USO DE SOLUCIONES CLOUD PARA LA AUTOMATIZACIÓN DE PROCESOS

Alejandro Sánchez Ortega

Resumen— El proyecto se centra en el desarrollo de una plataforma web que, junto con una API, es capaz de registrar todo tipo de información de las operaciones generadas por una empresa que trabaja en la cadena de suministros, como la gestión de envíos, llegadas y facturas de pedidos. Además, se utilizan servicios cloud computing para automatizar procesos de reportes y generar informes personalizados que proporcionen, al usuario, una disposición muy visual y organizada de los datos registrados. Mediante una serie de flujos de trabajo, generados con el recurso de *Microsoft Azure* llamado *Logic App*, la plataforma ofrece una serie de suscripciones a los informes generados.

Palabras clave—Cloud Computing, Logic App, App Service, Microsoft Azure, .NET Core, DDD, C#, MVC, HTML, CSS, programación web

Abstract— The project focuses on the development of a web platform that, together with an API, can record all kinds of information on the operations generated by a company that works in the supply chain, such as the management of dispatches, arrivals and invoices of orders. In addition, cloud computing services are used to automate reporting processes and generate personalized reports that provide the user with a highly visual and organized arrangement of the recorded data. Through a series of workflows, generated with the Microsoft Azure resource called Logic App, the platform offers a series of subscriptions to the generated reports.

Index Terms— Cloud Computing, Logic App, App Service, Microsoft Azure, .NET Core, DDD, C#, MVC, HTML, CSS, web development

1 Introducción

La gestión de registros es una de las funciones imprescindibles en cualquier empresa u organización. Da la posibilidad de tener un control exhaustivo de cualquier movimiento y acción dentro de la entidad. Este proceso aborda tres fases del ciclo de vida de los registros: la creación o recepción; el mantenimiento, el almacenamiento, la recuperación o el uso general; y la disposición.

Hoy en día, una empresa puede llegar a almacenar miles, e incluso millones, de registros y, más allá de las preocupaciones legales, debe haber un propósito adicional para su almacenamiento.

Los tipos de registros a considerar se pueden suponer una vez se entiende el flujo de una cadena de suministros. Cuando un cliente hace un pedido, por ejemplo. Primero debe elegir uno o varios productos los cuales tienen un identificador único, con un precio concreto, etc. El pago se realiza para recibirlo en una dirección, ciudad, país concreto, con un método concreto, etc.

Lo que se sugiere con este ejemplo es que, a partir de la sencilla acción de hacer un pedido, se deben generar varios informes que se necesitan registrar para su correcta gestión: datos de factura, datos de pago, datos de envío, datos recepciones, etc.

Partiendo de este punto, el proyecto está relacionado con dar una disposición a cualquier tipo de dato de los registros almacenados por una empresa que trabaja en la cadena de suministros.

Por otro lado, para tener un registro de estos datos, por parte de los usuarios, debe haber una herramienta que ofrezca la posibilidad de registrarlos. Son varios los métodos para ejercer esta función, como puede ser mediante registros físicos o, como es más común, con dispositivos móviles o plataformas web. Para estos dos últimos, el sistema de almacenaje suele ser común, es decir, en bases de datos compartidos por ambas opciones.

Hay que señalar que estos sistemas, capaces de registrar cuantiosas cantidades de datos, deben tener una monitorización y un control del flujo de las acciones. Esto es lo que permitirá dar fiabilidad y consistencia a los datos registrados por los usuarios y evitar cualquier tipo de error que provoque su pérdida, duplicación, etc. Según el impacto de la entidad, un error inesperado puede significar millones de datos erróneos o perdidos en cuestión de minutos.

- *E-mail de contacte: Alejandro.SanchezO@e-campus.uab.cat*
- *Menció realitzada: Enginyeria del Software*
- *Treball tutoritzat per: Jordi Serra Ruíz (Ciències de la Computació)*
- *Curs 2019/20*

Es precisamente por lo comentado antes que en este proyecto se ha optado por crear una plataforma web de gestión. La accesibilidad global, simultaneidad de usuarios y la disponibilidad de la información en un formato más visual, la han colocado por delante de la opción del dispositivo móvil. Por otro lado, este último es el más eficiente a la hora de registrar los datos *in situ*, por ejemplo, a la hora de entregar un pedido. En una situación real, ambas opciones se complementan.

A la hora de hacer el desarrollo de la plataforma, se ha optado por .NET principalmente por su soporte para múltiples lenguajes, el fácil desarrollo basado en componentes y la compilación *just-in-time*, que permite generar el código máquina propio de la plataforma, aumentado el rendimiento de la aplicación al ser específico para cada plataforma. Este último aspecto ha sido muy importante, ya que se ha necesitado una plataforma eficiente y óptima, capaz de acceder a muchos datos lo más rápido posible.

Para que sea posible la automatización de los procesos de la plataforma, se precisa de una API personalizada que permite abstraer funciones y ser ejecutadas desde llamadas externas. Esta biblioteca de subrutinas debía poder realizar las funciones necesarias para que la plataforma sea eficiente y, a grandes rasgos, útil.

Con tal de ofrecer una disposición de los registros al usuario, se han barajado varias opciones en las que se involucraban servicios de *cloud computing* que permitiesen automatizar procesos y crear flujos de trabajo.

En este desarrollo se ha optado por *Microsoft Azure [1]*, un conjunto de servicios de informática en la nube que se utilizan en una extensa cantidad de empresas y entidades. Ofrece mucha flexibilidad y globalización. Entre sus recursos disponibles se encuentra la posibilidad de crear máquinas virtuales, sistemas de almacenamiento, monitorización, migraciones, etc. Además es compatible con soluciones *.NET Core*.

Se han creado varios flujos de trabajo que permitan dar una disposición personalizada de los registros a los usuarios. Para ello se ha utilizado un servicio llamado *Logic Apps [2]*. Éste permite automatizar y orquestar flujos de acciones y procesos de una manera visual, organizada y simplificada. Entre sus principales aplicaciones está la de llamar a APIs y realizar procesos a partir de un desencadenador.

Para entender de manera más clara cómo funcionan las Logic Apps [3], se puede observar el ejemplo de la Fig 1. Como se aprecia, hay dos cajas las cuales representan una acción cada una de ellas.

La primera de ellas es el desencadenador de la aplicación lógica, es decir, el "¿qué tiene que pasar para que se active?". Todas las Logic Apps deben tener un elemento desencadenante *trigger* para empezar a ejecutarse.

La segunda, es la acción que encadena el desencadenador.

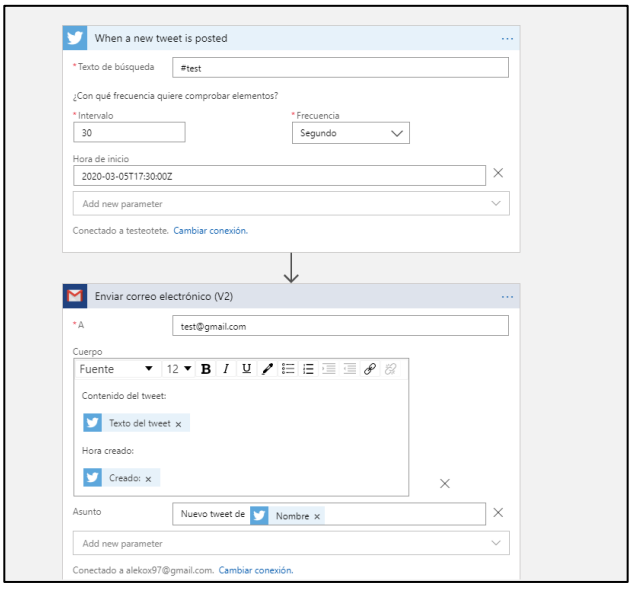

Fig. 1. Ejemplo Logic App

En el ejemplo, se puede apreciar que, en primer lugar, si se publica un tweet que contenga la etiqueta #test , se activará la *Logic App*. Seguidamente, pasará a la segunda caja, que recuperará datos del *tweet* y se enviará un correo electrónico personalizado.
Como se puede ver, es un flujo de trabajo muy básico, que simplifica varias acciones y que ha supuesto apenas un par de minutos de dedicación.

El potencial que ofrece este servicio se pondrá a prueba más adelante, en el apartado de resultados.

2 ESTADO DEL ARTE

A la hora de desarrollar una plataforma web de gestión, muchos son los ejemplos en que fijarse. La gran mayoría de las empresas del sector de la logística disponen de sus plataformas propias o estandarizadas, que comparten características y estructuras similares. Por lo que si un diseño ya funciona con eficiencia no hace falta variarlo en exceso.

Al final, este tipo de plataformas web no buscan obtener usuarios nuevos ni posicionarse en Internet, es decir, no necesitan ser atractivas o llamativas, sino que deben ser funcionales. Es por eso por lo que en este proyecto se ha optado por adoptar una visión simple, en cuanto a estructura, y funcional.

Por otro lado, cuando se habla de *cloud computing*, en la actualidad existen una gran cantidad de opciones que escoger. Tanto *Microsoft Azure [1]*, como *Amazon Web Services (AWS)*, como *IBM* ofrecen servicios en tres capas (IaaS, PaaS y SaaS).

La decisión de optar por Microsoft Azure por delante de AWS, se ha tomado principalmente por comodidad y compatibilidad de sus recursos con *Microsoft Studio* y *.NET Core*.

3 Objetivos

El objetivo principal del proyecto ha sido, como se ha mencionado, dar una disposición funcional a los usuarios de los registros que se tienen almacenados [6]. Para ello se ha dividido los objetivos del desarrollo en dos partes diferenciadas: el desarrollo del front-end y back-end, y la implementación de las Logic Apps.

En primer lugar , se ha desarrollado la plataforma web capaz de registrar toda la información. Las tareas que se han realizado están relacionadas con la implementación de las funcionalidades necesarias para la correcta disponibilidad de los datos y registros:

- **Operaciones**. Funcionalidades encargadas de registrar todo el flujo de acción de la aplicación desde el registro de un pedido, pasando por el registro de envío y llegando al registro de las recepciones:
  - Factura: se registran facturas de pedidos.
  - Envíos: se registra cuando un pedido se envía de una fábrica a su destinación.
  - Recepciones: se registra un pedido que ha llegado a su destinación.
- **Gestión.** Funcionalidades encargadas de registrar fábricas y datos de la empresa que posee fábricas:
  - EconomicOperator: se registran datos de empresas.
  - Facilities: se registran datos de fábricas.
- **Informes**. Sistema de reportes con los resultados de las operaciones efectuadas:
  - Facturas: muestra un resumen filtrado de los registros de facturas.
  - Envíos: muestra un resumen filtrado de los registros de envíos.
  - Recepciones: muestra un resumen filtrado de los registros de recepciones.
  - Seriales: muestra un resumen filtrado de los seriales asignados a cada producto y los registros relacionados.

Una vez desarrollado el front-end, la siguiente tarea ha sido crear la base de datos para poder registrar todas las entidades. Seguidamente, el desarrollo del back-end ha englobado una serie de tareas encargadas de dar puntos de accesos al front-end. Es decir, cualquier funcionalidad que ha necesitado acceder a la base de datos.

La segunda parte de los objetivos ha englobado la parte de implementación de las Logic App.

Para una correcta disposición de los datos al usuario, la plataforma ha de llamar a varios flujos de trabajo que automaticen el proceso de creación de ficheros con los informes de forma correcta. Se ha valorado las varias aplicaciones que ofrecen las Logic Apps y finalmente se ha optado por ofrecer a los usuarios acceso a documentos en formato *docx*, que recojan todos los resultados.

Como últimas tareas a considerar han sido las de desplegar la aplicación en un servidor, dar acceso global e implementar tipos de monitorización de errores.

4 Metodología

A la hora de decidir una metodología de trabajo, se ha valorado en primera instancia una metodología en cascada [4][5] pero que ha ido poco a poco modificándose, debido a la corrección de ciertos aspectos durante el desarrollo. En primera instancia, sin embargo, el desarrollo ha sido de manera secuencial, por fases, comenzando por un análisis, diseño y terminando con la puesta en producción y pruebas.

Las ventajas que supone este tipo de metodología son: (1) la capacidad de medir de forma más fácil el progreso del proyecto, (2) la planificación es más sencilla, (3) solo hay una persona implicada en el desarrollo y (4) se deben desarrollar varios componentes de software de manera paralela.

Las posibles desventajas de usar esta metodología están relacionadas con las primeras fases, de captación de requisitos y diseño. Es decir, hay que asegurar la correcta realización de las primeras fases para evitar arrastrar el error durante todo el desarrollo.

En cierto punto del proyecto se ha decidido, por necesidad, romper con esta metodología de trabajo, ya que muchas funcionalidades del back-end y front-end precisaban una modificación en cuanto a sus requisitos y diseño. Se ha tenido presente también el uso de metodologías Ágiles pero no se han adoptado, ya que están enfocadas en proyectos en equipo.

Finalmente, la metodología ha consistido en el desarrollo paralelo de front, back y logic apps, puesto que el desarrollo de según qué componente ha precisado modificaciones de otro y de otras fases.

Se han seguido las etapas clásicas del desarrollo de software: captación de requisitos, diseño, implementación y test.

La dinámica de trabajo de implementación que se ha seguido se divide en dos partes: el desarrollo de la plataforma y la implementación de los servicios de cloud computing de Azure.

5 Planificación

5.1 Planificación de las tareas

La fecha de finalización del proyecto ha sido marcada para finales de junio. Es decir, para entonces debe haber una solución funcional y testeada del conjunto del proyecto.

Para cumplir con los términos marcados, el proyecto se ha dividido, como se ha comentado, en dos partes:

- Fase de desarrollo de Plataforma Web y API
- Fase de automatización de procesos y monitorización con Logic Apps.

La primera de ellas ha sido marcada con un deadline el 25 de mayo. La segunda parte ha sido establecida para estar finalizada tres semanas después, el 15 de junio. De esta manera se han dispuesto de dos semanas, aproximadamente, para hacer las pruebas finales de ejecución. Los últimos días antes de la entrega final han servido para finalizar el informe final, realizar el artículo y preparar la exposición.

5.2 Fase Inicial

La primera fase del proyecto es la responsable de dar una idea concisa del trabajo a realizar, valorar las capacidades y límites del proyecto y dar una visión concreta del rumbo a seguir. Fecha final: 8 de marzo de 2020

5.3 Captación de requisitos

Para captar los requisitos, se ha realizado una investigación para concretar y responder varias preguntas como por ejemplo: qué necesita la plataforma web, cómo son las plataformas web similares, qué tipo de API se necesitará, qué tipo y cuántos recursos habrá que contratar en Azure, qué modelo de programación se seguirá, etc. Fecha final: 13-03-2020

5.4 Preparación del entorno y diseño

Una vez respondidas casi todas las preguntas anteriores, o todas las imprescindibles, se ha decidido cómo será la plataforma web y qué tipo de diseño tendrá. Fecha final: 10-04-2020

5.5 Desarrollo y Test

Se desarrollará mediante TDD, es decir, en todo momento se deberá probar que lo que se desarrolla cumplirá con las especificaciones expuestas. Para ello, en primer lugar habrá que crear el entorno web, y, paralelamente, crear la API responsable de realizar las acciones. Por último, se abordará el tema principal del proyecto, las Logic Apps. Fecha final: 30-05-2020

5.6 Test y rendimiento

Para finalizar el desarrollo, se han realizado las pruebas de rendimiento finales para saber si la plataforma web es eficiente, en tiempo y precio.

Puesto que la plataforma es una simulación de un caso real en el que se accede a miles de datos, hay que tener especial cuidado con las consultas a las bases de datos y controlar los tiempos. Fecha final: 26-06-2020

5.7 Pizarra

Para tener una organización concreta y saber qué tareas se deben completar, cuáles se han dado por cerradas, etc. se ha trabajado con una pizarra de Trello[12] que representa el backlog del proyecto con todas sus tareas.

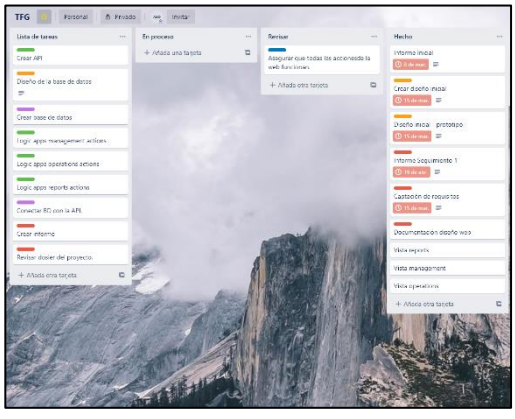

Fig. 2. Pizarra Trello

Evaluación de riesgos

Un aspecto esencial de la planificación de un proyecto es la evaluación y análisis de los posibles riesgos que puedan suponer un compromiso para el desarrollo. En este caso, los posibles riesgos a tener en cuenta han sido relacionados con el desconocimiento y falta de experiencia:

- Falta de experiencia en cloud computing: una parte del proyecto ha comportado la implementación de servicios que ofrece Microsoft Azure.
- Programación web: se disponían de conocimientos básicos en programación web, impartidos en el grado.
- Limitación de Logic Apps: en un principio, uno de los factores de riesgo más importantes para tener en cuenta ha sido las limitaciones que ofrecen las Logic Apps. Es por ello, que desde un inicio se ha investigado qué pueden ofrecer y qué no.

6 Desarrollo

6.1 Análisis de requisitos

El primer paso realizado ha sido la captación de requisitos. Como se ha comentado en los objetivos, ha sido muy importante el ¿qué hay que hacer?
Para ello, en primer lugar se ha estudiado la situación principal: ¿qué registros se precisan almacenar? y ¿qué tipo de funcionalidades se necesitan?

Como se ha explicado en el apartado de metodología, el proyecto ha sufrido muchos cambios de fases iniciales a lo largo del desarrollo y la captación de requisitos ha sido la principal.

La especificación de los requisitos se ha seguido según el estándar IEEE 830-1998, es decir, han de ser completos, consistentes, inequívocos, modificables, trazables… Se han dividido los requisitos en dos partes: funcionales y de datos.

En la Fig.3 y Fig.4 se aprecian ejemplos del formato seguido para la creación de éstos:

Código	RQD - 01
Nombre	Facturas
Versión	1.0
Autor	Alex
Tipo	Datos
Descripción	La información que se dispone de las facturas es la siguiente: • Id • EOID • Tipo • Numero • Fecha de factura • Cantidad • Moneda • Id del comprador • Estado • Fecha de creación
Prioridad	Alta

Fig. 3. Requisito de datos: Facturas

Código	RQF - 01
Nombre	Registrar factura
Versión	1.0
Autor	Alex
Tipo	Funcional
Descripción	El usuario debe poder registrar una nueva factura con sus datos pertinentes por parte del usuario.
Prioridad	Alta

Fig. 4. Requisito funcional: Registrar factura

6.2 Diseño

El paso que seguir después de la captación de requisitos ha sido el diseño. En todos los proyectos es un paso crucial, ya que antes del desarrollo e implementación de la web se debe tener un mapa global de cómo va a estar organizada ésta, cómo se van a disponer los datos y cómo se va a poder navegar por ella, además de la apariencia. También ofrece la posibilidad de hacer cambios en una etapa temprana del proyecto y llegar a una propuesta que satisfaga a los futuros usuarios. [7]

"Los prototipos nos permiten explorar nuestras ideas antes de invertir tiempo y dinero en el desarrollo. Lejos de ser una pérdida de tiempo, trabajar con prototipos nos permiten crear múltiples soluciones para poder fallar rápido y barato ".

Para este proceso, se disponen de varias herramientas y métodos que ayudan a desarrollar Wireframes, Mockups y prototipos. [8]

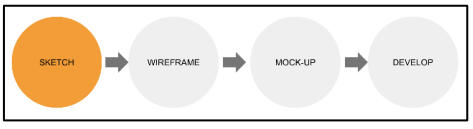

Fig. 5. Fases de diseño

Cabe destacar que el objetivo de la plataforma web ha sido, en todo momento, ofrecer a los usuarios una disposición eficiente de los datos y registros. No se ha buscado que tenga una apariencia destacable y captora de nuevos usuarios y, por lo tanto, se ha obviado el paso del Mockup, que busca precisamente eso.

6.2.1 Wireframe

El Wireframe [9] se ocupa de dar un diseño y definir una estructura básica de la web. Se suele utilizar con escala de grises para no destacar tanto la apariencia y centrar la discusión en el concepto, estructura y componentes básicos. La forma más eficiente y económica de crear un Wireframe es haciéndolo a mano, con papel y lápiz, dando la opción de realizar cambios de forma rápida.

Como punto de partida se ha diseñado la interfaz principal con la organización de los ítems de la página y el método de navegación de ésta.

Diseño página principal

En primer lugar, se ha creado la página principal con los elementos shared para el resto de las vistas de la web:

- Menú horizontal para seleccionar el idioma, acceder al cierre de sesión y al logo de la empresa, para ir a la página principal.
- Menú vertical navegable para acceder a cada una de las funcionalidades de la web.

Ver figura 22 en el apéndice de diseño.

Vista de informe de facturas

En segundo lugar, se ha diseñado una vista de ejemplo de informe de las facturas en el que vemos los elementos shared, comentados anteriormente, y el contenido de la vista en cuestión:

- Cajas de texto para aplicar filtros en los resultados que se muestran en la tabla, mostrando los datos relevantes de las facturas.
- Botón adicional en la esquina superior derecha para poder refrescar el contenido.

Ver figura 23 en el apéndice de diseño.

6.2.2 Prototipo

El paso siguiente ha sido crear el prototipo de la web. Estos son representaciones de alta fidelidad que simulan la interacción con la interfaz y que permiten al usuario experimentar la experiencia de uso.

Para su creación existen varias herramientas y aplicaciones muy versátiles y con mucha flexibilidad entre las que destacan Webflow y Axure. Ésta última, ha sido la escogida en este proyecto, puesto que ofrece a los estudiantes una licencia de forma gratuita. [10]

La aplicación Axure es muy completa y ofrece una amplia variedad de opciones, aunque es algo compleja. Permite crear las vistas de la página web una por una u organizadas en carpetas, da la posibilidad de crear plantillas compartidas o elementos shared, permite crear todo tipo de interacciones y vínculos y da, en todo momento, la posibilidad de visualizar el resultado en el navegador.

Creación página principal

Como se ha comentado anteriormente, la aplicación ofrece la posibilidad de crear elementos compartidos entre todas las vistas, evitando tener que ir creándolos en cada una de ellas y sin estar sujeto a tener que hacer cambios, si los precisa, en cada una de ellas. Es por eso por lo que lo primordial ha sido crear los menús de la barra superior y la barra lateral. Ver figura 24 en el apéndice de diseño.

Diseño de las vistas

El paso siguiente, ha sido el diseño de cada una de las vistas que se van a desarrollar en la web. Para ello, en el panel lateral de Axure se pueden organizar las vistas en carpetas, por ejemplo para organizarlas según el ítem del menú seleccionado. Una vez creadas todas las vistas, se han asignado las interacciones entre los elementos de las vistas y los vínculos entre los diferentes ítems del menú.

Ver figura 25 en el apéndice de diseño.

Por otro lado, Axure da la posibilidad de compartir un proyecto o diseño con otros usuarios y facilita el trabajo paralelo entre miembros de un mismo equipo.

El enlace del diseño es https://qejjfn.axshare.com (contraseña: TFGALEX) .

6.3 Creación de la plataforma web

Para implantar la información obtenida de los requisitos y orquestada en el diseño, en esta fase se detalla los pasos seguidos durante la programación de la plataforma web.

Como punto de partida, se ha establecido DDD [11] como enfoque de desarrollo del sistema en una arquitectura de web MVC.

Las ventajas de utilizar este enfoque es que se obtiene una mejor perspectiva a nivel de colaboración entre expertos del dominio y los desarrolladores, para concebir un software con los objetivos bien claros.

La estructura de DDD se aprecia a continuación:

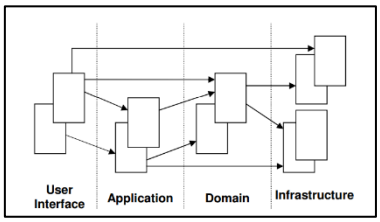

Fig. 6. Estructura DDD

El concepto principal de DDD es que el Dominio quede representado de forma clara y concisa en la estructura del software. Los modelos y reglas de negocio se incluyen en el core de la aplicación.

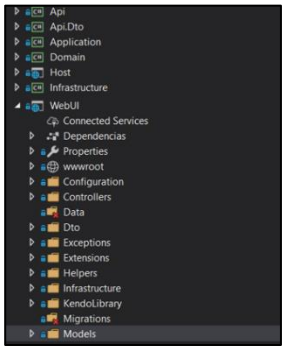

Fig. 7. Estructura del código

Como se aprecia en la Figura 7, la estructura del código se divide en cuatro puntos, a parte del host y la API.

- Application se encarga de realizar las operaciones entre el Domain y la Web
- Domain representa todas las entidades necesarias para representar los modelos y reglas de negocio.
- Infrastructure se encarga de acceder a bases de datos y elementos externos.
- WebUI es la interfaz con la que el usuario interactúa y se encarga de mostrar todos los datos.

6.4 Creación de la base de datos

Como se ha comentado anteriormente, Microsoft Azure da la posibilidad de acceder a numerables recursos y servicios. El servicio empleado para la creación de la base de datos ha sido SQL Database junto con SQL Server [13].

El DTU y el almacenamiento son de 10 y 250GB respectivamente. Se ha escogido un tipo de servicio estándar puesto que el proyecto no manejará grandes cantidades de datos como, por otro lado, sí lo haría en la práctica.

El uso del servidor de SQL es el de hacer la base de datos accesible desde la aplicación.

El siguiente paso ha sido la creación de las tablas en la base de datos. Para ello se ha empleado el software de Microsoft SQL Server Management Studio que permite la conexión a la base de datos montada sobre el servidor creado en el paso anterior.

Después de la creación de las tablas y sus relaciones, el diagrama de la base de datos [14] es el que se observa en la Figura 36 del apéndice de base de datos.

6.5 Creación de la API

Para la realización de cualquiera de las acciones de nuestra web se ha de interactuar con la base de datos para leer, editar o eliminar cualquier tipo de registro. De la misma manera y como previamente se ha comentado, Logic Apps tiene una gran flexibilidad y eficiencia a la hora de trabajar con llamadas HTTP y es por eso por lo que se necesita la creación de una API.

Una API es un conjunto de definiciones y protocolos que se emplean para desarrollar e integrar el software de las aplicaciones. Permiten que los productos y servicios se comuniquen con otros sin tener la necesidad de saber cómo están implementados. A parte de la simplicidad que esto aporta, supone un ahorro de tiempo y dinero.

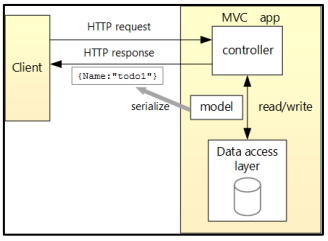

Fig. 8. Esquema API

Para satisfacer las solicitudes de obtener, editar y eliminar datos de la SQL Database, sobre SQL Server, se ha implementado la API web sobre ASP.NET Core [15] para que sea compatible con el front-end, es decir la plataforma web. Al haber desarrollado desde el enfoque de DDD y poder respetar la independencia de cada una de las partes, la API REST se encarga únicamente de recibir peticiones del front-end y ejecutar, mediante el patrón de diseño Mediator, llamadas a la capa de Application, que posteriormente accederá a Infrastructure para acceder a la base de datos. Más adelante se concretará con ejemplos el flujo de ejecución del back-end.

Puntos de acceso API

Para satisfacer todas las solicitudes del front-end para obtener elementos de la base de datos o actuar en ella (crear, actualizar o eliminar), existen una serie de puntos de acceso organizados según las tablas en las que se requiere acceso.

Como vemos en la imagen a continuación, los puntos de acceso se dividen en cinco archivos que actúan de controladores:

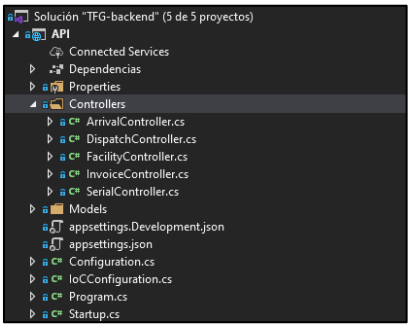

Fig. 9. Organización puntos de acceso

Para la gestión de las peticiones y gestionar los accesos a la base de datos, manteniendo la independencia entre componentes, se emplean dos patrones de diseño principales a lo largo de todo el código.

• Inyección de dependencia

Es el encargado de suministrar objetos a una clase en lugar de ser la propia clase que cree dichos objetos. Este patrón es parte de uno de los cinco principios de diseño de clases conocido como SOLID [16][17].

• Mediator [18]

Es el encargado de comunicar a los Handlers la operación a realizar, pasándoles la información necesaria para realizarla. Como se ha explicado anteriormente, este patrón se utiliza mediante la inyección de dependencia.

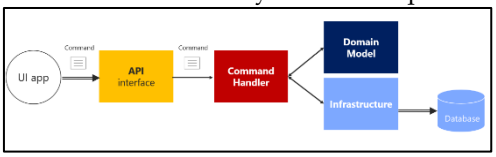

Fig. 10. Esquema de Mediator

Mediante un Command Handler (el receptor del command generado en el controlador), la API es capaz de pasar la información necesaria para obtener los datos o actuar sobre las tablas concretas de la Base de datos, respetando la independencia entre cada uno.

Para una mejor explicación, se simulará el flujo de trabajo de una llamada en el controller Arrival para recibir un resumen de las recepciones registradas [20].

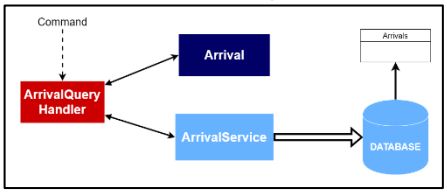

Fig. 11. Esquema de ArrivalController

Como se observa en la figura anterior, en este caso la API será llamada en el ArrivalController. En él se creará una query, con la petición recibida, que se ejecutará y se recogerá el *handler*. Éste será el encargado de generar el objeto del domain con la query, acceder al service de Infrastructure y acceder a la base de datos desde allí, a la tabla correspondiente, en este caso la de Arrivals. Una vez obtenidos los datos, se generará la respuesta HTTP para el cliente.

Pruebas y resultados API

Para ayudar en el desarrollo y comprobar que la API funciona correctamente, es decir recibe el formato de petición correctamente y devuelve el resultado esperado, se ha utilizado Postman[19], una extensión del navegador Google Chrome, que permite el envío de peticiones HTTP REST sin necesidad de ejecutarlas desde un cliente.

Una vez se ejecuta la API, ésta permanece a la espera de recibir algún tipo de llamada HTTP.

A continuación, se muestra la ejecución de la llamada GET a api/Arrival/Summary con sus pertinentes parámetros:

https://apitfgalex.azurewebsites.net/api/arrival/Summary?CreationDateFrom=2020-01-01&CreationDateTo=2020-08-08&id=0&fid=

Fig. 12. Respuesta API

Como se observa, en el body con la respuesta aparecen los Arrivals registrados hasta el momento (en este caso 1). El tiempo de respuesta ha sido 1956 ms y el código 200 OK.

En el caso de registrar un nuevo Arrival, se genera un body con la información necesaria y se ejecuta una llamada POST a

https://apitfgalex.azurewebsites.net/api/arrival/api/arrival/Create

Fig. 13. Respuesta POST

Como se puede ver, se recibe la respuesta de la API indicando que no hay ningún error y especificando el numero de la Arrival que se ha registrado en la referencia.

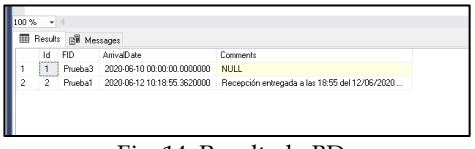

Fig. 14. Resultado BD

Si, posteriormente, se comprueba la BD en busca del Arrival nuevo, se aprecia que éste si se ha registrado con éxito.

6.6 Deploy

Como se ha explicado en los puntos anteriores y una vez implementado el front y el back end, el siguiente paso ha sido desplegar la plataforma y la API para darles una disponibilidad global de forma externa. [21]

Como otros componentes del proyecto, se ha utilizado un servicio de Microsoft Azure, en este caso App Service. Con este servicio, se puede realizar una migración y traer el código del proyecto a la nube, sin aplicar cambios en él.

Entre las características más destacables de este servicio se encuentran la fácil gestión y administración, mediante la interfaz de recursos de Azure, y la integración y despliegue continuo CI/CD, que se comentará más adelante.

En la siguiente figura se observa la información general una vez el App Service está creado. Como puede observarse, la URL donde se ha desplegado es

https://tfgalex.azurewebsites.net

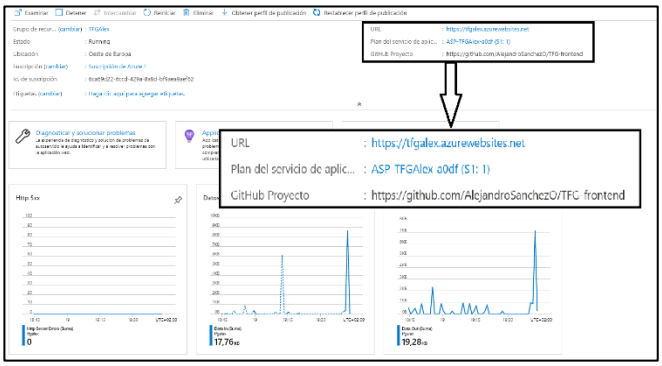

Fig. 15. Despliegue.

Para el despliegue continuo, App Service da la opción de asignarse a un repositorio. Cualquier push que se efectúe en https://github.com/AlejandroSanchezO/TFG-frontend hará que el App Service implemente el proyecto automáticamente. Para ello, en el repositorio hay dos ramas, una llamada master donde se desarrolla y otra de deploy, que está conectada con el App Service.

Desde la interfaz de App Service se pueden gestionar las versiones implementadas del proyecto. La siguiente figura muestra los últimos cambios efectuados:

Fig. 16. Implementaciones del proyecto en App Service

6.7 Logic Apps

Como se ha explicado al inicio de este documento, la segunda parte del proyecto ha consistido en automatizar procesos de generación de informes de los registros a partir del uso de Logic Apps. [22]

Funcionamiento y flujo

Para entender el flujo de operación de las Logic Apps generadas, se comentará un caso específico, ya que todas funcionan de forma similar pero acceden a entidades diferentes.

LogicApps para las facturas

El objetivo es, una vez aplicados los filtros en la pantalla de informes de facturas, poder generar archivos docx que recojan los resultados obtenidos del filtro. El documento generado se enviará por correo electrónico a la dirección que se facilite. Además, como modo de suscripción, se puede especificar la periodicidad con la que se quiere recibir este documento, con los resultados actualizados. Es decir, se puede generar la suscripción a un tipo de informe para recibir las facturas filtradas en un intervalo de tiempo a seleccionar.

Para ello, se emplean tres tipos de Logic Apps conectadas entre ellas.

UpdateDocInvoices

Esta Logic App es la encargada de ser el punto de conexión con la plataforma web.

Como vemos en la figura 26 del apéndice de Logic Apps, la acción desencadenante, es decir que ejecuta el flujo, es una llamada POST, que recibe la solicitud.

Seguidamente, se genera el JSON con la petición. Para generar el archivo *docx*, se tiene que rellenar una plantilla que, con Visual Basic[23] , completa los campos con el JSON del POST. Una vez generado el documento nuevo se guarda en Google Drive.

Después de generar el documento, se envía éste por correo electrónico a la dirección especificada.

La segunda parte de este flujo es el encargado de comprobar si se ha especificado una suscripción al informe. Ver figura 27 del apéndice de Logic Apps.

En ese caso entra en un bucle y llamada a la segunda Logic App, explicada a continuación.

BucleInvoiceSummary

Esta segunda Logic App se activa cuando se especifica la periodicidad con la que se desea recibir el informe al correo electrónico.

Ver figura 28 del apéndice de Logic Apps.

La última llamada HTTP del flujo se encarga de llamar a la última Logic App, encargada de enviar el correo electrónico.

EnviarCorreo

Como se muestra en la figura anterior, este flujo solo se encarga de enviar el correo electrónico.

Ver figura 29 del apéndice de Logic Apps.

6.7.1 Resultados

Para mostrar los resultados obtenidos se mostrarán capturas de pantalla de todo el proceso desde que se aplican los filtros hasta que se recibe el correo electrónico con el informe. Consultar apéndice de Resultados Logic Apps.

6.7.2 Listado Logic Apps

Con la ejecución anterior se muestra el flujo completo para las facturas y la generación de sus informes. De la misma manera y como se muestra en la siguiente figura, todas las entidades de la plataforma web funcionan de forma similar pero utilizando sus llamadas a la API y uso de plantillas pertinentes.

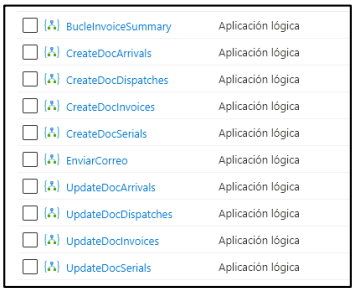

Fig. 17. Listado Logic Apps

6.7.3 Monitorización y testeo

Tal y como se ha comentado en el apartado de la API, la parte de testeo ha consistido en hacer pruebas de usuario y manuales, aprovechando la interfaz de monitorización que ofrece Azure y que permite detectar los errores de las ejecuciones, tanto de la plataforma web, de la API y de Logic Apps. Un ejemplo sería que la ejecución de la Logic App se detiene y avisa que la petición es incorrecta. De esta forma sabemos que parte de la ejecución revisar de una forma casi automatizada. Ver apéndice de monitorización y testeo.

Para la monitorización más específica, se puede acceder a la parte de supervisión. Allí se observan varias gráficas personalizables. En este caso, muestran el resultado de la ejecución durante las últimas 3 horas, las ejecuciones facturables y la latencia de éstas.

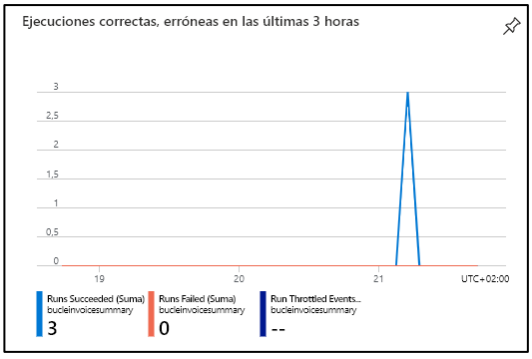

Fig. 18. Gráfico 1

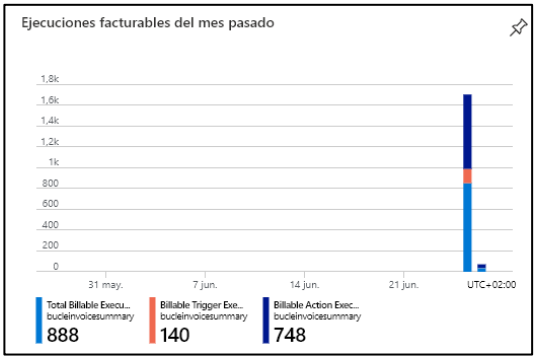

Fig. 19. Gráfico 2

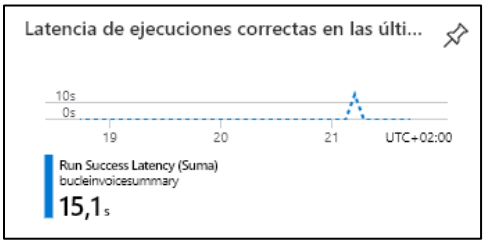

Fig. 20. Gráfico 3

7 Presupuesto

Después de la implementación, puesta en producción y posterior ejecución y pruebas, se ha previsto un gasto aproximado del proyecto en cuestiones de mantenimiento. A través de la herramienta de análisis de costos se ha estimado que el conjunto de todos los recursos implementados suma una cantidad de 72.02€. La previsión al mes es de 80€.

Esta previsión incluye:

- Logic App
- Plan de suscripción para las API
- Base de datos
- App Service

El costo desglosado queda de la siguiente manera:

Recurso	Resource type	Cost
asp-tfgalex-b827	Plan de App Service	€35.79
asp-tfgalex-a0df	Plan de App Service	€35.79
enviarcorreo	Aplicación lógica	€0.17
tfg-getinvoicesreport	Aplicación lógica	€0.12
updatedocinvoices	Aplicación lógica	€0.07
asp-tfgalex-bd0a	Plan de App Service	€0.05
bucleinvoicesummary	Aplicación lógica	€0.03
asp-tfgalex-ae7d	Plan de App Service	€0.03
updatedocarrivals	Aplicación lógica	€0.01
createdocarrivals	Aplicación lógica	<€0.01
getinvoicesreportbypost	Aplicación lógica	<€0.01
updatedocserials	Aplicación lógica	<€0.01
updatedocdispatches	Aplicación lógica	<€0.01
createdocinvoices	Aplicación lógica	<€0.01
createdocserials	Aplicación lógica	<€0.01
createdocdispatches	Aplicación lógica	<€0.01
apitfgalex	Application Insights	€0
tfgalex / tfgalex	SQL Server	€0
apitfgalex	App Service	€0

Fig. 21. Costos

Observando la tabla anterior, se debe destacar que el mayor coste proviene de la contratación del servicio para cada App Service por 35€ cada uno, que puede variar considerablemente en una situación en la que se necesite que accedan varios usuarios.

La base de datos y su servidor suponen 0€ de gasto, puesto que viene contratadas con un servicio básico.

Las Logic Apps, por otro lado suponen un coste muy bajo puesto que suponen un coste por uso, y, como se ha comentado anteriormente, solo ha accedido a la aplicación un usuario de forma concurrente. En caso de que se activaran las suscripciones y accedieran miles de usuarios, este precio podría aumentar considerablemente. Aun así, siguen siendo una opción muy eficiente para este tipo de aplicaciones.

8 Conclusiones

Como ya se especificó al inicio del documento, este proyecto no ha sido solo un Trabajo de Fin de Grado, ha sido un desarrollo de software analizado, diseñado e implementado siguiendo los pasos del software integrado de una empresa real. El proceso de automatización de procesos en las llamadas a la API es la motivación por la cual se han desencadenado la consecución de tareas por cada etapa de la vida del software.

La fase de planificación del proyecto, una etapa clave para la gestión de los tiempos, se realizó en la fase más temprana y se ha ido actualizando a la vez que lo hacía el proyecto. Por un lado, el tiempo estipulado del proyecto en un inicio no se sopesó con total certeza debido a la falta de experiencia y a la situación extraordinaria vivida durante su desarrollo. Por otro lado, se aplicaron los errores de cálculo a tiempo para poder obtener un resultado funcional y lo más eficiente posible:

- Durante algunas fases, el tiempo calculado para algunas tareas fue muy optimista. Concretamente en la codificación del front-end de la aplicación.
- Las etapas del ciclo de vida del software no se siguieron totalmente de forma cronológica como se representa en este documento. Durante el desarrollo se tuvo que volver a la fase de diseño más de una vez para corregir ambigüedades y errores. La fase de requisitos ha sido modificada sobretodo en la fase de desarrollo del front-end, para ajustarse al resultado que realmente se requería.

Como último aspecto, me gustaría hablar en primera persona y expresar lo que ha significado el desarrollo de este proyecto. Gracias a este trabajo y durante estos meses, mi interés en relación con el Cloud Computing ha ido creciendo hasta el punto de descubrir un conjunto de recursos y servicios disponibles para gestionar proyectos, monitorizar su desarrollo y ejecución, y varias aplicaciones futuras para todos ellos. En definitiva, un gran abanico de posibilidades para proyectos futuros.

Para mí el proyecto ha supuesto un primer contacto en el deploy de una plataforma web con su API, la gestión y automatización de la integración continua, la monitorización de su ejecución, etc. En cuanto al desarrollo, me ha servido para poner a prueba los conocimientos obtenidos a lo largo del grado y la especialización en Ingeniería del Software, como el control de versiones, requisitos del software, documentación, gestión y administración de bases de datos, etc.

Como puntos negativos que me gustaría destacar, la planificación tan optimista dada en un inicio ha supuesto una piedra en el camino que ha provocado la reorganización de las tareas a realizar y abstracción del trabajo. La carga de trabajo manejada en un inicio no se valoró correctamente hasta llegar a un punto muy avanzado en el desarrollo.

Como evaluación personal, considero que el trabajo ha sido bueno y la motivación llevada durante estos meses ha significado un gran punto a favor. Como aspecto de aprendizaje, personalmente ha significado un gran salto para mí.

9 Líneas futuras y puntos para mejorar

Finalmente se hace un balance de las líneas de trabajo futuro en las que se incluyen los siguientes puntos:

- Se debe destacar la necesidad de implementación de test, que durante todo el proyecto ha sido manual y user testing.
- La finalización de algunos componentes, que finalmente no se han desarrollado para reducir la carga de trabajo, también es un aspecto para revisar:
  - **Informes**. Sistema de reportes con los resultados de las operaciones efectuadas:
    - Introducir sesiones de usuario y autenticación. Envíos
    - Localización para traducir los textos del cliente, es decir un sistema de traducción
    - Permitir una total administración al usuario en cuanto a las suscripciones a las Logic Apps
    - Asegurar responsive web, que hasta el final no ha significado un punto para tener en cuenta, ya que la plataforma web era solamente un punto de inicio para empezar a trabajar con la API y los servicios cloud como Logic Apps. Es decir, no se pretendía desarrollar una página web para atraer nuevos usuarios, sino una que fuese lo más funcional y eficiente posible para los reales usuarios, empresas relacionadas con logística, que necesitan registrar facturas, envíos, etc.
- Valorar alternativas a Microsoft Azure. No por muy buenos resultados que haya dado esta elección significa que sea la mejor de todas. Un estudio de alternativas que ofrezcan cloud computing puede significar un manejo más amplio y una capacidad de desarrollo mayor.

10 Agradecimientos

Para empezar, me gustaría agradecer a Zetes, la entidad donde empecé en septiembre del 2019 las prácticas curriculares. A parte de darme la idea del TFG y ofrecerme un empleo, han sido pacientes con mi aprendizaje y me han dejado formarme durante estos meses. Al tutor de la empresa, Dani, quería agradecerle su ayuda en la acotación del trabajo a realizar, su aportación de ideas para éste y su comprensión en estos meses de mucho trabajo y estrés.

En segundo lugar, quisiera agradecer a mis seres queridos, mi familia y amigos, que me han acompañado durante estos meses y han hecho posible que haya llegado hasta aquí.

Por último, un agradecimiento especial a mis padres, por el apoyo, motivación y comprensión constante durante todos estos años de formación.

ÍNDICE DE FIGURAS

BIBLIOGRAFIA

[1] Portal de Microsoft Azure
https://portal.azure.com/#home
[2] Información acerca de Azure
https://azure.microsoft.com/es-es/services/logic-apps/
[3] Documentación Logic Apps
https://docs.microsoft.com/es-es/azure/logic-apps/logic-apps-overview
[4] Metodología Waterfall vs Agile
https://thedigitalprojectmanager.com/es/agile-frente-a-waterfall/
https://www2.deloitte.com/es/es/pages/technology/articles/waterfall-vs-agile.html
https://www.esan.edu.pe/apuntes-empresariales/2017/05/proyectos-en-metodologia-waterfall-o-agile-hacia-donde-vamos/
[5] Información de Metodología Waterfall
http://design-toolkit.recursos.uoc.edu/es/waterfall/
[Recursos de la asignatura de Gestión de Proyectos e Ingeniería del Software]
[6] Gestión de recursos
https://www.wrike.com/es/blog/que-es-la-gestion-de-recursos-y-por-que-es-importante/
https://www.ceupe.com/blog/la-gestion-de-los-recursos.html
[7] Partes del diseño
https://es.wikipedia.org/wiki/Dise%C3%B1o_web
https://neoattack.com/neowiki/diseno-web/
[8] Wireframes, Mockups y Prototipos
https://medium.com/rocket-studio-ux/wireframe-mockup-y-prototipos-en-busca-de-sus-diferencias-23a03bcbdb69
[9] Wireframes
https://webdesdecero.com/wireframes-que-son-y-como-crear-los/
[10] Herramientas de prototipado
https://webdesdecero.com/wireframes-que-son-y-como-crear-los/
https://www.axure.com/
https://docs.axure.com/axure-rp/reference/getting-started-video/
https://neoattack.com/proyectos/
[11] DDD
https://medium.com/@jonathanloscalzo/domain-driven-design-principios-beneficios-y-elementos-primera-parte-aad90f30aa35
https://devexperto.com/domain-driven-design-1
[12] Trello
http://trello.com
[13] Creación de la base de datos
https://portal.azure.com/#create/Microsoft.SQLDatabase
https://www.youtube.com/watch?v=QrCa25p4ixE
[14] Diagrama de la base de datos
https://www.youtube.com/watch?v=KECvDwc9E3A
[15] Creación de la API web con ASP.NET Core
https://medium.com/@azaharafernandezguizan/8-pasos-para-crear-una-web-api-con-net-core-25323708cac0
https://docs.microsoft.com/es-es/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1&tabs=visual-studio
[16] Inyección de dependencias
https://www.arquitecturajava.com/el-patron-de-inyeccion-de-dependencia/
[17] Principios SOLID
https://profile.es/blog/principios-solid-desarrollo-software-calidad/#:~:text=Los%205%20principios%20SOLID%20de,%E2%80%93%20Liskov%20Substitution%20Principle%20(LSP)
[18] Mediator
https://docs.microsoft.com/es-es/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/microservice-application-layer-implementation-web-api
https://www.programmingwithwolfgang.com/mediator-pattern-in-asp-net-core-3-1/
[19] POSTMAN
https://profesores.virtual.unian-des.edu.co/~isis2603/dokuwiki/doku.php?id=tutoriales:postman
https://www.postman.com/
[20] Esquema Flujo BackEnd
https://app.diagrams.net/
[21] Logic Apps
https://docs.microsoft.com/es-es/azure/logic-apps/
[22] App Service
https://azure.microsoft.com/es-es/services/app-service/
https://channel9.msdn.com/Blogs/msmdotnet/Qu-es-App-Service
[23] Crear documentos con plantillas y Visual Basic con PLUMSAIL
https://plumsail.com/docs/documents/v1.x/search.html?q=logic+apps+word#

APÉNDICE

A1. DISEÑO WIREFRAME Y PROTOTIPO

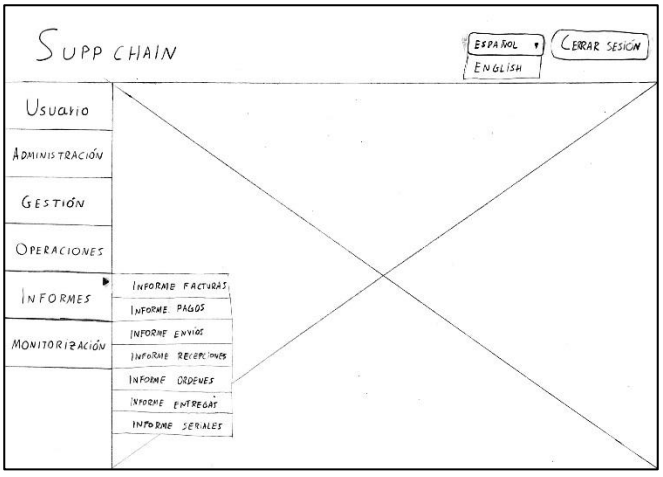

Fig. 22. Wireframe página principal

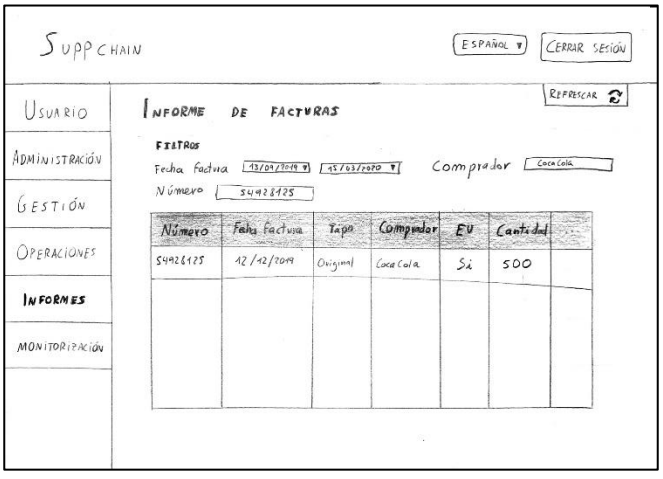

Fig. 23. Wireframe informe de facturas

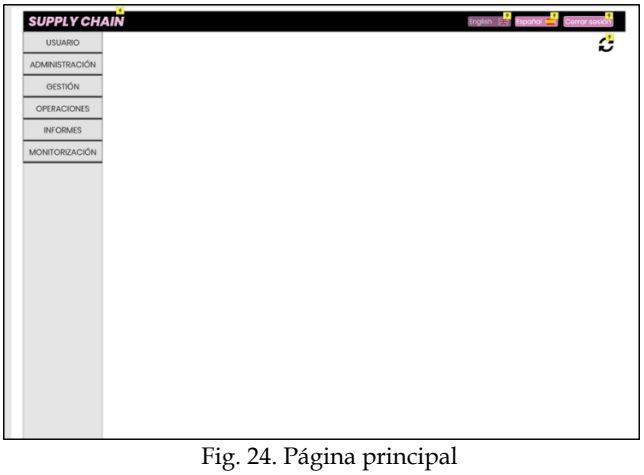

Fig. 24. Página principal

Fig. 25. Diseño registro facturas

A2. LOGIC APPS

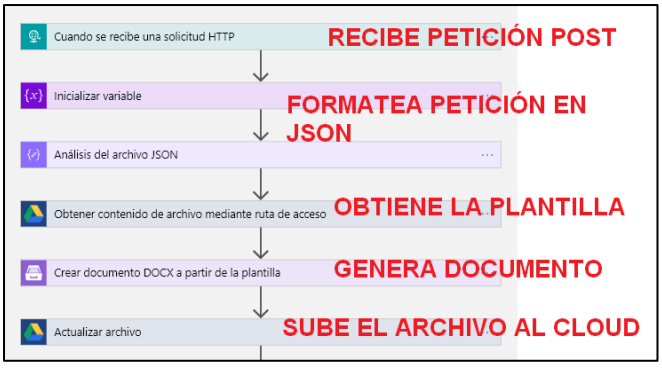

Fig. 26. UpdateDocInvoices

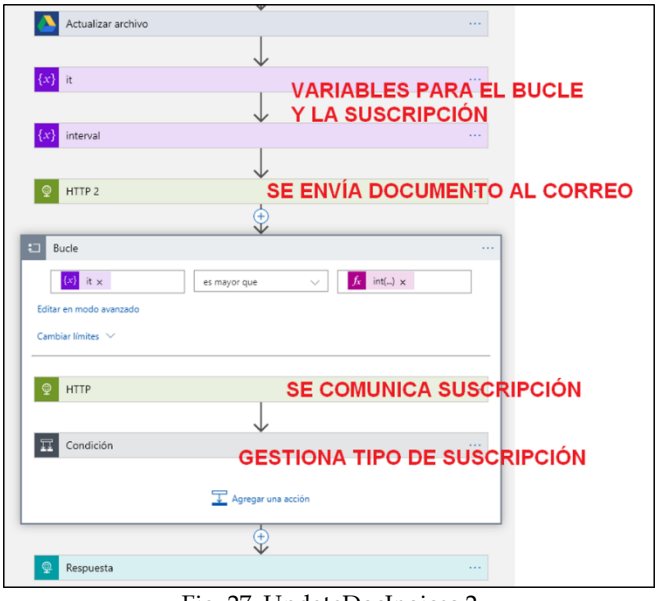

Fig. 27. UpdateDocInoices 2

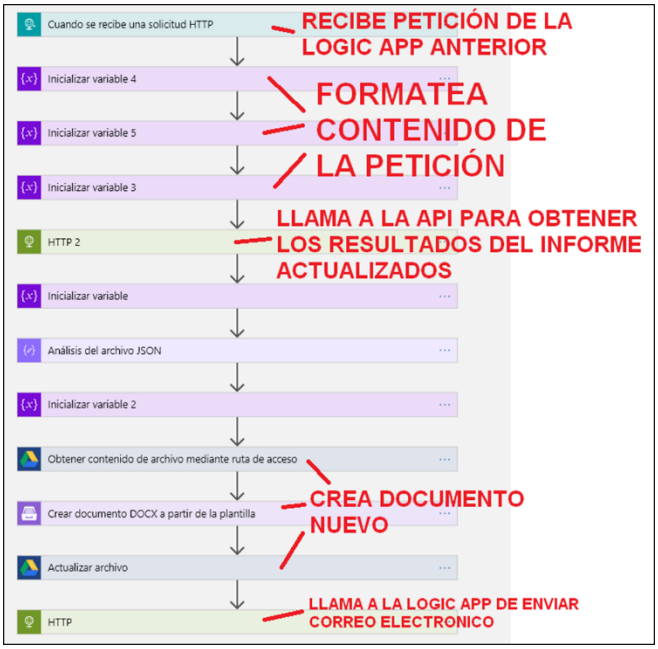

Fig. 28. BucleInvoiceSummary

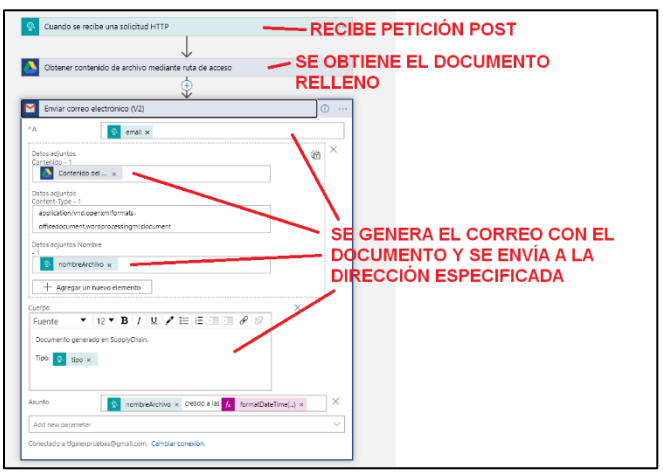

Fig. 29. Enviar correo

A3. RESULTADOS LOGIC APP

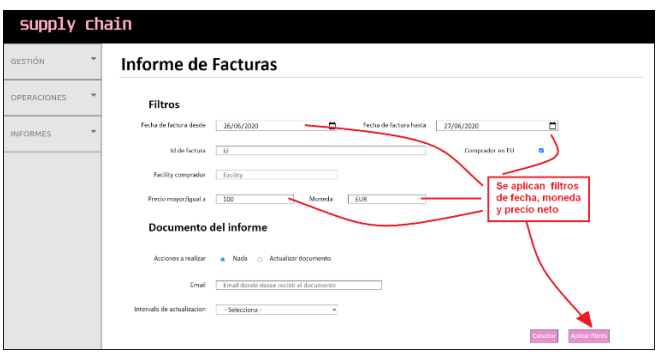

Fig. 30. Aplicar filtros

Fig. 31. Solicitud enviada a Logic App

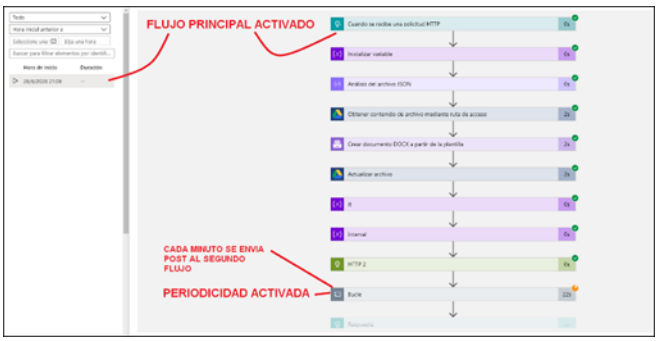

Fig. 32. Flujo principal activado.

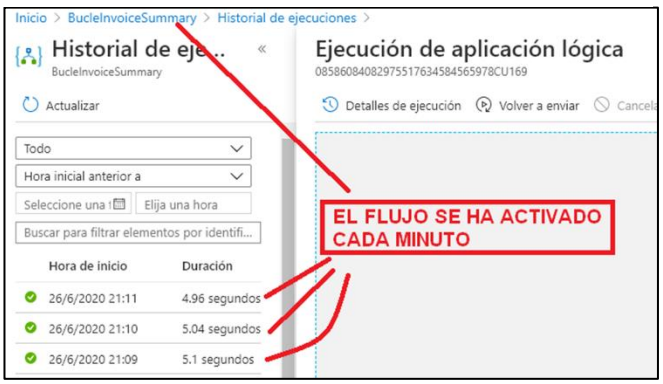

Fig. 33. Flujo 2

Fig. 34. Correos recibidos

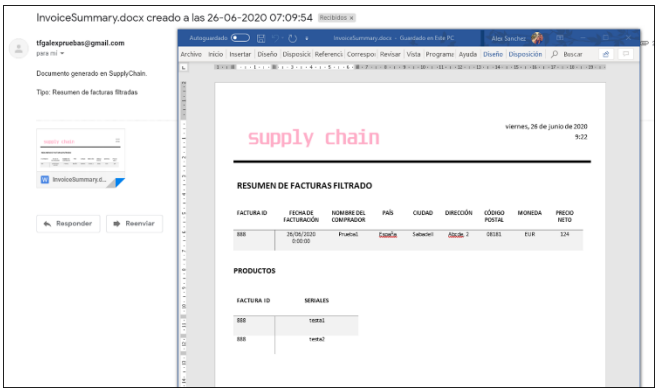

Fig. 35. Documento rellenado correctamente.

El flujo seguido en las figuras de los resultados de las Logic App es el siguiente:

1. Se accede a la pantalla de informe de facturas, en la que ya habrá varias facturas registradas. Se aplican los filtros deseados.
2. Se refresca la tabla con los resultados de facturas filtrados obtenidos. Se especifica un correo donde se desea recibir el informe y se establece una periodicidad de un minuto. Es decir, se recibirá el correo cada minuto con los resultados filtrados y actualizados.
3. Se activa el flujo principal, es decir, la Logic App encargada de realizar todo el proceso de generar el documento a partir de la llamada POST recibida de la web.
4. Se activa la segunda Logic App y que, como se ve en la figura 33, se ejecuta 3 veces, una vez cada minuto.
5. Cada minuto que se ha ejecutado, la Logic App de enviar correo se ha activado y ha enviado el correo a la dirección establecida anteriormente.
6. Por último, en la figura 35, se puede observar el documento con los resultados filtrados.

A3. Base de datos

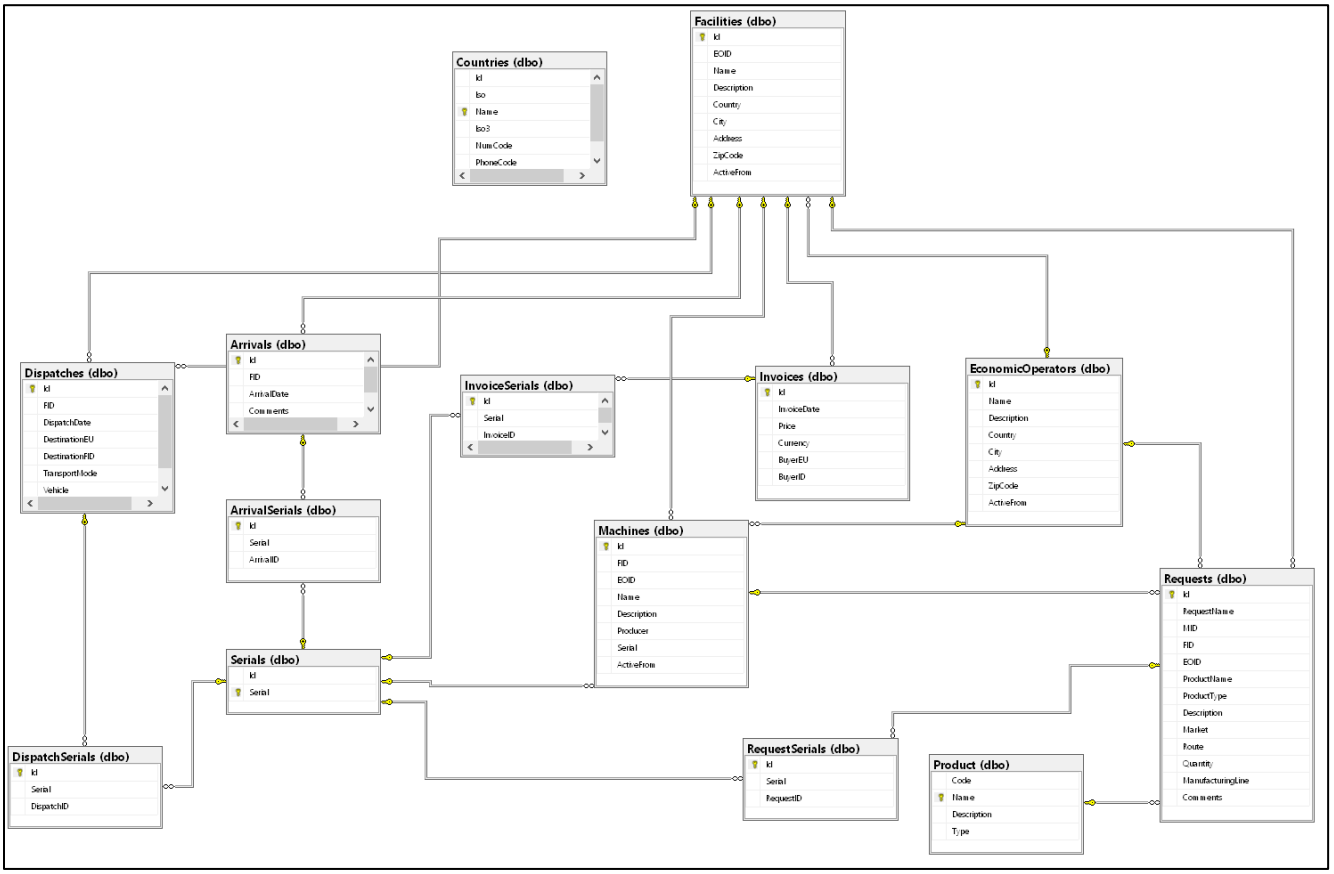

Fig. 36. Diagrama de base de datos