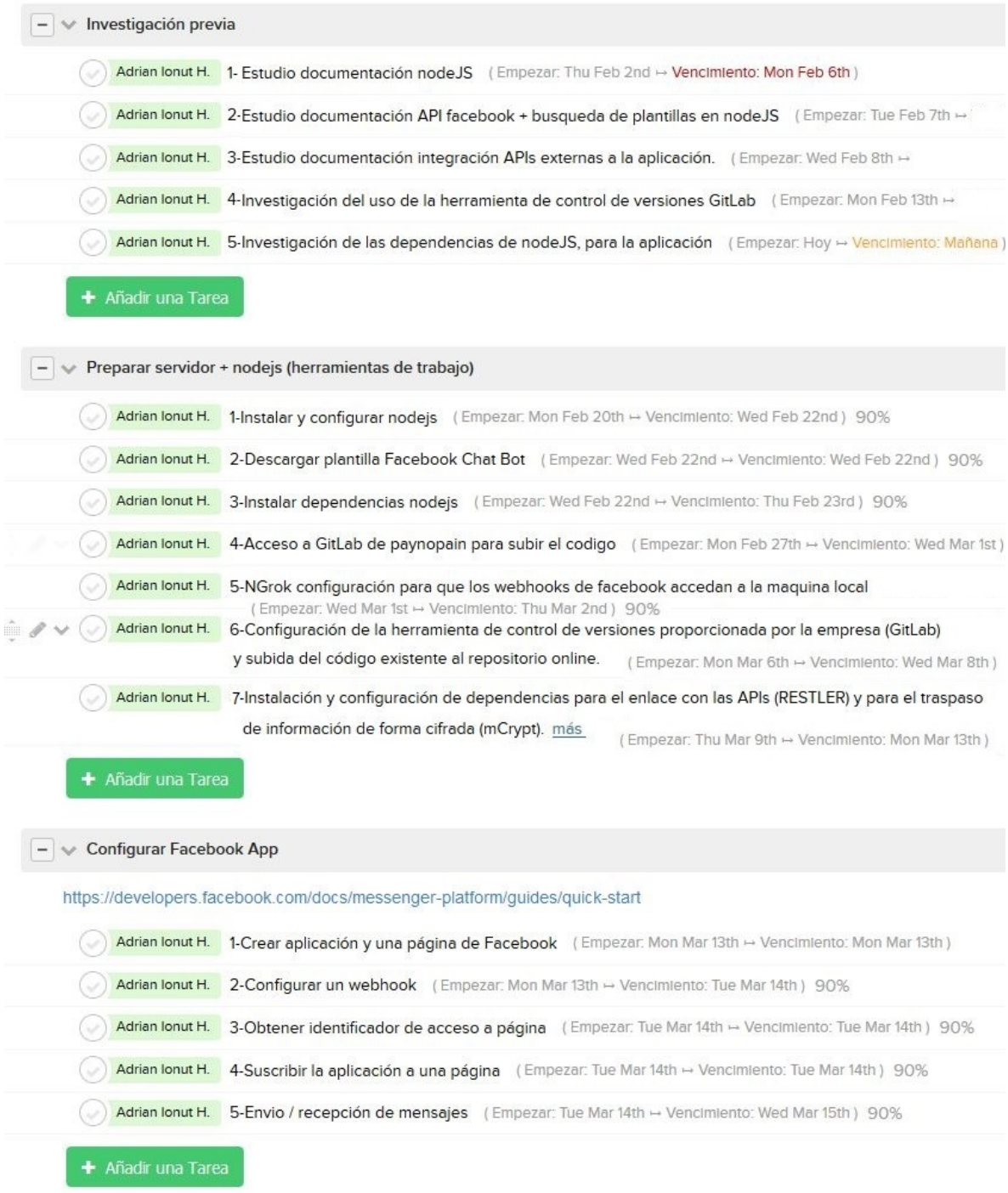


Figura 3.1: Planificación inicial (parte 1)

Figura 3.2: Planificación inicial (parte 2)

En cuanto a la planificación final se dividió el proyecto en 7 tareas principales. Cada etapa está dividida en varias subtareas que ofrecen una explicación más concisa sobre la estructura del proyecto.

- Investigación previa

  1. Estudio de la documentación de Node.js.
  2. Estudio de la documentación del API de Facebook y búsqueda de plantillas para el diseño de chatbots en Node.js.
  3. Estudio de la documentación de las APIs externas y conexión mediante 'request'.
  4. Investigación del uso de la herramienta de control de versiones GitLab
  5. Investigación de las dependencias necesarias para la aplicación.

- Preparación del servidor express en Node.js y las herramientas necesarias para la aplicación.

    1. Instalación y configuración del framework Node.js.
    2. Descarga y configuración de la plantilla Facebook Chat-Bot
    3. Instalación de las dependencias de la plantilla utilizada.
    4. Acceso al portal web de GitLab, ofrecido por la empresa, para monitorizar el control de versiones del código del proyecto.
    5. Configuración e instalación de GIT en el equipo y creación de las claves SSH para poder subir el código a la plataforma de GitLab.
    6. Utilización de la aplicación NGrok para hacer un túnel entre la ruta local del webhook (http://localhost/webhook) y un link https público que utilizará la plataforma de Facebook para enviar o recibir mensajes del chatbot a la aplicación y viceversa
    7. Instalación y configuración de la librería MCrypt para la conexión con la API catálogo.

- Configuración en la página de Facebook para desarrolladores tal como se cita en [3].

    1. En la página de desarrolladores de Facebook creamos una nueva aplicación y una página de Facebook.
    2. Se configura para que la aplicación use 'webhooks' que son interfaces o puntos de llamada que permiten conectar e intercambiar datos entre aplicaciones tal como se cita en [1] .
    3. Se obtiene un identificador o token de acceso a la página que usaremos para conectar con nuestra API.
    4. Se suscribe la aplicación a la página creada anteriormente.
    5. Se prueba el envío / recepción de mensajes básicos implementados en la plantilla.

- Conexión con la API catálogo, uso de MongoDB para almacenar tanto el carrito temporal del cliente como los pedidos definitivos y el formateo de los datos obtenidos desde la API catalogo a las diferentes plantillas de Facebook (lista, carrusel, botones...)

    1. Conversión de las diferentes respuestas del chatbot (antes era solo texto) a mensaje estructurado.
    2. Conexión entre el chatbot y la API catálogo de productos mediante un REQUEST al API con el contenido cifrado en rijndael-256.
    3. Desarrollo e implementación de la estructura hexagonal para separar la aplicación de la parte del core para facilitar la futura migración a diferentes plataformas. Uso de EcmaScript6 básicamente para las respuestas asíncronas usando promesas y Typescript para poder usar objetos en Node.js.
    4. Conexiones entre la API (plantilla) - casos de uso - gateway / APIs externas.
    5. Implementación de funciones para obtener variantes de productos y categorías del API catálogo. (get, list)
    6. Conversión de los productos devueltos del API en formato JSON(JavaScript Object Notation) a plantillas de Facebook.
    7. Se añade una nueva funcionalidad al listar productos del API, buscando primero las variantes de un producto y luego las categorías.

8. Creación de las colecciones CART y ORDER en MongoDB e implementación de las funciones de conexión desde el chatbot.
9. Implementación de paginación para la plantilla 'list'de Facebook.
10. Diseño e implementación del carrito de compra.
    a) Creación de postbacks para llamar y traspasar datos entre diferentes plantillas de Facebook. En la lista de productos tenemos un botón ' Precio Desde' que al pulsarlo nos muestra las variantes de dicho producto en plantilla de botones. Al pulsar el botón con una variante, se llama al método añadir y se guarda el producto en la colección CART de MongoDB.
    b) Se implementa la función 'Listar Carrito' y ' Ver Factura Proforma'.
    c) Se modifican las funciones añadir / listar / eliminar producto del carrito donde cada registro pertenece a un carrito diferente.
    d) Se añade a la paginación el número de productos restantes y una imagen de carro de compra al listar el carrito.
11. Primera refactorización de código. Traspaso de la lógica del gateway a los casos de uso.
12. Segunda refactorización de código. Se añade un parámetro 'quantity' en las funciones de añadir y borrar del carrito. Además, de traslada todas las funciones del gateway en un fichero.
13. Tercera refactorización de código. Se traspasan las funciones de añadir y borrar carrito desde el fichero principal app.js a los casos de uso.
14. Cuarta refactorización de código. Se convierten los JSON devueltos de API catalogo a objetos Typescript (Entities) y esto implica la modificación de todas las funciones de conversión API-plantilla Facebook.

- Conexión e integración al proyecto, del API de pago, Paylands y se termina de implementar la funcionalidad del carrito después del pago.

    1. Investigación de la documentación e integración del API de pago al pulsar el botón de PAGAR en el chatbot que muestra la plataforma de pago para introducir la tarjeta, fecha caducidad, CCV etc.
    2. Se implementa la funcionalidad final del carrito. Al pulsar el botón comprar ejecuta un postback mostrando el botón pagar. Luego al pulsar dicho botón nos redirige a la plataforma de pago. Si el pago se efectúa correctamente redirige a una URL(Uniform Resource Locator) de confirmación y si no a una URL mostrando un error.
    3. Al pagar se hace una copia del carrito en la colección orders y se modifica el estado a ' PENDING'.
    4. Para detectar que se efectúa el pago correcto se pasan cifrados los parámetros devueltos desde la plataforma de pago mediante POST. Luego se comprueban dichos parámetros, y si el pago es correcto cambia el estado a 'CONFIRMED', y si hay algún error cambia el estado a 'DENIED'.

- Inteligencia artificial básica.

    1. Implementación de la función BasicAI(messsage) que recoge el texto del usuario, busca palabras clave en ese texto (artículos, sinónimos de buscar, etc.) y así identifica la localización del nombre del producto que luego se le pasa a la función ListProducts del API catálogo.

- Tests Unitarios (no se ha podido implementar debido a falta de tiempo, pero sí que se ha comprobado el correcto funcionamiento de los diferentes casos de uso que explicaré en el apartado 4.1.1 Casos de uso)
    1. Elaborar tests unitarios para comprobar la validez de los diferentes casos de uso y las llamadas a MongoDB.
    2. Validación en las entidades mediante comprobación de los parámetros enviados a los constructores de las entidades. (tipado, tamaño, etc.)

A continuación, explico una serie de tareas que no había previsto en la planificación inicial a causa de la poca experiencia con la gestión de las APIs en Node.js que tenía al comenzar este proyecto:

1. La adaptación de la estructura hexagonal al proyecto, Ecmascript6 y Typescript que han facilitado la organización, la implementación y el futuro mantenimiento del proyecto.
2. El uso de código asíncrono, mediante 'promesas', para poder gestionar las diferentes peticiones de los usuarios a la vez.
3. Una búsqueda personalizada, al hacer una consulta al chatbot, que busca primero en los productos y si no encuentra ningún producto, luego busca en las categorías del API catálogo.
4. Debido a las limitaciones de las plantillas del API de Facebook tuve que implementar una paginación improvisada, guardando todos los productos devueltos desde la API catálogo y mostrarlos al usuario de 5 en 5, mediante un botón "Ver Más".
5. La implementación de una función que entiende el usuario mediante palabras clave en vez de usar la API externa Wit.ai para entrenar las respuestas del chatbot.
6. Después de efectuar el pago se tienen que gestionar diferentes funcionalidades como:
    - Implementación de la URL_OK y URL_KO.
    - La modificación del estado del pedido dependiendo si el pago se efectúa correctamente o no y la copia del carrito a pedidos.
    - Las cuatro refactorizaciones no estaban previstas porque son debidas a la adaptación de la estructura hexagonal y a varios problemas que me surgieron en la implementación como:
    - El uso del parámetro 'quantity' en la estructura del carrito, que solucionó el problema de los productos repetidos tanto al añadir como al borrar.
    - La conversión de los JSON devueltos por la API catálogo a objetos con tipado gracias a TypeScript y EcmaScript6, ayudó a solucionar varios problemas a la hora de añadir/borrar productos del carrito y a poner restricciones en los parámetros de los productos. Por ejemplo, al añadir un producto al carrito se comprueba que el parámetro stock es mayor que cero.

A continuación en las Figuras 3.3, 3.4, 3.5 y 3.6 podemos ver el diagrama de Gantt que representa la planificación final.

Figura 3.3: Diagrama de Gantt parte 1

Figura 3.4: Diagrama de Gantt parte 2

Figura 3.5: Diagrama de Gantt parte 3

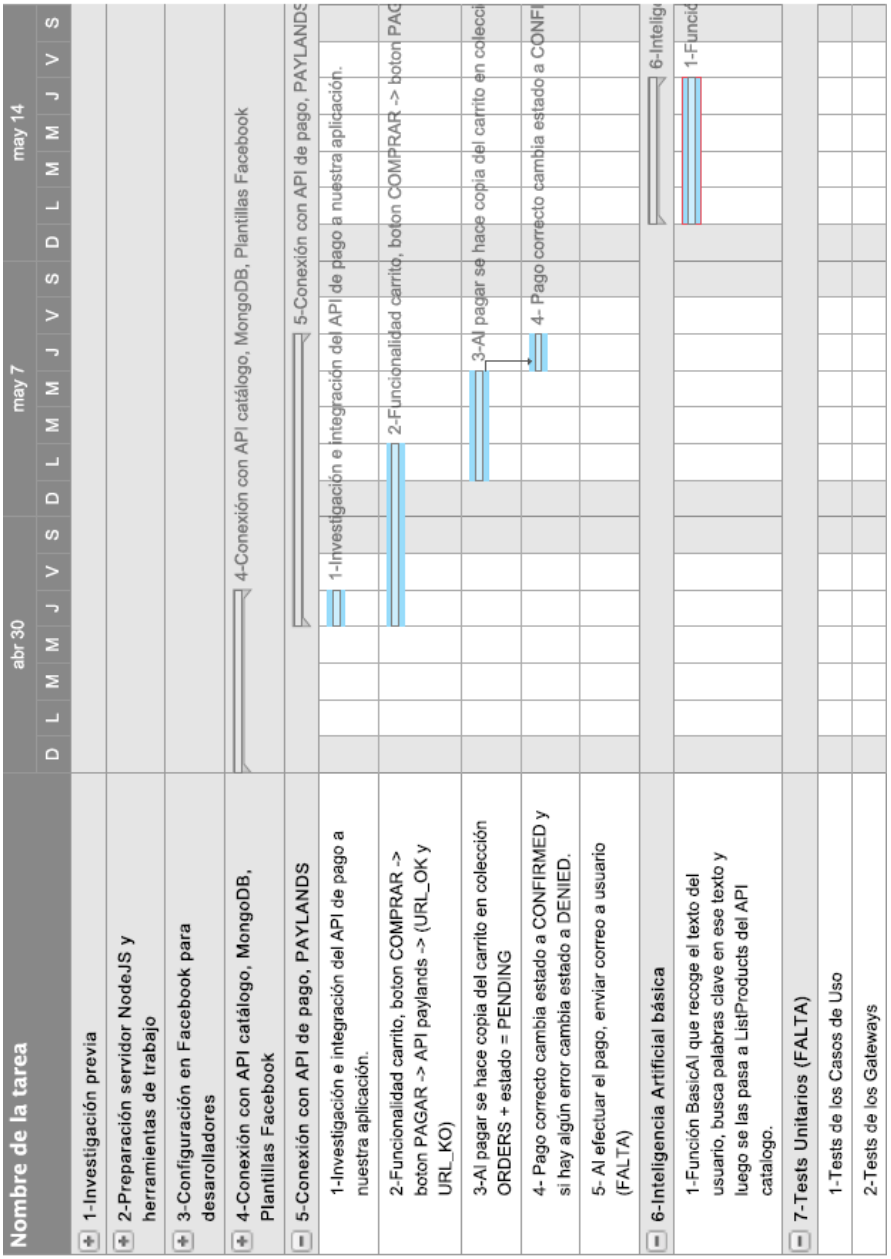


Figura 3.6: Diagrama de Gantt parte 4

3.3. Estimación de recursos y costes del proyecto

Para realizar el proyecto se ha utilizado un PC/portátil de trabajo de gama media durante los 3 meses y medio de la estancia. 600eur x 26 % = 156eur anuales. Pero como solo lo he utilizado durante 3,5 meses seria: (156 x 3.5)/12 = 45,5eur ha costado el uso del PC durante la realización del proyecto.

Una herramienta de control de versiones gratuita, GitLab, ofrecida por la empresa Payno-

pain. Un servidor en la nube como HEROKU donde alojar nuestra API, que ofrezca un dominio seguro con HTTPS para que cumpla los requisitos de Facebook en cuanto al webhook. Utilicé el plan gratuito al no necesitar demasiada potencia para el servidor y no tener un número excesivo de peticiones. Uso gratuito del modo de prueba o sandbox del API de pago PayLands y del API catálogo, proporcionado por la empresa.

El sueldo del programador por las 300h invertidas en desarrollar el proyecto ha sido de 300h x 15eur/h = 4500 eur. La estimación de recursos y costes del proyecto asciende a un total de 4500+156 = 4656 eur.

3.4. Seguimiento del proyecto

El seguimiento del proyecto lo he efectuado mediante una cuenta que me han proporcionado desde la empresa, para acceder a la herramienta web Teamwork donde iba añadiendo las futuras tareas a realizar en un rango de tiempo determinado. Todo esto estaba supervisado por mi tutor, como otro usuario del Teamwork añadido al proyecto. Desde allí podía ver cuando acertaba con mis estimaciones y cuando aparecían fallos inesperados y tenía que alargar el tiempo de alguna tarea. Gracias a esta herramienta, mi supervisor pudo darse cuenta de varios fallos que cometí al implementar la estructura hexagonal que se convirtieron en nuevas tareas no previstas en el proyecto (las 4 refactorizaciones de código que explicaré más adelante en el capítulo 5, apartado 5.1, Detalles de la implementación).

Además, como seguimiento por parte de la empresa se convocó una reunión a principios de abril de 2017, con la directiva de la empresa, el tutor y el supervisor para enseñar una pequeña demo del proyecto. Después de la presentación del proyecto, me comentaron varias mejoras a introducir en el proyecto, y discutimos sobre las diferentes aplicaciones de esta tecnología en el ámbito empresarial.

Capítulo 4

Análisis y diseño del sistema

4.1. Análisis del sistema

A continuación, se presentan los diferentes casos de uso, los requisitos de datos, los requisitos de usuario y las diferentes APIs externas utilizadas en este proyecto.

4.1.1. Casos de uso

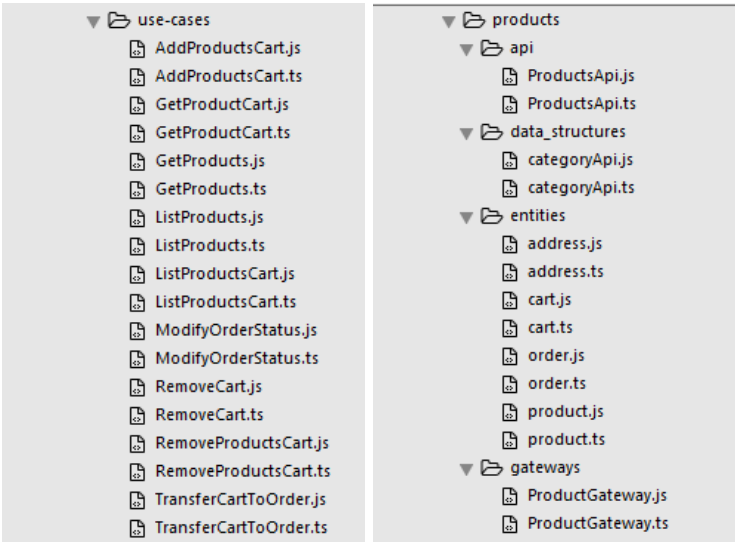
Los casos de uso usados en este proyecto se dividen en 2 partes :

- Los casos de uso referentes al API catálogo.

Casos de uso	Descripción	Validación
GetProducts	Devuelve el producto buscado a partir de un id y un texto cifrado necesario para conectar con la API catálogo	Se compara los datos del producto devuelto con el producto que está en la base de datos.
ListProducts	Devuelve un listado de productos que coincidan con el texto proporcionado en el parámetro 'search' de la llamada a API catálogo	Se comprueba que el texto del parámetro search existe en los nombres de los productos devueltos por la API catálogo.

- Los casos de uso referentes a las colecciones de MongoDB (Cart y Order que se explican más detalladamente en el apartado 4.2 Diseño de la arquitectura del sistema).

Casos de uso	Descripción	Validación
AddProductCart	Sirve para añadir un producto al carrito.	1. Existe carrito y el producto. 2. Existe el carrito pero no el producto. 3. No existe el carrito.
GetProductCart	Devuelve un producto que corresponde con la id del carrito de un usuario.	Se comprueba si el id corresponde con el de un producto que tenemos en la base de datos.
ListProductsCart	Devuelve el carrito perteneciente al usuario conectado al chat.	Se crean diferentes carritos desde diferentes cuentas de usuario de Facebook y se comprueba que lista el carrito correspondiente a cada usuario.
RemoveProductCart	Borra o resta una unidad de un producto determinado del carrito.	1. Existe producto y tiene varias unidades. Se hace la resta del en una unidad en el campo 'quantity'. 2. Existe una unidad del producto. Se elimina el producto.
ModifyOrderStatus	Dependiendo del resultado del pago se cambia el estado del pedido a 'CONFIRMED' o 'DENIED'	1. Se utiliza un número de tarjeta proporcionado por la empresa para efectuar un pago satisfactorio. Se obtienen los parámetros del pago cifrados por POST, se descifran y si el parámetro paid = true se modifica a 'CONFIRMED' 2. Número de tarjeta (Fallo pago). El parámetro paid = false comprobamos que se modifica a 'DENIED'.
TransferCartToOrder	Efectúa una copia del carrito a la colección de pedidos, con el estado a 'PENDING'.	Pulsamos el botón comprar en el carrito devuelto en el chat y luego compruebo que se ha hecho la copia correcta en la colección de pedidos.
RemoveCart	Borra el carrito del usuario conectado al chat.	Al efectuar un pago satisfactorio del carrito, se copia a la colección de pedidos y se elimina en la colección del carrito. Si después del pago satisfactorio intentamos acceder al carrito nos saldrá que está vacío.



(a) Casos de uso (b) Requisitos de datos

Figura 4.1: Estructura de los casos de uso y los requisitos de datos

4.1.2. Requisitos de datos

Seguidamente, se explicarán con más detalle los requisitos de datos que hemos visto en la Figura 4.1b.

Requisitos de datos	Parámetros	Descripción
CategoryApi	id, parentId, applicationId, commerceId, name, imgUrl, Products	Devuelve una estructura con la categoría y un listado de productos.
Address	street, number, zipcode, city, country	Almacena la dirección del envio.
Cart	id, userId, temporal, Products	Guarda los datos del carrito.
Order	id, userId, temporal, status, total, Address, Products	Guarda los datos del pedido.
Product	id, categoryId, commerceId, name, categoryName, price, imgUrl, quantity	Guarda los datos del producto.

4.1.3. Requisitos de usuario y APIs externas

En cuanto al análisis de los requisitos de usuario, puedo decir que se basa en una función, basicAI(message), que se compone de:

- Un vector con lexemas de sinónimos de las palabras: buscar, comprar, mostrar, quiero...
- Un vector con artículos.

Primero se comprueba si en el texto introducido por el usuario se encuentra el lexema de alguna palabra clave del vector, y si no envía todo el texto al parámetro 'search' de la función listProductos del API catálogo. Si encuentra algún lexema que coincida entonces comprueba si la siguiente palabra a la palabra clave es o no un artículo. Si es un artículo entonces envía el texto desde el artículo en adelante y si no desde la palabra clave en adelante.

Ejemplos : Cómo puedo entenderte?

Busco,dame,muestra, quiero un <...> ⇒ busca el producto después del articulo (el, la ,los...).

Ver,busca, dame... < ... > ⇒ busca un producto después de la palabra clave.

< ... > categorías < ... > ⇒ devuelve todas las categorías.

Busco,dame,muestra,ver ... ⇒ escribir solo una palabra clave devuelve todos los productos.

< ... > factura / Total < ... > ⇒ devuelve una factura proforma.

< ... > carro/carrito < ... > ⇒ devuelve el carrito de compra.

Busco,dame,muestra,ver + <nombre categoría> + <nombre producto> ⇒ devuelve producto o categoría (falta implementar, como mejora).

Por otro lado, la función listProducts devuelve primero los productos que coinciden con la búsqueda y si no encuentra ningún producto, intenta buscar si el nombre de alguna categoría coincide con lo que el usuario busca. Dentro del API catálogo encontramos una estructura que cuenta con una categoría padre y dentro las categorías hijo que pueden tener más categorías o un listado de productos. Un ejemplo de esta estructura sería la categoría 'Móviles' que contiene varias categorías 'Nokia 6, Samsung S8, Iphone 5', y en cada categoría hijo hay diferentes productos como por ejemplo, 'Nokia 6 black, Samsung S8 32gb'.

Para efectuar las peticiones al API se tiene que enviar mediante POST un JSON con el applicationId, el iv (la inicialización del vector generado al encriptar los datos) y el encrypt. El encrypt contiene los parámetros requeridos por la función del API (listaProductos) cifrados con el algoritmo Rijndael-256. La API catálogo contiene funciones de listar, añadir y editar, tanto de categorías como de productos y cada una con diversos parámetros internos citados en el documento de gestión del API catálogo facilitado por la empresa [6].

Otro API ofrecido por la empresa ha sido la API de pago, Paylands. Tal como se cita en la documentación [2] ofrecida por la empresa, la API de Paylands está basada en un servicio REST(Representational State Transfer) con diversos servicios que utilizan códigos de respuesta HTTP para indicar los errores. También utiliza JSON para devolver tanto las respuestas como los errores. La autenticación de la API se realiza mediante HTTP Basic Auth donde la API key proporcionado por la empresa se usa como valor en el campo username omitiendo el campo password. Todas las peticiones al API deben realizarse sobre HTTPS, ya que las que se realicen

sobre HTTP o sin autenticar serán denegadas.

En la pasarela de pagos hay dos formas de realizar un pago:

- Redirección: con este pago el sistema pregunta al usuario los datos de la tarjeta a la hora de realizar un pago. Estos datos se piden a través de nuestra carta de pago.
- WebService: este pago se realiza a través de una tarjeta almacenada en el sistema.

En el proyecto usamos el pago a través de 'Redirección'. Este proceso de pago consta de cuatro fases:

- Generar la orden de pago ⇒ El comercio envía las características del pago al endpoint /payment. Paylands devuelve un objeto order junto con un TOKEN que se utilizará para finalizar el pago.
- Completar los datos de pago ⇒ El comercio redirige al usuario a la carta de pago. El usuario introduce los datos de tarjeta. Paylands recibe los datos y procesa la orden.
- Notificación ⇒ El comercio recibe una notificación en la URL especificada en el campo url_post. Se recibe el objeto order actualizado con el resultado de la operación.
- Redirección del usuario ⇒ Paylands redirige al usuario a la URL correspondiente. Si se ha efectuado el pago correctamente se redirige a la URL especificada en el campo url_ok. En cualquier otro caso se redirige a la URL especificada en el campo url_ko.

4.2. Diseño de la arquitectura del sistema

Al principio el diseño de la arquitectura del sistema venía definido por la plantilla inicial, bastante básico donde todas las funciones que llamaban al API de Facebook estaban guardadas en el fichero app.js. A medida que se implementaban más funciones en el proyecto se necesitaba de una estructura donde poder dividir la lógica, los accesos a bases de datos, los accesos a APIs, las entidades... Para ello, desde la empresa se ha propuesto usar la estructura hexagonal, una variante de la arquitectura Modelo-Vista-Controlador, pero más compleja y adaptada al proyecto en Node.js que vimos en la Figura 2.1.

Primero, se crea una función en el fichero app.js. Esta función llama al caso de uso (por ejemplo listar productos) y luego recoge los datos para efectuar la conversión del objeto obtenido a la plantilla de Facebook para enviarla como respuesta al usuario.

Segundo, en los casos de uso se implementa la lógica y se llama a funciones del gateway o de accesos al API que devolverán una promesa con la información deseada.

Tercero, en los gateway y accesos a API, se implementan las funciones de añadir, editar, mostrar, listar... tanto en la base de datos MongoDB como en la API catálogo. Al devolver los

resultados se tienen que convertir de JSON a Objetos Product, Category, Cart, etc., definidos en las diferentes entidades.

Cuarto, las entidades son clases con parámetros, getter y setters escritas en Typescript y compiladas luego a Javascript. Para esta tarea usar Typescript con EcmaScript6 ha facilitado enormemente la implementación, al poder definir clases, getters y setters en un lenguaje similar a Java, que es el lenguaje de programación que más he usado durante esta carrera.

Al principio el proceso de introducción de dicha estructura fue lento y complejo al no encontrar ningún proyecto similar en Node.js que usara dicha estructura. A continuación comentaré el contenido de las diferentes carpetas de este proyecto :

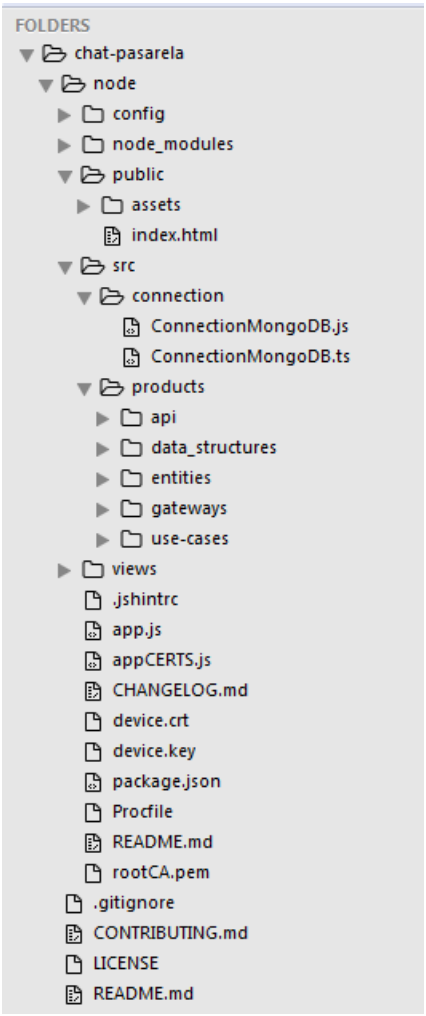


Figura 4.2: Estructura de carpetas

/chat-pasarela/node/...

- 'config' ⇒ Contiene un fichero con las variables globales, API keys, la URL del servidor, ID de aplicación y de comercio del API catálogo...

- 'node_modules' ⇒ Aquí se almacenan las diferentes librerías que necesita la aplicación como Express, Mcrypt, Request, Typescript etc.
- 'public' ⇒ se almacena la página índice que se muestra al acceder a la dirección localhost:5000 al arrancar el servidor. También contiene una carpeta con imágenes que necesitamos para alguna plantilla del chatbot.
- 'src' ⇒ contiene el CORE del proyecto.
- 'src/connection' ⇒ Contiene las funciones de conexión con la base de datos MongoDB.
- 'src/products' ⇒ Contiene la lógica de las funciones, y la implementación de las funciones que manipulan las APIs y la BD.
- 'src/products/api' ⇒ contiene la implementación de las funciones que acceden al API catálogo para devolver los productos / categorías al caso de uso correspondiente convertido de JSON a objeto category o product.
- 'src/products/data_structures' ⇒ define la estructura del objeto category obtenido del API catálogo.
- 'src/products/entities ⇒ contiene las estructuras de los objetos dirección, carrito, pedido, y producto con sus getters y setters.
- 'src/products/gateways'⇒ conjunto de funciones que añaden, eliminan, o modifican en la base de datos MongoDB.
- 'src/products/use-cases' ⇒ es la función global basada principalmente en la lógica (if/else) que usa funciones de manipulación definidas en el gateway (MongoDB) y de listado del API catálogo.
- 'app.js' ⇒ Es el fichero principal donde se encuentra el servidor express, basicAI(), un switch con las diferentes respuestas/plantillas del chatbot, las funciones de conversión a plantilla de Facebook, conexiones con API catálogo y API de pago... Es donde se encuentra la aplicación del chatbot. Este fichero sería el único que deberíamos modificar(plantillas que aceptaría la API de Telegram) si migramos a otra plataforma como por ejemplo Telegram, ya que el CORE sería el mismo.

Para este proyecto se ha usado una base de datos no relacional en MongoDB alojada en la página de mLab, con las dos colecciones usadas en el proyecto.

Una colección para el carrito temporal 'CART', tal como se aprecia en la Figura 4.3, que contiene:

- Un id de carrito único.
- Un objeto 'recipient' que contiene el id de usuario y la fecha y la hora en formato 'timestamp'.
- Un vector 'Product' que guarda los productos añadidos al carrito. De cada producto tenemos un id, el id de la categoría padre, el id del comercio, el nombre de la variante del producto, el nombre del producto, el precio, un enlace a una imagen y las unidades de dicho producto.

```
{
    "_id": {
        "$oid": "5914718ca2e8960ea44b8806"
    },
    "recipient": {
        "id_user": "1231642110265320",
        "temporal": 1494512096145
    },
    "Product": [
        {
            "id": 2685,
            "category_id": 2878,
            "commerce_id": 8,
            "name": "BLACK",
            "category_name": "Nokia Lumia",
            "price": 3,
            "img_url": "https://pre-catalogo.paynopain.com/uploads/58c1829746d7e.jpg",
            "quantity": 2
        }
    ]
}
```


Figura 4.3: Esquema de la colección CART

Otra colección para almacenar el pedido efectuado ‘ORDER‘, como podemos ver en la Figura 4.4 que está formada por:

- Un id único de pedido
- Un objeto ’recipient’ que contiene el id del usuario, la fecha y la hora del pedido en formato ’timestamp’, el estado y el precio total.

    1. Además, dentro de objeto ’recipient’ encontramos otro objeto ’address’ que guarda los datos de envío como: la calle, el número, el código postal, la ciudad y el país.

- Un vector ’Product’ que contiene los productos del pedido con la misma estructura que el vector de productos de la colección CART.

```
{
    "_id": {
        "$oid": "591c91ce40a47d0f04d57e79"
    },
    "recipient": {
        "id_user": "1231642110265320",
        "temporal": 1495044557644,
        "status": "PAYMENT ACCEPTED",
        "total": 363,
        "address": {
            "street": "Av/ Valencia",
            "number": 20,
            "zipcode": 12005,
            "city": "Castellon",
            "country": "Spain"
        }
    },
    "Product": [
        {
            "id": 2684,
            "category_id": 2878,
            "commerce_id": 8,
            "name": "BLUE",
            "category_name": "Nokia Lumia",
            "price": 3,
            "img_url": "https://pre-catalogo.paynopain.com/uploads/58c1829746d7e.jpg",
            "quantity": 1
        }
    ]
}
```


Figura 4.4: Esquema de la colección ORDER

4.3. Diseño de la interfaz

Para este proyecto no se ha tenido que implementar una interfaz ya que la API de Facebook nos ofrece funciones de envío/recepción de plantillas desde el chatbot a la plataforma de Messenger de Facebook. Pero para enlazar con dicha interfaz, previamente se necesita crear una página de Facebook y obtener el token de dicha página. Entonces podremos efectuar un túnel mediante el webhook y suscribirnos a dicha página creada. Como podemos observar en la Figura 4.5, el Messenger de Facebook actúa como canal entre el usuario y el chatbot.

Figura 4.5: Interfaz del Messenger

Capítulo 5

Implementación y pruebas

5.1. Detalles de implementación

La poca preparación con Node.js, promesas, la estructura hexagonal, y de la poca documentación que pude encontrar por Internet, hizo que me basara más en el funcionamiento del chatbot, sin llegar a entender bien del todo el significado de la estructura. Esto provocó varias refactorizaciones en el código bastantes costosas.

Primera refactorización : Como la lógica estaba mezclada tanto en el caso de uso como en el gateway, tuve que hacer el traspaso de la lógica de los gateways y adaptarlo a los casos de uso, con el consiguiente costo de investigación de la devolución de una promesa dentro de otra.

Segunda refactorización: Al definir el primer carrito iba añadiendo productos en la colección y pensaba listarlos por el id del cliente. Pero ahora ha sido modificado a un solo registro que contiene información del carrito y un listado de productos. Como al añadir productos al carrito no tenía en cuenta la cantidad, al borrar un producto repetido me borraba los 2 productos. Finalmente, para solucionarlo se modifica la función de añadir/eliminar con la cantidad y su lógica. Además, se unifican los varios ficheros que tenía en el gateway (add, edit...) en un solo fichero con todas las funciones.

Tercera refactorización: Aun tenía llamadas al API en el fichero principal app.js, y he tenido que crear una nueva jerarquía de caso de uso (addproduct y remove product), la nueva llamada a MongoDB y las conexiones mediante promesas.

Cuarta refactorización: Implica un cambio muy importante y costoso en el proyecto. Hasta ese momento trabajaba con JSON y creando las plantillas a partir de los JSON devueltos por la API. Pero mi tutor me comenta que tengo que utilizar las entidades y crear objetos tipo Cart, Product... Esto implica modificar la salida de los accesos a API catálogo, la salida de los casos de uso, todas las funciones de modificación de JSON a plantilla Facebook, accesos a esos datos con get y set...

El principal detalle de implementación es la estructura hexagonal ya que ayuda a que el

CORE del chatbot sea multiplataforma. Luego el uso de promesas asíncronas (mucho más sencillo que un callback) también fue un reto tanto para mí, como para este proyecto ya que ayuda a que el chatbot responda más rápidamente a gran cantidad de usuarios a la vez.

Además, para solucionar la limitación de la plantilla de listado de Facebook implementé una paginación improvisada devolviendo listados de 5 en 5 de un vector de resultados. Al usar el sistema operativo Windows 7 para el proyecto he tenido varios problemas con las llamadas CURL que luego solucioné con la extensión de chrome, Advanced REST Client.

Luego descubrí que al efectuar demasiados accesos fallidos al webhook de Facebook recibía un bloqueo temporal. Ese fue el momento de subir mi código a Heroku. La subida al servidor de Heroku fue mucho más simple, pero en el despliegue del proyecto en la máquina virtual Amazon AWS de la empresa necesitaba modificar el acceso al servidor express para incluir los certificados de la empresa y obtener una conexión por HTTPS y enlazar mediante webhook con la página del Facebook previamente creada en la cuenta de Paynopain.

5.2. Verificación y validación

En la propuesta técnica ya tenía planeado que la verificación / validación se llevaría con técnicas TDD(Test-driven development) y test unitarios. Pero a causa de las refactorizaciones de código no tuve el tiempo suficiente de hacer la implementación. Desde la empresa he podido ver bastantes ejemplos de TDD en Java. Según me han explicado hay diferentes casos:

- Test sobre la lógica de los casos de uso, aportando distintos valores para comprobar los diferentes caminos que puede tomar una función de caso de uso.
- Test sobre los gateways o funciones de añadir, editar, eliminar en la base de datos de Mongo. Se inserta primero y se comprueba que se ha insertado correctamente, luego se edita y comprueba que el valor editado es el mismo pasado a la función. En estos test se tiene que ir con cuidado de no trabajar sobre la base de datos del proyecto si no sobre una copia.
- Test sobre las entidades o los parámetros de las clases tipo carrito, producto... Lo que se lleva a cabo son comprobaciones en el tipado de los parámetros que se pasan al constructor de la entidad (int, string, etc.) y en poner restricciones de tamaño como por ejemplo: número teléfono 9 dígitos, nombre no más de 200 caracteres...

Hay varias herramientas que se podrían usar para dichos test en Node.js, como Mocha, Jasmine, NodeUnit, Vows...

En cuanto a la verificación de la implementación del chatbot, fue realizada mediante preguntas al chatbot tal como se indica anteriormente en el apartado 4.1 Análisis del sistema, buscando un producto, comprobando que el listado es correcto, añadiendo a carrito, efectuando el pago de un pedido. Además, al añadir un producto al carrito también compruebo que en la colección CART se inserta correctamente, y al hacer el pago, que se copia correctamente en la colección ORDER.

Capítulo 6

Conclusiones

Este proyecto ha sido un reto para mí, al no tener demasiada experiencia en Node.js ni sobre la estructura hexagonal. Pero investigando y tomando en cuenta los consejos de mi supervisor y de los compañeros de la empresa conseguí cumplir la mayoría de los objetivos iniciales como por ejemplo:

- Comprender el funcionamiento del webhook, y conseguir un túnel desde el servidor local a una dirección HTTPS exigida por Facebook.
- Adaptar y usar la estructura hexagonal en un proyecto Node.js.
- Utilizar las promesas para agilizar las respuestas del chatbot mediante código asíncrono.
- Implementar la función para entender las peticiones del usuario.
- Conectar el proyecto con las diferentes APIs (Facebook, catalogo y pago) y enviar y recibir datos de dichas APIs.
- Implementar un carrito de compra.
- A pesar de las limitaciones del API de Facebook conseguir implementar una paginación improvisada en la plantilla list.
- Conectar con la plataforma de pago, recoger y descifrar la respuesta después de la transacción (pagado o no).

Por otro lado, también hay varios objetivos que no conseguí cumplir como:

- El envío de un correo con la factura al usuario, después de realizar un pedido.
- Utilizar técnicas TDD para comprobar el correcto funcionamiento de los casos de uso, entidades, accesos a base de datos etc.

En cuanto a la experiencia en la empresa Paynopain ha sido muy buena ya que desde el primer día me acogieron como un trabajador más. La gente es sociable, creativa y trabaja

en equipo resolviendo los problemas que aparecen en los proyectos. Gracias a esta estancia he podido experimentar y aprender más sobre el funcionamiento de una empresa basada en actividades de programación informática y comprender el gran potencial que puede ofrecer un chatbot multiplataforma no solo a las compras online sino también a las consultas a un API o base de datos mediante una pregunta en un chat de un servicio de mensajera.

Bibliografía

[1] Victor Campuzano. Webhooks: Qué son y por qué deberías aprender sobre ellos. https://vicampuzano.com/webhooks-deberias-aprender. [Consulta: 10 de Julio de 2017].

[2] Desarolladores de PaynoPain. Documentación api de pago paylands. http://docs.paylands.apiary.io/#introduction/proceso-de-pago/pago-a-traves-de-redireccion. [Consulta: 20 de Junio de 2017].

[3] Facebook Documentation. Configuración página facebook desarrolladores. https://developers.facebook.com/docs/messenger-platform/guides/setup. [Consulta: 15 de Junio de 2017].

[4] Jorge González (jorgonor). Promesas en nodejs. https://jorgonor.com/es/blog/javascript/2016/06/10/programacion-asincrona-promesas.html. [Consulta: 18 de Junio de 2017].

[5] Chris Maas. Diagrama estructura hexagonal. http://www.ntaso.com/towards-a-clean-architecture-for-ionic-2-apps. [Consulta: 12 de Junio de 2017].

[6] Paynopain. Gestión de productos y categorías (api catálogo). [Consulta: 14 de Junio de 2017].

[7] Juan Quijano. Typescript y ecmascript6. https://www.genbetadev.com/javascript/hello-world-en-typescript-el-lenguaje-en-el-que-se-construira-angular-2. [Consulta: 08 de Junio de 2017].

[8] Parambir Singh. Plantilla facebook node.js. https://github.com/fbsamples/messenger-platform-samples/tree/master/node. [Consulta: 05 de Junio de 2017].

[9] Softeng. Metodología scrum. https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html. [Consulta: 17 de Julio de 2017].

[10] Thierry Templier. Implementing a messenger bot. https://www.hitchhq.com/facebook-graph-api/docs/messenger-bot. [Consulta: 24 de Junio de 2017].

Anexo A

Capturas de mensajes del chatbot

En la Figura A.1 se muestra un mensaje de saludo como respuesta por parte del chatbot y varios mensajes estructurados con la plantilla 'lista' al buscar el producto 'Nokia' y la categoría 'móviles'.

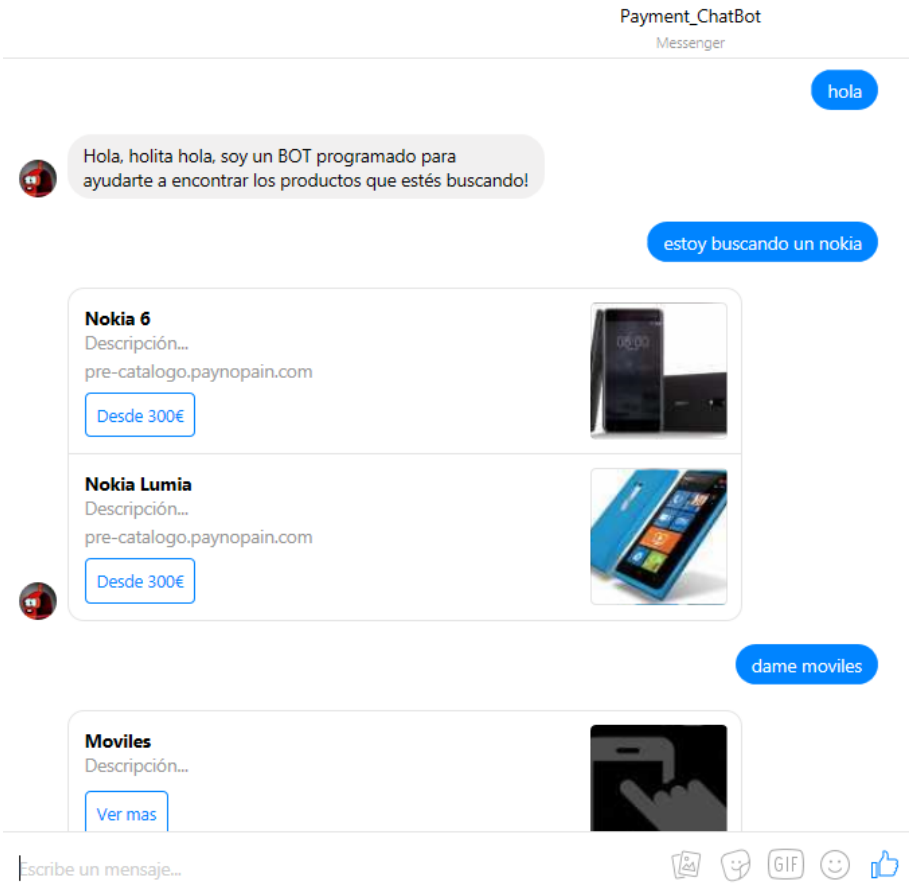


Figura A.1: Búsqueda de productos y categorías

Como vemos en la Figura A.2, para añadir un producto al carrito se pulsa primero en el botón del precio. Después aparecerán diferentes botones con las variantes de dicho producto. Al elegir la variante deseada se añade el producto al carrito y muestra un mensaje de confirmación.

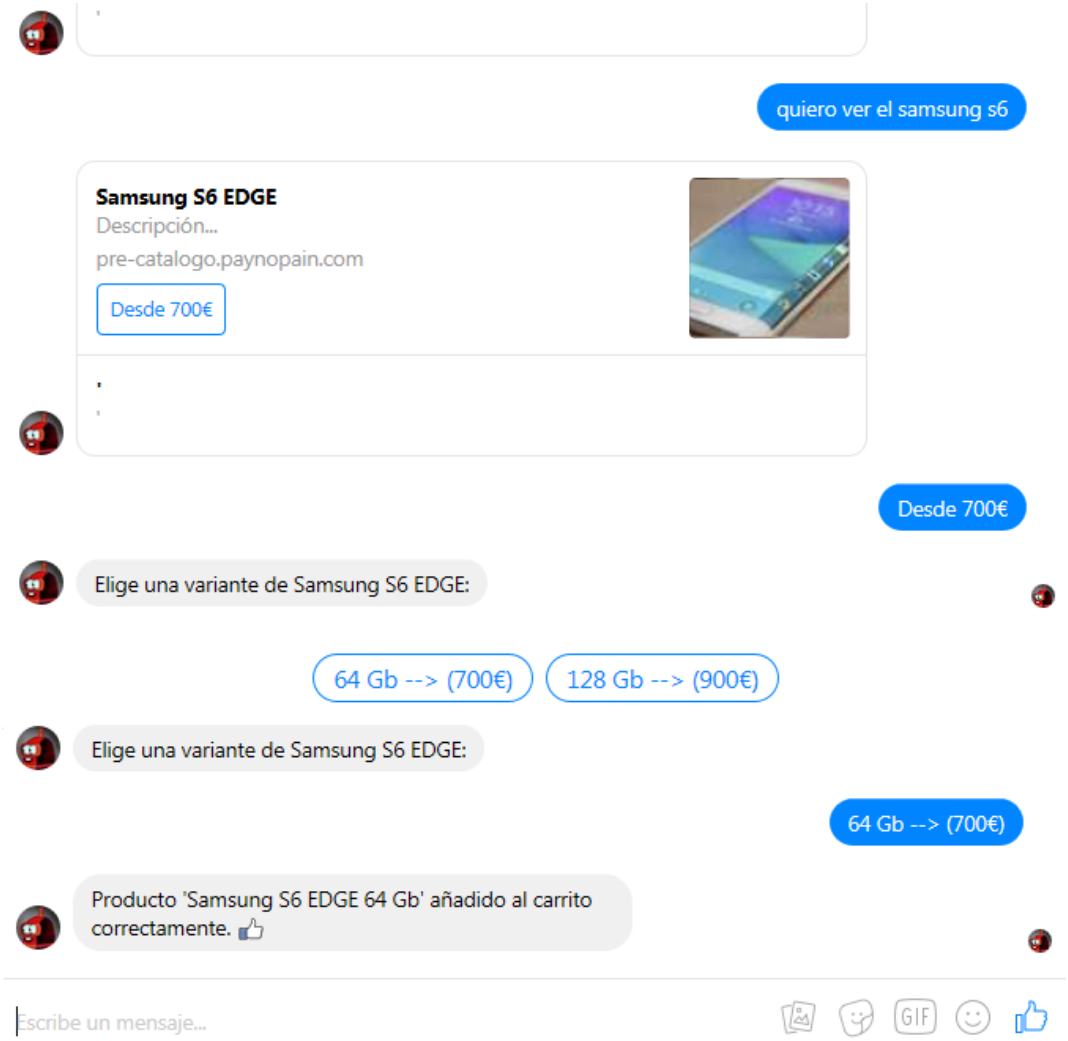


Figura A.2: Añadir producto a carrito

Cuando se encuentra la palabra 'categorías' en la frase introducida por el usuario, nos muestra un listado con todas las categorías del API catálogo como podemos ver en la Figura A.3. Además, cada categoría tiene la opción 'Ver más' que al seleccionarla muestra todos los productos de dicha categoría.
Pero cuando la frase contiene la palabra 'carrito' se muestra un listado con los productos añadidos al carrito, con la opción de eliminarlos uno a uno y la opción de efectuar la compra tal como se muestra en la Figura A.4.

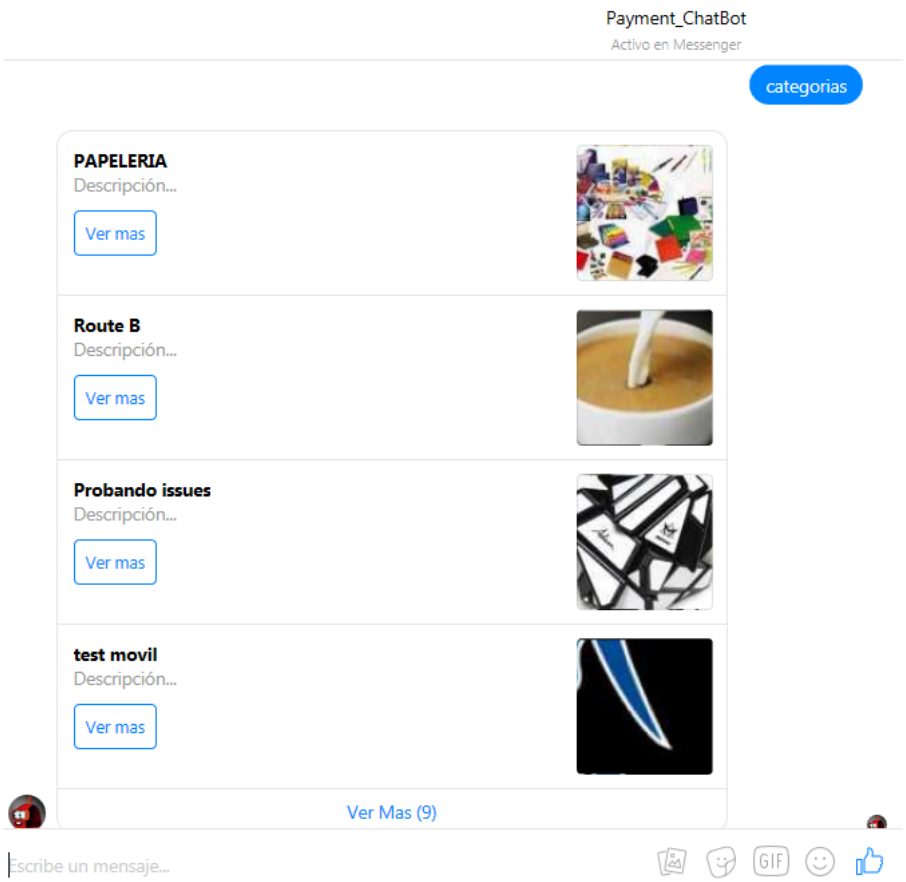


Figura A.3: Listar categorías

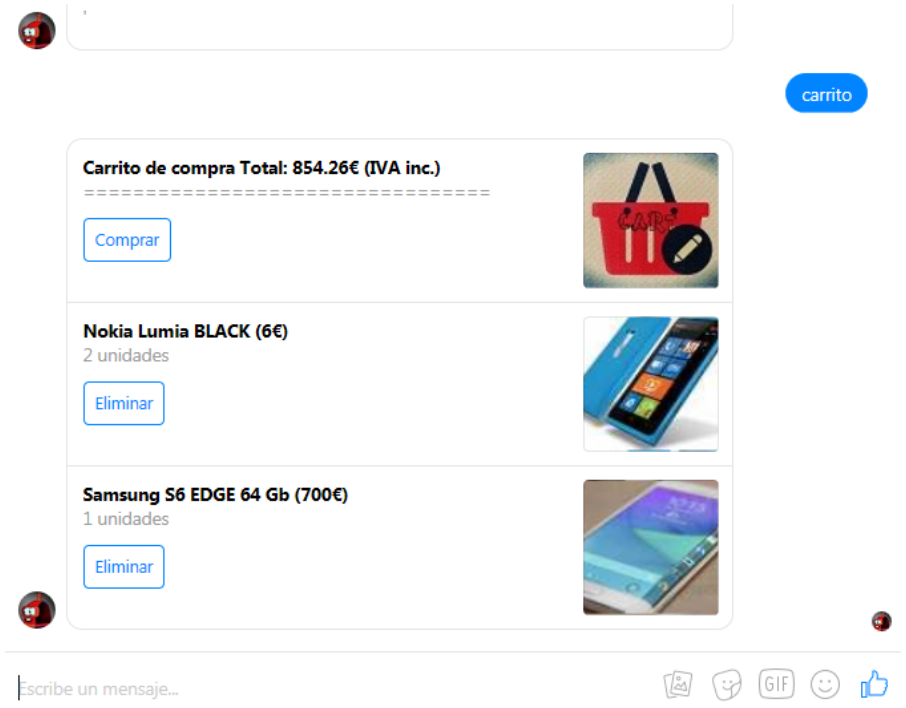


Figura A.4: Carrito de compra

Para mostrar un resumen mas detallado del carrito se introduce la palabra 'factura' en la frase del usuario tal como se aprecia en las Figuras A.5 y A.6.

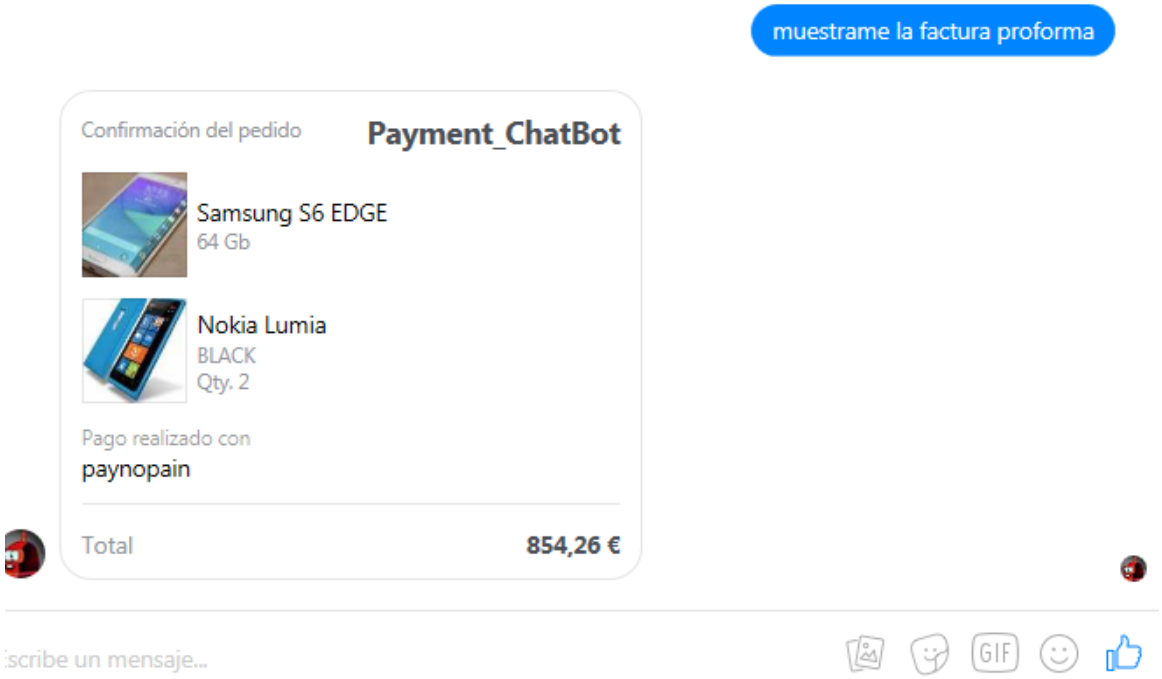


Figura A.5: Factura proforma

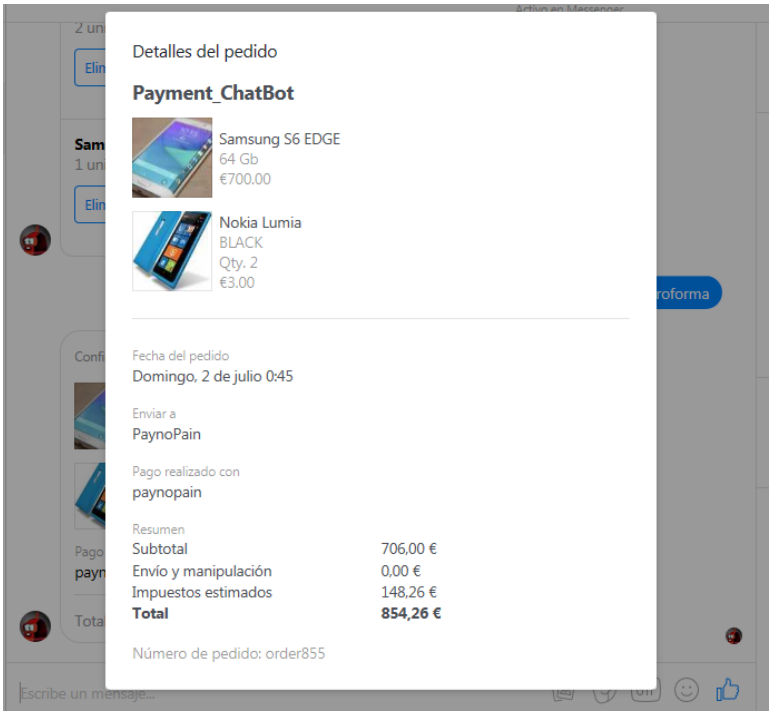


Figura A.6: Detalles de la factura proforma

Al efectuar la compra desde el carrito, nos redirige a la pagina web de la plataforma de pago que podemos ver en la Figura A.7. Dependiendo si el pago se realiza de forma satisfactoria o con errores nos mostrará los mensajes que podemos ver en la Figura A.8.

Figura A.7: Interfaz de pago

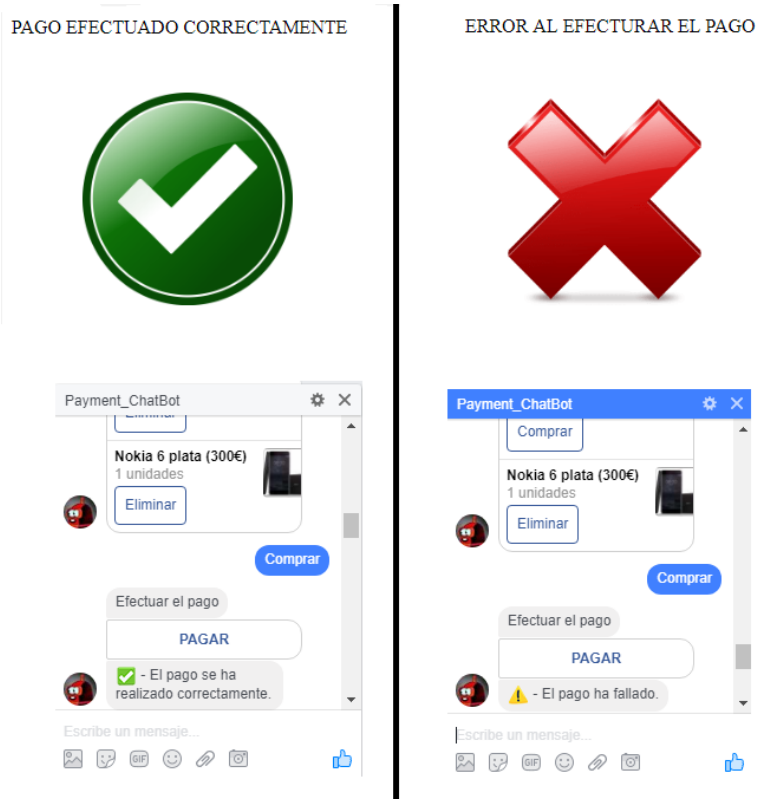


Figura A.8: Mensajes de confirmación