PROYECTO FIN DE GRADO

TÍTULO: **Diseño e implementación de una aplicación de gestión de información empresarial en la nube**

AUTOR/A: Diego Gutiérrez Pajor

TITULACIÓN: Telemática

TUTOR/A: David Meltzer Camino

DEPARTAMENTO: DTE

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Raquel Águeda Maté

TUTOR/A: David Meltzer Camino

SECRETARIO/A: Carlos Carrillo Sánchez

Fecha de lectura: 09/07/2024

Calificación:

El Secretario/La Secretaria,

Agradecimientos

Me gustaría agradecer a la escuela, a su profesorado y a mi tutor David Meltzer por permitirme realizar este PFG en el que he podido realizar un trabajo propio desde cero y hacer lo que realmente me apasiona que es la gestión de proyectos y su desarrollo. Además, agradezco a mi tutor su seguimiento constante y buenas recomendaciones.

Quisiera agradecer a mi colega Jorge que tras comentarle mi idea me recomendó las tecnologías de Flask y SQL-Alchemy para la realización del proyecto lo que dio un rumbo al desarrollo de la solución propuesta.

También especiales agradecimientos a mi familia y a Ian y Álex por hacer pruebas sobre la aplicación web y comprobar su funcionamiento. Gracias a Valeria por leer el TFG y ayudarme con correcciones de redacción y encontrando errores.

Muchas gracias a todos los que me han apoyado por confiar en mí en la carrera y por confiar que podría desarrollar este TFG exitosamente.

Resumen

Título: Diseño e Implementación de una Aplicación de Gestión de Información Empresarial en la Nube

Este trabajo de fin de grado se ha centrado en el diseño y la implementación de una aplicación web de gestión de información empresarial hospedada en la nube. La aplicación se ha diseñado en el marco de una empresa de mensajería que necesita gestionar la entrega de correos.

El propósito principal y el condicionante tecnológico ha sido el diseño de una aplicación web siguiendo el paradigma de la reutilización y la ventaja del uso de herramientas modernas. Se ha optado por utilizar tecnologías como el lenguaje de programación Python, debido a su simplicidad, su gran cantidad de librerías, extensa documentación y su capacidad de programación orientada a objetos, el *framework* Flask para el desarrollo de la web app y la extensión SQL-Alchemy para crear y trabajar con un modelo de datos y asegurar la persistencia entre la aplicación web y la base de datos.

Como arquitectura de diseño se ha usado Modelo Vista Controlador (MVC) que divide el sistema en tres componentes principales lo que facilita la modularización, la reutilización del código y el mantenimiento del software. Además, permite la evolución independiente de cada componente, lo que facilita la escalabilidad.

Se ha utilizado una plantilla de diseño basada en Bootstrap5 para el componente de Vista, lo que proporciona un diseño atractivo y una interfaz responsiva. Para el componente del Controlador se ha diseñado una interfaz de tipo API Rest para atender a las peticiones de la aplicación web y para el Modelo, se ha modelado mediante el uso de SQL-Alchemy un conjunto de clases que hacen referencia a los objetos de la aplicación.

Todos los componentes de la aplicación web han sido hospedados en la plataforma Microsoft Azure lo que proporciona una infraestructura confiable y escalable que garantiza un alto nivel de disponibilidad y rendimiento para la aplicación.

La metodología seguida se ha basado en una metodología SCRUM de trabajo real en la que se ha modularizado el trabajo en entornos de prueba y se han realizado entregas de forma secuencial para asegurar el funcionamiento correcto de cada una de las partes del proyecto.

Los resultados obtenidos incluyen un prototipo de aplicación web funcional que cumple con los requisitos establecidos hospedada en Microsoft Azure. Se concluye que la aplicación desarrollada ofrece una solución sólida y segura para la gestión de información empresarial en la nube, destacando la importancia de utilizar tecnologías adecuadas y la facilidad hoy en día de desarrollar este tipo de soluciones.

En conjunto, estos resultados destacan la importancia de utilizar tecnologías adecuadas y metodologías de desarrollo efectivas para ofrecer soluciones sólidas y seguras en el entorno empresarial actual.

Abstract

Title: Design and Implementation of a Cloud-Based Enterprise Information Management Application

This thesis has focused on the design and implementation of a cloud-hosted web application for enterprise information management. The application has been designed within the business framework of a courier company that needs to manage mail delivery.

The main purpose and technological constraint have been the design of a web application following the paradigm of reuse and the advantage of using modern tools. Technologies such as the Python programming language, due to its simplicity, extensive library support, comprehensive documentation, and object-oriented programming capabilities, were chosen. Additionally, the Flask framework was chosen for web app development and the SQL-Alchemy extension for creating and working with a data model to ensure persistence between the web application and the database.

The Model-View-Controller (MVC) design architecture has been used, dividing the system into three main components, which facilitates modularity, code reuse, and software maintenance. Furthermore, it allows independent evolution of each component, facilitating scalability.

A design template based on Bootstrap5 has been used for the View component, providing an attractive design and a responsive interface. For the Controller component, a REST API interface has been designed to handle requests from the web application, and for the Model, a set of classes referencing application objects has been modeled using SQL-Alchemy.

All components of the web application have been hosted on the Microsoft Azure platform, providing a reliable and scalable infrastructure that ensures a high level of availability and performance for the application.

The methodology followed has been based on a real-world SCRUM methodology in which work has been modularized into test environments, and deliveries have been made sequentially to ensure the correct functioning of each part of the project.

In conclusion, this project has demonstrated the feasibility and effectiveness of designing and implementing a cloud-based enterprise information management web application. By leveraging modern tools and following the Model-View-Controller (MVC) design architecture, the application achieves modularity, code reuse, and ease of maintenance. The use of technologies like Python, Flask, and SQL-Alchemy ensures a robust and scalable solution with seamless integration between the web application and the database. Hosting all components on the Microsoft Azure platform provides a reliable infrastructure with high availability and performance. The adoption of SCRUM methodology for real-world development ensures iterative progress and thorough testing, leading to a functional and reliable application.

Overall, this project highlights the importance of leveraging appropriate technologies and methodologies to deliver solid and secure cloud-based solutions for modern business needs.

Índice de figuras

Lista de acrónimos

CRUD - *Create, Read, Update and Delete*

IDE – *Integrated Development Environment*

ODBC – *Open Database Connectivity*

ORM – *Object Relational Mapper*

SLA – *Service Level Agreement*

Índice de contenidos

1. Introducción

1.1 Marco y motivación del proyecto

El presente proyecto se enmarca en el ámbito del desarrollo de aplicaciones web, específicamente en el sector de la gestión de la información y de los servicios de hospedaje en la nube. En la actualidad, disponemos de herramientas que facilitan el desarrollo de aplicaciones con lenguajes de programación como Python que abogan por la reutilización y la simplicidad. Por tanto, se pretende utilizar Python como lenguaje de programación moderno y herramientas como *frameworks* y librerías para desarrollar un prototipo de solución para la gestión de la información de una empresa de correos y mensajería con arquitectura en la nube.

El problema principal que se pretende resolver es la utilización de la arquitectura de aplicación clásica basada en Java con sistemas locales y persistencia basada en modelos mucho más complejos de lo necesario. Para este proyecto se pretende desarrollar una aplicación web desde cero utilizando nuevas tecnologías y demostrando la simplicidad y la facilidad que conlleva su uso. Además, se tratará el hospedaje en la nube y todas las ventajas que conlleva. En la figura se muestra un ejemplo de acceso a la aplicación hospedada en la nube en la que el único equipamiento necesario será un cliente con acceso a internet.

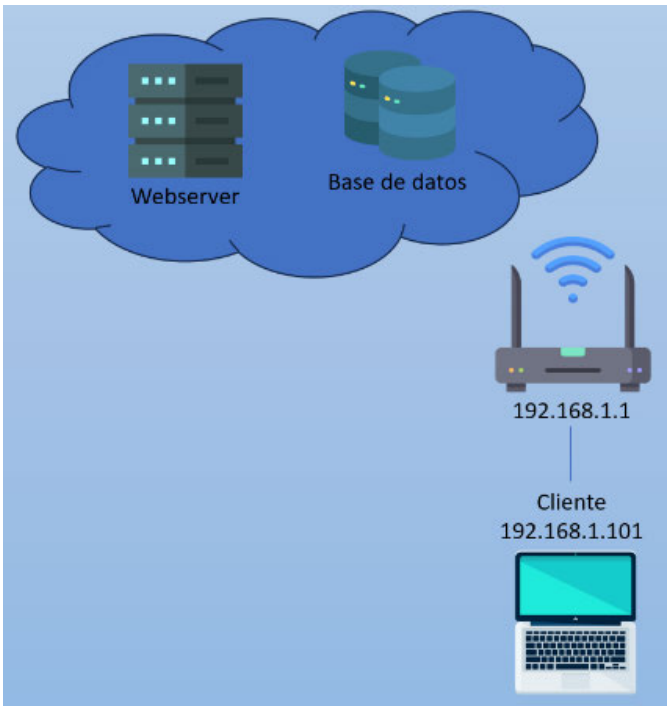


Ilustración 1. Ejemplo de acceso a una aplicación hospedada en la nube

Esta solución, propone el desarrollo de una aplicación web que centralice la gestión de los servicios de mensajería, permitiendo a la empresa coordinar y realizar un seguimiento de los envíos de manera eficaz. Asimismo, es una solución que supone un menor coste para una empresa nueva ya que el despliegue de servicios en la nube siempre conlleva un coste reducido en comparación con el despliegue de una infraestructura local.

La motivación del proyecto incluye una ambición personal basada en la aspiración del estudiante a crear una solución funcional para un problema de ingeniería trabajando en un ámbito de desarrollo real. La elección de un proyecto técnico de programación que incluye tecnologías Cloud ha sido realizada por el estudiante debido a su inclinación al trabajo en este sector.

1.2 Objetivos técnicos y académicos

Los objetivos de este proyecto fin de carrera son, desde el punto de vista técnico:

- Consolidación de los conocimientos en programación orientada a objetos, creación de un modelo de datos y el correcto funcionamiento de las clases. Trabajado a lo largo de la carrera en las asignaturas de programación y lenguajes de modelado.
- Consolidación de los conocimientos en programación y específicamente en Python, trabajados en la asignatura de programación multiparadigma para las TIC.
- Consolidación de los conocimientos sobre arquitecturas de aplicación vistas en programación de aplicaciones avanzadas.
- Despliegue de componentes en una plataforma de hospedaje Cloud, trabajado teóricamente en la asignatura de *Introduction To Artificial Intelligence In The Cloud*.

Desde el punto de vista académico, el proyectista adquiere las siguientes competencias y habilidades:

- Investigación y autoaprendizaje: se investigarán las posibles tecnologías para el desarrollo de la aplicación, eligiendo de forma objetiva la que mejor se adapta a la solución. Además, se estudiarán las tecnologías y se comprenderá su uso con la documentación disponible.
- Se seguirá una metodología de trabajo secuencial y ordenada, haciendo posible el seguimiento del desarrollo para una fácil redacción de la memoria.
- Se consolidan las competencias adquiridas en la carrera, tanto técnicas, como competencias relacionadas con el autoaprendizaje y la investigación.

1.3 Estructura del resto de la memoria

En los próximos capítulos de la memoria se ofrece información sobre el marco tecnológico y el estado del arte de las tecnologías implicadas, lo que incluye un breve resumen de la utilidad de cada una de las herramientas y una breve descripción de todas las tecnologías disponibles. En este apartado se utilizará la bibliografía correspondiente para explicar arquitecturas de aplicaciones web, APIs y *frameworks* de desarrollo, extensiones, librerías de Python y plataformas de hospedaje en la nube.

A continuación, se explicará el caso de uso propuesto con las especificaciones y los requisitos de diseño para abordar el problema de ingeniería. Para esta parte de la memoria se abordará el

problema de los modelos de aplicación de gestión de información empresarial, dónde una nueva empresa debe decidir sobre qué modelo usar.

Una vez expuesto el caso de uso, se procederá con la descripción de la solución, dónde se incluirá la justificación de todas las tecnologías elegidas y un abordamiento punto por punto en el que se explicará detalladamente la solución por la que se ha optado y cómo se ha desarrollado. El trabajo se ha dividido en 4 fases, fase de investigación, dónde se han buscado soluciones y tecnologías a utilizar; fase de modelado, dónde se ha definido un modelo de datos para la aplicación; fase de desarrollo, dónde se ha realizado el desarrollo de la aplicación web y se ha codificado todo lo necesario para su funcionamiento y fase de migración, dónde pasamos de trabajar de forma local a realizar una migración a una plataforma de hospedaje Cloud.

La parte de resultados muestra capturas que verifican el funcionamiento de la aplicación web y explica el trabajo realizado durante el proceso de desarrollo. Hace referencia a las fases explicadas de desarrollo y muestra los resultados de cada una de ellas. Posteriormente se incluye un análisis de presupuesto, dónde se explica el coste monetario que ha implicado el proyecto y el coste que constituiría su mantenimiento anual.

Seguidamente, el impacto del proyecto, dónde se explican sus posibles implicaciones ambientales, económicas, tecnológicas e industriales. También se explican sus contribuciones con los objetivos de desarrollo sostenible. Para finalizar, se incluye un apartado de conclusiones, dónde se termina de realizar una visión general del trabajo y sobre cómo se han cumplido los objetivos del proyecto.

2. Marco tecnológico

2.1 Arquitecturas y patrones de diseño software

Los patrones de arquitectura software son soluciones para el diseño y desarrollo de software que proporcionan un enfoque estructurado y reusable para resolver problemas comunes, abordando aspectos como la organización de componentes, la interacción entre ellos, la distribución de responsabilidades y la escalabilidad del sistema.

Estos patrones permiten definir las características y los comportamientos básicos de una aplicación, siendo crucial conocer sus particularidades para elegir el más adecuado según las necesidades y objetivos empresariales específicos, justificando así, las decisiones arquitectónicas. Los principales patrones de arquitectura software son los siguientes: [1]

(i) Arquitectura de eventos: La arquitectura basada en eventos está formada por componentes de procesamiento de eventos altamente desacoplados y de un solo propósito que reciben y procesan eventos de manera asíncrona. Este patrón de arquitectura consta de dos topologías principales, el mediador y el *broker.*

La topología del mediador se utiliza comúnmente cuando se necesitan orquestar múltiples pasos dentro de un evento a través de un mediador central, mientras que la topología del *broker* se utiliza cuando se desea encadenar eventos sin el uso de un mediador central.

La topología del mediador se utiliza para eventos que tienen múltiples pasos y requieren cierto nivel de orquestación para procesar el evento. Los componentes de esta topología incluyen colas de eventos, un mediador de eventos, canales de eventos y procesadores de eventos. El flujo de eventos comienza con un cliente enviando un evento a una cola de eventos, que luego es transportado al mediador de eventos. El mediador de eventos orquesta el evento inicial enviando eventos de procesamiento adicionales de manera asíncrona a los canales de eventos para ejecutar cada paso del proceso. Los procesadores de eventos reciben los eventos de los canales de eventos y ejecutan la lógica de negocio específica para procesar el evento. En esta arquitectura es importante elegir la implementación adecuada del mediador de eventos que se adapte a las necesidades y requisitos específicos de cada aplicación.

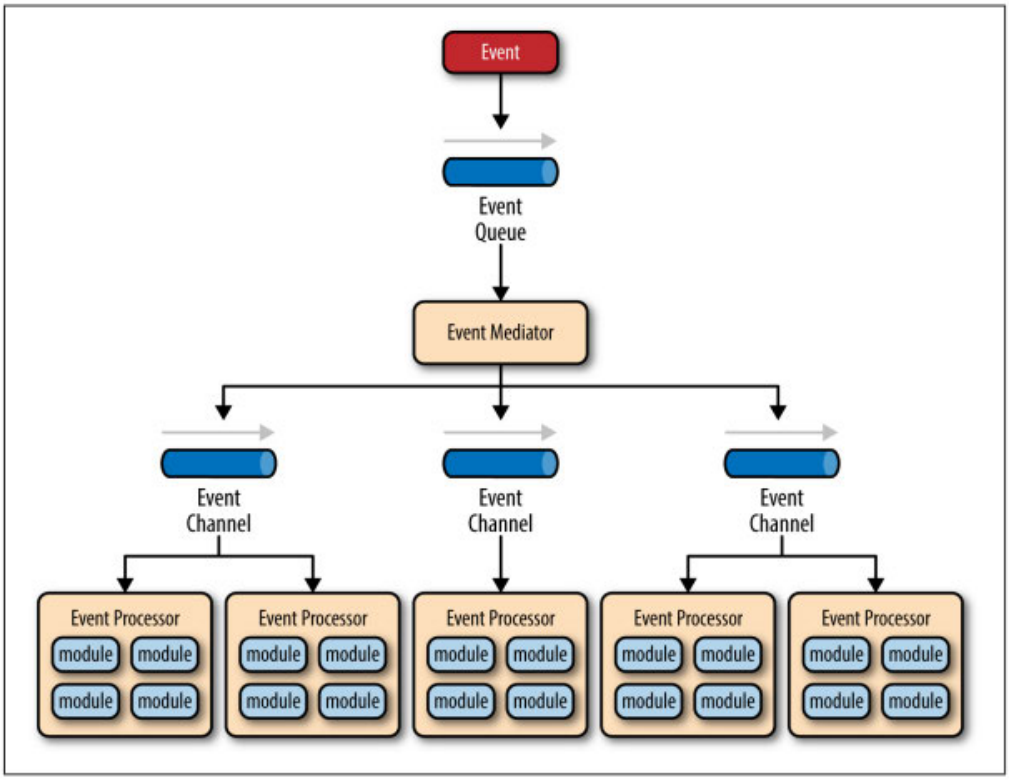


Ilustración 2. Topología de mediador de una arquitectura de eventos [1]

La topología del *broker* difiere de la del mediador en que no hay un mediador central de eventos; en cambio, el flujo de mensajes se distribuye entre los componentes procesadores de eventos de manera encadenada a través de un ligero *broker* de mensajes (por ejemplo, ActiveMQ, HornetQ, etc.). Esta topología es útil cuando se tiene un flujo de procesamiento de eventos relativamente simple y no se desea (o necesita) orquestación central de eventos. Los componentes principales de esta topología son el componente del *broker* y el componente del procesador de eventos, que pueden ser centralizados o federados y contienen todos los canales de eventos utilizados en el flujo de eventos. Los canales de eventos pueden ser colas de mensajes, temas de mensajes o una combinación de ambos.

En esta topología, cada componente procesador de eventos es responsable de procesar un evento y publicar un nuevo evento que indique la acción realizada. Este enfoque se asemeja a una carrera de relevos, donde cada corredor (o componente procesador de eventos) realiza su tarea y luego pasa el "testigo" (o evento) al siguiente corredor (o componente). Una vez que un evento es publicado por un componente, dicho componente, no vuelve a estar involucrado en su procesamiento.

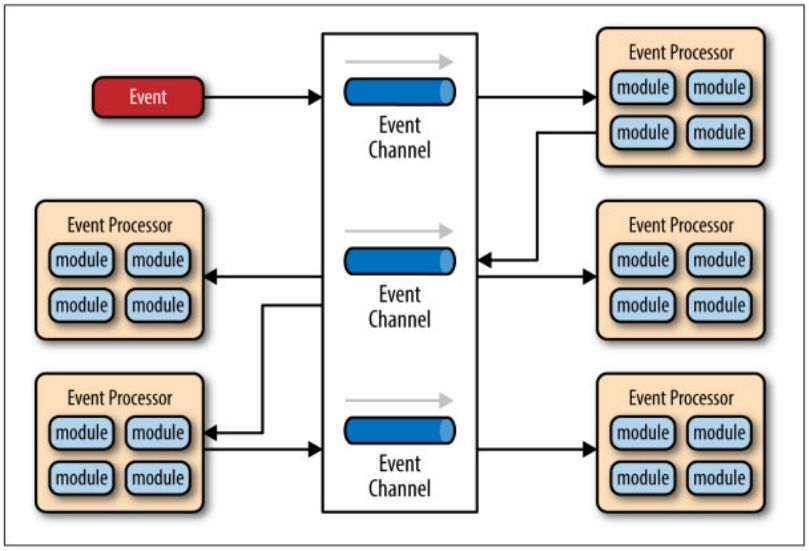


Ilustración 3. Topología de broker de una arquitectura de eventos [1]

(ii) Arquitectura microkernel: consiste en dos tipos de componentes, un sistema central y módulos de complementos. La lógica de la aplicación se divide entre los módulos de complementos independientes y el sistema central básico, lo que proporciona extensibilidad, flexibilidad y aislamiento de las características de la aplicación y la lógica de procesamiento personalizada.

El sistema central contiene solo la funcionalidad mínima necesaria para hacer operativo el sistema, mientras que los módulos de complementos son componentes independientes que contienen procesamiento especializado, características adicionales y código personalizado destinado a mejorar o extender el sistema central para producir capacidades empresariales adicionales. Estos módulos pueden conectarse al sistema central de diversas formas, como a través de OSGi, mensajería, servicios web o incluso vinculación directa punto a punto, dependiendo de las necesidades específicas de la aplicación.

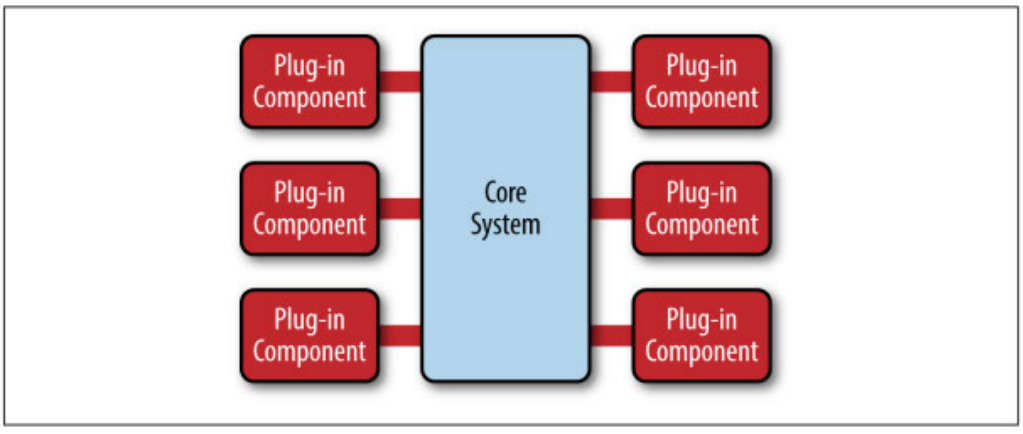


Ilustración 4. Patrón de arquitectura microkernel [1]

(iii) Arquitectura de microservicios: Cada componente de la arquitectura de microservicios se despliega como una unidad separada. Un componente de servicio es un concepto clave que puede variar en granularidad desde un solo módulo hasta una gran parte de la aplicación. Los componentes de servicio contienen uno o más módulos que representan una función de un solo propósito o una porción independiente de una aplicación empresarial más grande.

Además, la arquitectura de microservicios es una arquitectura distribuida, en la que todos los componentes dentro de la arquitectura están completamente desacoplados entre sí y se acceden a través de algún tipo de protocolo de acceso remoto.

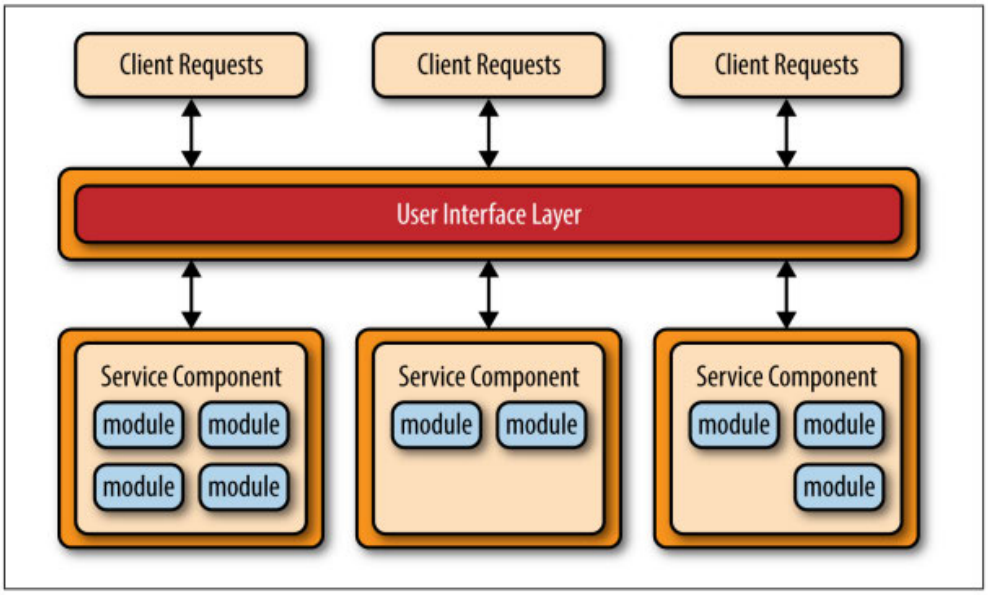


Ilustración 5. Patrón básico de arquitectura de microservicios [1]

(iv) Arquitectura basada en espacio: El patrón basado en espacio (también conocido a veces como patrón de arquitectura en la nube) minimiza los factores que limitan la escalabilidad de las aplicaciones. Este patrón se basa en el concepto de espacio de tuplas lo que constituye la idea de una memoria compartida distribuida.

Se logra una alta escalabilidad al eliminar la restricción de la base de datos central y utilizar en su lugar redes de datos en memoria replicadas. Los datos de la aplicación se mantienen en memoria y se replican entre todas las unidades de procesamiento activas. Las unidades de procesamiento pueden iniciarse y apagarse dinámicamente a medida que aumenta y disminuye la carga de usuario, abordando así la escalabilidad variable.

Dado que no hay una base de datos central, se elimina el cuello de botella de la base de datos, lo que proporciona una escalabilidad casi infinita dentro de la aplicación. Los componentes principales de esta arquitectura son la unidad de procesamiento y el middleware virtualizado, que se encargan de alojar los componentes de la aplicación y de gestionar la sincronización de datos y las comunicaciones, respectivamente.

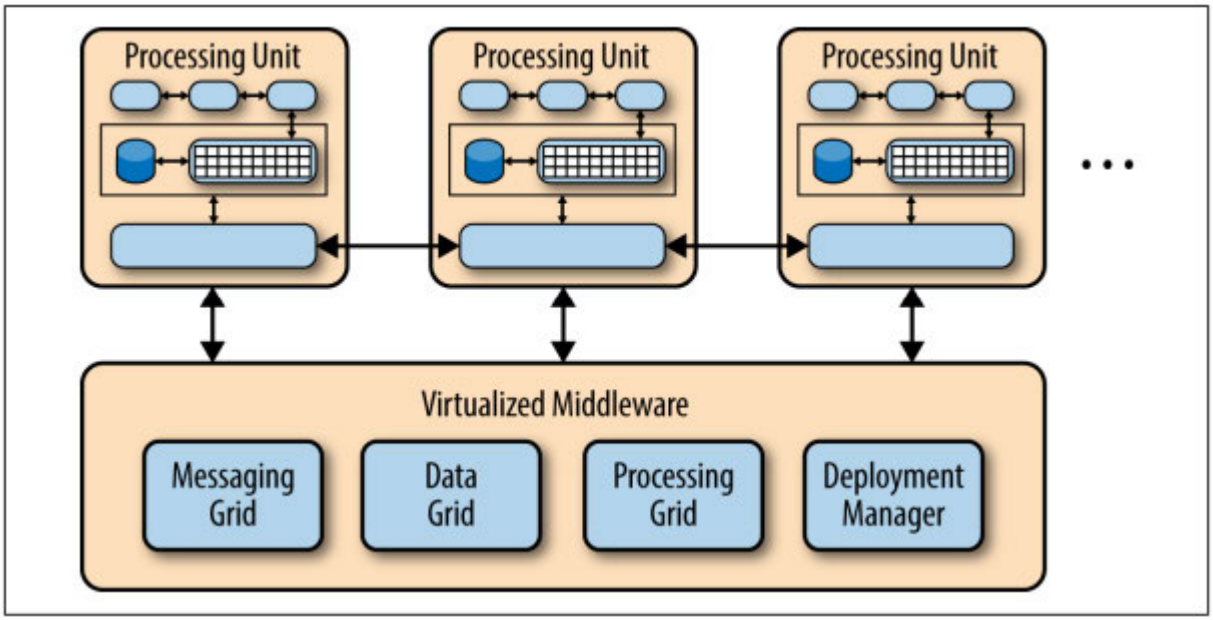


Ilustración 6. Patrón de arquitectura basada en espacio [1]

(v) Arquitectura en capas: El patrón de arquitectura en capas organiza los componentes en capas horizontales, cada una con un rol específico dentro de la aplicación (por ejemplo, lógica de presentación o lógica de negocio). Aunque este patrón no especifica el número y tipo de capas que deben existir, la mayoría de las arquitecturas en capas consisten en cuatro capas estándar: presentación, negocio, persistencia y base de datos. En algunos casos, las capas de negocio y persistencia se combinan en una sola, especialmente cuando la lógica de persistencia (por ejemplo, SQL o HSQL) está incrustada dentro de los componentes de la capa de negocio.

Cada capa tiene un rol específico y una responsabilidad dentro de la aplicación, lo que permite una clara separación de preocupaciones entre los componentes y facilita el desarrollo, prueba y mantenimiento de la aplicación.

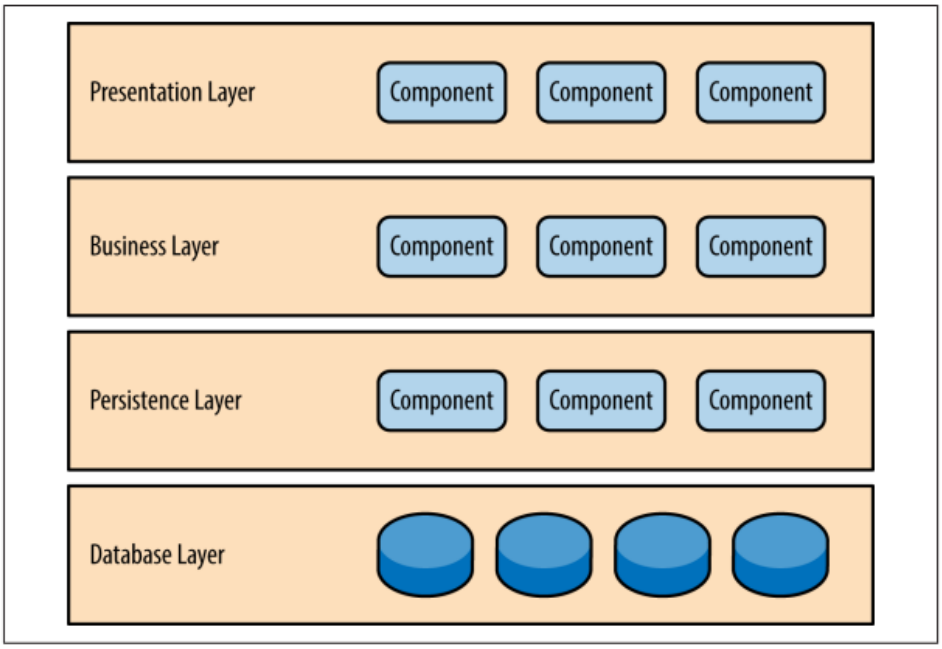


Ilustración 7. Patrón de arquitectura en capas [1]

Dentro de este patrón existen distintas arquitecturas como Modelo-Vista-Controlador, Modelo-Vista-Presentador, Modelo-Vista-VistaModelo, Modelo-Vista-Interactividad y Modelo-Vista-Controlador de Actividad, cada una con sus especificaciones y utilidades. Muchas arquitecturas combinan la capa de negocio con la capa de persistencia.

La más extendida y utilizada es Modelo-Vista-Controlador que separa una aplicación en tres capas principales: el modelo, la vista y el controlador. El modelo representa los datos y la lógica de negocio de la aplicación, la vista es responsable de la presentación de la información al usuario y el controlador actúa como intermediario entre el modelo y la vista, gestionando las solicitudes del usuario y actualizando el modelo en consecuencia. Esta separación de responsabilidades facilita la modificación y el mantenimiento del código, promueve la reutilización y facilita la colaboración entre equipos de desarrollo. [2]

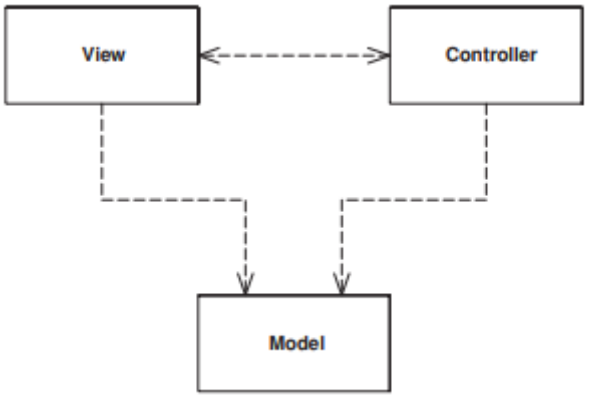


Ilustración 8. Roles en la arquitectura MVC [2]

2.2 APIs web

Las APIs (Application Programming Interfaces) representan una parte fundamental en el desarrollo de software moderno, proporcionando los medios para que diferentes aplicaciones y servicios se comuniquen y colaboren entre sí. Las APIs definen un conjunto de reglas y protocolos para la construcción y la interacción con aplicaciones de software, permitiendo integrar fácilmente funcionalidades externas y exponer funcionalidades propias a otros sistemas. [3]

Una API es una interfaz que permite la interacción entre diferentes sistemas de software. Está compuesta por varias partes clave, como los *endpoints*, que son las URL donde las APIs reciben las solicitudes. Cada *endpoint* realiza una función específica y responde a ciertos métodos HTTP, como GET, POST, PUT, DELETE. Los métodos HTTP definen la acción que se realizará en el servidor, como recuperar datos con GET, enviar datos con POST, actualizar datos con PUT o eliminar datos con DELETE. Las APIs comúnmente utilizan JSON (JavaScript Object Notation) o XML (eXtensible Markup Language) para estructurar los datos que se envían y reciben. Además, para proteger el acceso a los datos, las APIs implementan mecanismos de seguridad como OAuth, API *keys* y tokens JWT (JSON Web Tokens).

El uso de APIs ofrece múltiples beneficios en el desarrollo de software, permitiendo que diferentes aplicaciones y servicios, posiblemente construidos con distintas tecnologías, se comuniquen entre sí, mejorando la interoperabilidad. Además, facilitan la reutilización de componentes, ya que las funcionalidades existentes pueden ser reutilizadas en nuevas aplicaciones, reduciendo el tiempo de desarrollo y los costos. Las APIs también facilitan el escalamiento de aplicaciones distribuidas, permitiendo que los servicios se ejecuten en distintos servidores o incluso en diferentes localizaciones geográficas. Fomentan un diseño modular donde diferentes componentes de una aplicación pueden ser desarrollados, desplegados y mantenidos de manera independiente, haciendo más fácil la integración con servicios de terceros y ampliando las capacidades de una aplicación sin necesidad de desarrollarlas desde cero. También unifican el coste computacional de operaciones en una única plataforma permitiendo que todas las operaciones se realicen en un único servidor.

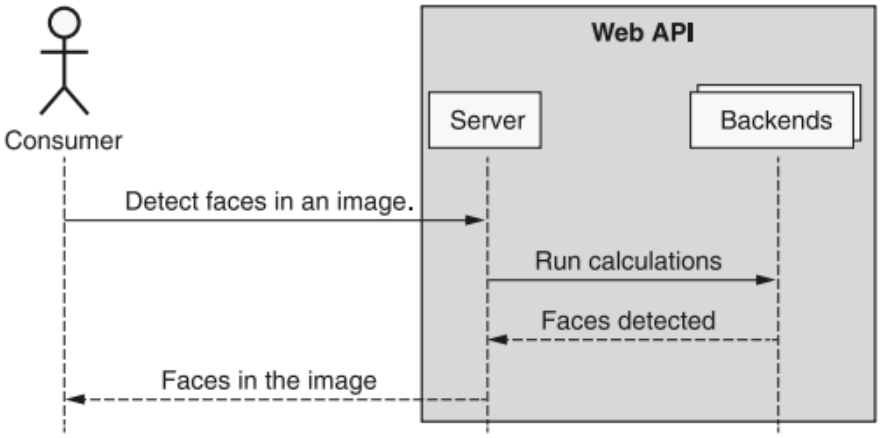


Ilustración 9. Ejemplo de Web API escondiendo costes computacionales [3]

2.3 Desarrollo web en Python

Python es un lenguaje de programación con una variedad de herramientas y *frameworks* que facilitan la construcción de aplicaciones web eficientes y escalables. Es ampliamente utilizado en el desarrollo web debido a su versatilidad y facilidad de uso.

2.3.1 Integridad, consistencia del entorno y envoltorios virtuales

En Python es muy característico el uso de múltiples dependencias, librerías y herramientas. Los envoltorios virtuales permiten mantener la integridad y consistencia del entorno de desarrollo al aislar las dependencias de cada proyecto. Esto evita conflictos entre versiones de paquetes y garantiza que cada proyecto pueda funcionar de manera independiente sin afectar a otros.

Los envoltorios virtuales facilitan la reproducibilidad del entorno de desarrollo. Al especificar las dependencias exactas de un proyecto en un archivo de requisitos, es posible recrear el mismo entorno en cualquier máquina, lo que hace que sea más fácil colaborar en proyectos y distribuir aplicaciones de manera consistente.

Adicionalmente, los envoltorios virtuales pueden mejorar la portabilidad de las aplicaciones, ya que el entorno de desarrollo puede ser empaquetado junto con la aplicación para garantizar que se ejecute de manera consistente en diferentes sistemas. El uso de envoltorios virtuales y de la especificación del fichero de requisitos es fundamental para el despliegue en la nube de cualquier aplicación web.

La definición del fichero de requisitos no se suele realizar de manera manual. La forma tradicional de obtener el fichero *requirements.txt* de requisitos se realiza utilizando el comando *pip freeze*. El problema de este método es que obtendrá todas las dependencias del envoltorio virtual, aunque no se haga uso de esas dependencias en ningún lugar del código.

Para solucionarlo existe una herramienta llamada *pipreqs* que analizará el proyecto y especificará en el fichero de requisitos solo las dependencias que se estén utilizando de forma activa en el código del proyecto.

2.3.2 Frameworks de desarrollo web en Python

Los *frameworks* de desarrollo web ofrecen una estructura y un conjunto de herramientas predefinidas que proporcionan funciones comunes y características integradas como la gestión de rutas, la interacción con bases de datos y la autenticación de usuarios.

2.3.2.1 Flask

Flask es un *framework* que destaca por su simplicidad y su capacidad para proporcionar un núcleo sólido con servicios básicos y a su vez permitiendo la extensibilidad a través de una amplia gama de extensiones. Su diseño modular permite seleccionar y combinar las herramientas que mejor se adapten a las necesidades de su proyecto, evitando así la inclusión de funcionalidades innecesarias. [4]

Flask es utilizado para crear aplicaciones web sencillas, pero ofrece una gran personalización a la hora de elegir qué funcionalidades necesitamos en la aplicación web.

2.3.2.2 Django

Django ofrece una amplia gama de herramientas y funcionalidades listas para usar, incluyendo autenticación de usuarios, manejo de rutas y vistas, una interfaz de administración, seguridad sólida y soporte para múltiples motores de base de datos, entre otros. Su enfoque facilita el desarrollo rápido y eficiente de aplicaciones web, permitiendo centrarse en la lógica de la aplicación en lugar de preocuparse por la infraestructura básica. [5]

2.3.3 Object Relational Mappers (ORM)

Los ORM, o mapeadores objeto-relacional son interfaces que simplifican la interacción entre los modelos de datos de las aplicaciones y las bases de datos. Estas herramientas permiten interactuar con la base de datos utilizando objetos y clases en lugar de consultas SQL directas, lo que simplifica la manipulación de datos y reduce la complejidad del código. Garantizan la integridad y consistencia de la base de datos mediante la gestión automática de transacciones.

2.3.3.1 SQLAlchemy

SQLAlchemy es una herramienta de mapeo objeto-relacional (ORM) para Python. Su ventaja principal es que permite definir modelos de datos utilizando clases de Python que representan tablas en la base de datos. Estos modelos definen la estructura de los datos y las relaciones entre las tablas, lo que facilita la descripción de los datos. También proporciona una interfaz unificada para interactuar con varios sistemas de gestión de bases de datos relacionales, incluyendo PostgreSQL, MySQL, SQLite y Oracle. Esto permite escribir código independiente de la base de datos subyacente ya que el modelo se define en la aplicación y no hay que adaptar un modelo de datos existente en la aplicación al modelado de la base de datos.

SQLAlchemy consta de dos partes principales: el núcleo y el ORM. El núcleo ofrece funcionalidades de integración y descripción de SQL y bases de datos, destacando la SQL *Expression Language*, una herramienta independiente del ORM que permite construir y ejecutar expresiones SQL contra una base de datos.

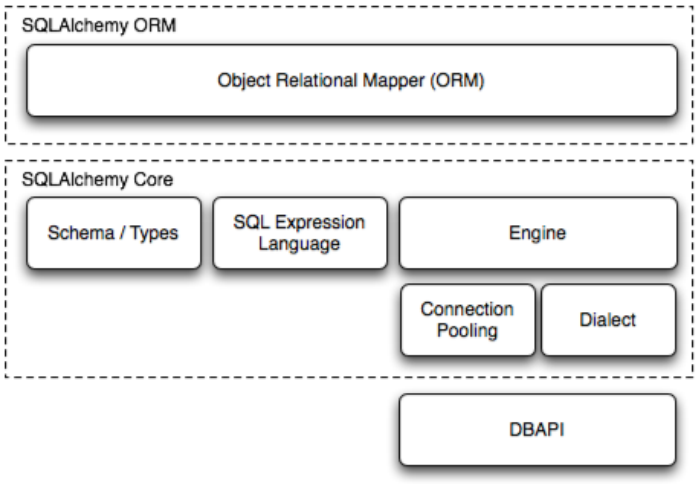


Ilustración 10. Arquitectura de SQLAlchemy [6]

Por otro lado, el ORM, proporciona un medio para trabajar con un modelo de objetos de dominio, mapeado a un esquema de base de datos. Mientras que el Core y la SQL *Expression Language* están centrados en el esquema de la base de

datos y orientados a comandos, el ORM ofrece la posibilidad de la definición de los datos como clases de Python por medio de una instancia de clase principal que agrega una vista con un enfoque orientado a objetos y estados. Esto refleja la diferencia entre trabajar con la estructura de la base de datos y trabajar con los objetos del dominio en el contexto de bases de datos relacionales. [6]

2.3.3.2 Flask-SQLAlchemy

Flask-SQLAlchemy es una extensión de Flask que proporciona integración con SQLAlchemy simplificando la interacción con bases de datos relacionales en aplicaciones web Flask. Permite definir modelos de datos utilizando clases de Python que representan tablas en la base de datos y atributos de clase que especifican sus relaciones. Estos modelos se definen como subclases de la clase '*db.Model*' proporcionada por Flask-SQLAlchemy, lo que facilita la descripción y la manipulación de datos en la aplicación.

En resumen, Flask-SQLAlchemy proporciona una capa de abstracción para interactuar con bases de datos relacionales en aplicaciones web Flask. [7]

2.3.3.3 Librerías de estructuración de datos (Pandas)

Pandas es una biblioteca de código abierto de Python que proporciona estructuras de datos y herramientas de análisis. Pandas ofrece dos estructuras de datos principales: Series y *DataFrame*s. Estas estructuras permiten almacenar y manipular datos de forma tabular, similar a una hoja de cálculo, lo que facilita el análisis y la manipulación de datos.

La capacidad de crear este tipo de estructuras de datos permite unificar formatos como CSV y utilizar *DataFrames* para realizar todo tipo de consultas y tareas de procesamiento de información de forma eficiente.

Además, ofrece una amplia gama de funciones y métodos para el tratamiento de datos, incluyendo filtrado, selección, agrupación, ordenación y agregación de datos. Esto permite realizar operaciones complejas de forma eficiente y con poco código. Finalmente, Pandas permite manejar conjuntos de datos de gran tamaño, utilizando técnicas de optimización y operaciones vectorizadas. [8]

2.4 Hospedaje en la nube

Los servicios de hospedaje en la nube son plataformas que ofrecen recursos informáticos, como servidores, almacenamiento, bases de datos y redes, a través de internet. Estos recursos se encuentran alojados en centros de datos distribuidos geográficamente y son gestionados por proveedores de servicios en la nube. Los usuarios pueden acceder a estos recursos bajo demanda y pagar únicamente por el uso que hagan de ellos, lo que proporciona flexibilidad, escalabilidad y eficiencia en costos. Los servicios de hospedaje en la nube pueden clasificarse en tres categorías principales: Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS), según el nivel de abstracción y gestión que proporcionen al usuario final. [9]

(i) Infraestructura como Servicio (IaaS): Proporciona recursos de infraestructura virtualizada, como servidores virtuales, redes y almacenamiento, permitiendo a los usuarios implementar y gestionar sus propios sistemas operativos y aplicaciones.

(ii) Plataforma como Servicio (PaaS): Ofrece un entorno de desarrollo y ejecución completo para aplicaciones, incluyendo herramientas de desarrollo, sistemas operativos, middleware y servicios de base de datos, eliminando la necesidad de gestionar la infraestructura subyacente.

(iii) Software como Servicio (SaaS): Proporciona aplicaciones completamente desarrolladas y listas para usar a través de la nube, eliminando la necesidad de instalación y mantenimiento local.

2.4.1 Plataformas de hospedaje en la nube

Las principales plataformas de hospedaje en la nube son: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud y Oracle Cloud. Cada plataforma puede ofrecer una selección ligeramente diferente de servicios en la nube, en áreas específicas como inteligencia artificial, análisis de datos o IoT.

Los precios de los servicios en la nube pueden variar entre las diferentes plataformas, y cada una puede tener modelos de precios diferentes, como precios por consumo, precios por uso de recursos o precios por suscripción.

Las plataformas de hospedaje en la nube pueden integrarse con otros productos y servicios de la misma empresa o de terceros, lo que puede influir en la elección de una plataforma sobre otra dependiendo de las necesidades del usuario. La calidad del soporte técnico y la comunidad de usuarios también puede variar entre las diferentes plataformas de hospedaje en la nube, lo que puede influir en la experiencia del usuario y en la resolución de problemas.

3. Especificaciones y restricciones de diseño

Una empresa ficticia llamada Express Delivery dedicada al sector de la mensajería necesita un sistema para la gestión de envíos y así, poder empezar a prestar servicio. La empresa necesita gestionar un sistema de correos y mensajeros, que sea capaz de agregar correos rápidamente y mantener actualizado el estado de cada una de las entregas.

La empresa quiere empezar a prestar servicio en 2025 y necesita elegir una solución para su sistema de gestión de información empresarial, por tanto, ha pedido a distintos socios tecnológicos que le propongan una solución y le presenten un prototipo.

El sistema será utilizado por la empresa y sus mensajeros para gestionar y organizar la entrega de paquetes físicos a domicilio. La empresa opera en Madrid y tiene oficinas físicas en varios municipios de la provincia con mensajeros asignados a cada una de las oficinas.

La empresa no cuenta no con infraestructura tecnológica por tanto la solución propuesta deberá contar con el despliegue de algún tipo de infraestructura para el correcto funcionamiento de los componentes de la solución.

Requisitos funcionales:

- La aplicación almacenará datos a gran escala en un servidor de base de datos
- La base de datos debe contener información geográfica real, como información sobre códigos postales, municipios y ciudades.
- Como administrador del sistema se deben poder realizar acciones de adición, consulta, actualización y eliminación de cada uno de los tres tipos principales de entidades (correo, mensajero, entrega).
- Como mensajero debe ser posible consultar las entregas asignadas y la actualización de información como servicio contratado para la entrega, fecha de entrega y estado de entrega.
- La aplicación debe tener un sistema de autenticación para garantizar la seguridad de los datos.

- La aplicación será utilizada por dos agentes de usuario: un usuario de gestión que registre los mensajeros, añada los correos a la base de datos y les asigne un mensajero para convertirlos en entregas y un usuario mensajero que consulte y actualice los estados de las entregas.

Requisitos no funcionales:

- La interfaz de usuario debe ser intuitiva y fácil de usar, con una navegación clara y eficiente.
- La aplicación debe ser capaz de estar correctamente modularizada para garantizar que se pueda actualizar según la necesidad.
- La aplicación debe estar disponible las 24 horas del día, los 7 días de la semana, con un tiempo de inactividad mínimo para mantenimiento programado.

4. Descripción de la solución propuesta

4.1. Descripción general de la solución

La solución propuesta consiste en una aplicación web construida utilizando la arquitectura Modelo Vista Controlador desde la que se pueda gestionar toda la información detallada en la especificación. Por tanto, los componentes que incluirá la aplicación son un servidor de base de datos, en el que se almacenará la información acorde al modelo de los datos de la aplicación y un servidor web que contendrá la aplicación web de gestión.

La arquitectura MVC permite modularizar y separar por componentes la aplicación lo que facilita el desarrollo y permite probar individualmente cada una de las partes de la aplicación además de aislar los problemas.

Se ha optado por una aplicación web hospedada en la nube frente a, por ejemplo, una aplicación local hospedada en la red de la empresa debido a la gran cantidad de ventajas y facilidades que nos proporciona el desarrollo web y los servicios de hospedaje *Cloud* en este tipo de aplicación. A continuación, se justificará la elección para elegir este tipo de aplicación.

4.1.1. Aplicación web vs aplicación local

Para la solución se optará por desarrollar una aplicación web debido a las ventajas que supone frente a una aplicación local dedicada. El desarrollo web es un campo de la programación muy extendido y hoy en día se cuenta con una inmensa cantidad de documentación y facilidades que no se encuentran en el desarrollo de aplicaciones locales.

Las principales ventajas por las que se elige esta solución frente a las aplicaciones locales son:

- Accesibilidad multiplataforma: Las aplicaciones web pueden ser accesibles desde cualquier dispositivo con un navegador web y una conexión a internet. Esto es importante debido a que no conocemos el despliegue de sistemas de la empresa y a la accesibilidad desde los dispositivos móviles de los mensajeros. Con una aplicación web no tenemos que desarrollar una aplicación distinta para cada uno de los dispositivos o sistemas operativos, con tener una página web con diseño responsivo es suficiente.

- Despliegue: Las aplicaciones web se ejecutan directamente en el navegador web del usuario, lo que significa que no es necesario descargar ni instalar ningún software adicional en el dispositivo. Esto reduce la fricción para los clientes y reduce costes e incidencias.

- Actualizaciones: Las aplicaciones web se ejecutan en servidores remotos, por lo que las actualizaciones y mejoras pueden implementarse de forma centralizada y automática sin necesidad de que los usuarios realicen ninguna acción. Esto garantiza que todos los usuarios tengan acceso a la versión más reciente de la aplicación en todo momento.

- Costos de desarrollo y mantenimiento reducidos: El desarrollo de una aplicación web y su mantenimiento es más económico que el desarrollo de una aplicación local dedicada debido al amplio conocimiento y las herramientas que se disponen en el campo del desarrollo web.

- Escalabilidad y flexibilidad: Las aplicaciones web pueden escalar fácilmente para manejar un mayor volumen de usuarios o datos sin necesidad de realizar cambios significativos en la infraestructura. Además, pueden integrarse con otras aplicaciones y servicios web de manera sencilla, lo que aumenta su flexibilidad y capacidad de adaptación a las necesidades cambiantes del negocio o del usuario. Esta característica de las aplicaciones web garantiza la posibilidad de introducir una gran cantidad de mejoras o solucionar problemas de una forma mucho más cómoda que con una aplicación dedicada.

4.1.2. Hospedaje en Cloud vs hospedaje local

El despliegue de la aplicación se realizará en un servicio Cloud debido a las ventajas que supone frente a hospedarla en un servidor web dentro de la red de la empresa.

La disponibilidad es uno de los recursos no funcionales de la especificación de diseño y mediante las plataformas Cloud se puede conseguir un alto nivel de disponibilidad por un coste más reducido que en un hospedaje local. Las plataformas en la nube ofrecen una alta disponibilidad y redundancia en sus centros de datos, lo que garantiza que la aplicación esté disponible para los usuarios en casi todo momento. De la misma manera, muchas plataformas en la nube ofrecen acuerdos de nivel de servicio (SLA) que garantizan un tiempo de actividad mínimo, lo que reduce el riesgo de tiempo de inactividad no planificado.

Los servicios de plataforma en la nube ofrecen la capacidad de escalar fácilmente la infraestructura de hospedaje para satisfacer las demandas cambiantes del tráfico web. Esto significa que la aplicación puede crecer y adaptarse a medida que la empresa expande su base de usuarios sin preocuparse por la gestión de servidores locales y la capacidad de hardware. Existen dos tipos de escalados.

(i) Escalado vertical: se centra en las características relacionadas con el hardware virtual de los servicios virtuales contratados, si una aplicación necesita más potencia de procesamiento o memoria se pueden mejorar las características de la máquina virtual, su CPU o su memoria RAM, por el contrario, si se han sobre especificado las necesidades, sería posible reducir las especificaciones de la máquina.

(ii) Escalado horizontal: se centra en aumentar el número de recursos contratados, por ejemplo, aumentando el número de contenedores que contuviesen el servidor de la aplicación web.

Debido al diseño de la arquitectura de la nube, los recursos se pueden implementar en regiones de todo el mundo. De esa forma en caso de que ocurriese algún problema en alguna región, otras seguirían en funcionamiento permitiendo a la aplicación continuar sin problemas debido a la redundancia en zona que es capaz de proveer un servicio de Cloud *"La confiabilidad es la capacidad de un sistema de recuperarse de los errores y seguir funcionando. "* [10]

Hospedar una aplicación web en la nube puede ser más rentable que mantener servidores locales. Los servicios en la nube suelen ofrecer modelos de precios flexibles como el pago por uso o la facturación mensual, lo que permite a las empresas pagar solo por los recursos que utilizan. Además, elimina la necesidad de invertir en hardware costoso y mantenimiento de infraestructura. Las plataformas de Cloud incluyen sistemas de métricas y previsibilidad que hacen que una empresa sea capaz de analizar y prever los costes de hospedaje de sus servicios. También incluyen métricas de rendimiento que hacen posible el análisis y la previsión para un posible escalado.

Los proveedores de servicios en la nube ofrecen medidas de seguridad avanzadas para proteger los datos y las aplicaciones hospedadas en sus plataformas. Esto incluye cifrado de datos, cortafuegos y detección de intrusiones. Hoy en día, además, existe la necesidad de acomodar los modelos de seguridad a gobernanzas y políticas de seguridad exigidas por leyes o por la administración de las propias empresas. Los servicios Cloud incluyen plantillas que garantizan que todos los recursos implementados cumplan estándares corporativos y requisitos normativos del gobierno. Esto confiere la posibilidad de actualizar esos modelos de seguridad a medida que cambien y de realizar auditorías basadas en la nube.

A pesar de todo, esta característica de los servicios de hospedaje en la nube significa que se deja en manos de otra entidad todo lo relacionado con la seguridad de los datos empresariales. La seguridad de los datos o de los servicios en una empresa siempre va a ser mayor y va a permitir un mayor nivel acomodación a las soluciones de seguridad que se necesitan si se realiza de forma local.

Con la infraestructura en la nube, la gestión y el mantenimiento de servidores se externalizan al proveedor de servicios en la nube. Esto libera a la empresa de la carga de administrar hardware. Los servicios de Cloud incluyen muchas facilidades para la gestión del entorno y los recursos en la nube y facilitan mucho el despliegue de cualquier servicio.

4.2. Fases del desarrollo del proyecto

Una vez justificado todo lo relacionado con la decisión de una aplicación web hospedada en la nube frente a cualquier otra solución para las especificaciones de diseño se proceden a describir las fases de desarrollo de la aplicación:

- Fase de investigación y elección de herramientas: en la que se investigarán las herramientas que se utilizarán para el diseño de la aplicación y los posibles servicios que se contratarán relacionados con el hospedaje en la nube.
- Fase de modelado: Se analizarán la especificación de diseño y se definirán las clases y objetos necesarios para el diseño de la aplicación web.

- Fase de desarrollo: Se desarrollará un prototipo que trabaje de forma local para probar el correcto funcionamiento de la lógica de la aplicación.
- Fase de migración: Se realizará la migración de los servicios a la nube y se probará el correcto funcionamiento.

A continuación, se hará uso de las fases de desarrollo para organizar la estructura de la solución propuesta.

4.3. Fase de investigación y elección de herramientas

En esta fase se investigarán las soluciones tecnológicas y las herramientas que utilizaremos dentro del marco de las aplicaciones web hospedadas en la nube, ya que es la solución por la que hemos optado.

4.3.1. Arquitectura software

A pesar de que, dada la solución propuesta, el patrón de arquitectura software más conveniente para la aplicación sería un patrón de arquitectura basado en la nube, en el que se evitaría el cuello de botella a una única base de datos mediante módulos de procesamiento virtuales que se pueden escalar y mediante la replicación de datos en la parte del cliente, para el prototipo de la aplicación y para este trabajo de fin de grado se optará por una arquitectura software basada en capas siguiendo el patrón Modelo-Vista-Controlador.

- Esta decisión se debe a que a nivel funcional ambas arquitecturas funcionan igual a pequeña escala y no se llegará a un punto de embotellamiento de la base de datos hasta que no haya un gran número de usuarios concurrentes o un volumen de datos muy alto. Para nuestro caso de uso solo se especifica que es una empresa que trabaja con agentes de gestión de datos y mensajeros, lo que significa que no tienen por qué trabajar de forma simultánea.

También al haber optado por una aplicación hospedada en la nube, se podría escalar la potencia del servidor de la base de datos o del servidor web en caso de darse embotellamiento o bajo rendimiento.

4.3.1.1. Descripción de la arquitectura

Como bien hemos justificado, se ha definido una arquitectura en tres capas, a continuación, se muestra el diagrama de componentes de la aplicación.

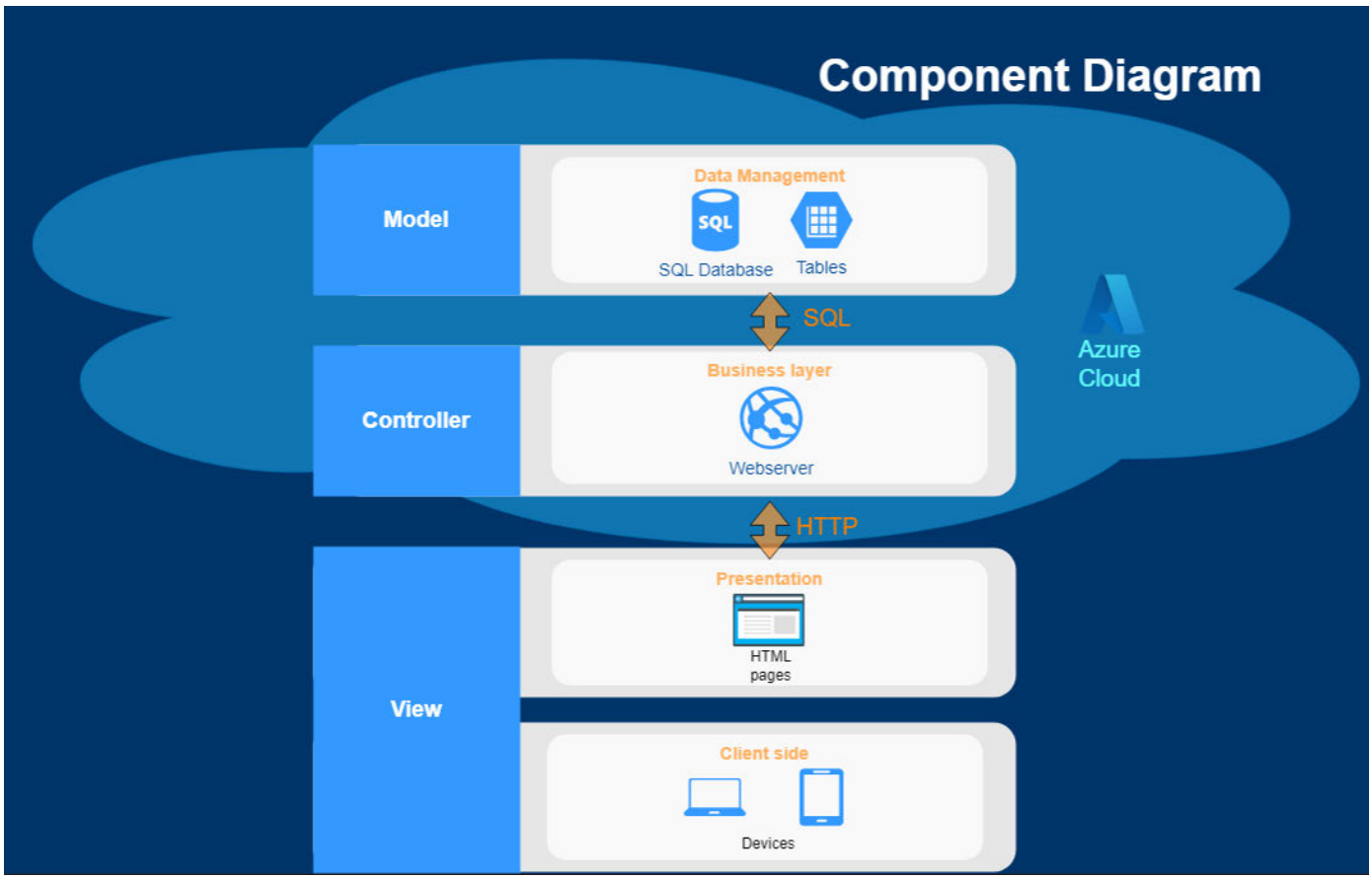


Ilustración 11. Diagrama de componentes de la aplicación.

- Vista: La capa de vista se encarga de la representación visual para el usuario. En esta capa de la aplicación incluimos los dispositivos que contienen las páginas HTML en las que se ve representada la información regresada por el servidor web. A pesar de que las plantillas de las páginas HTML son gestionadas por el servidor web, podemos considerar el diseño de estas plantillas como parte de la capa de vista. Además, su representación es distinta dependiendo de la información que se consulte.
  La capa de vista no contiene ningún componente dentro de la arquitectura al tratarse de la capa que gestiona la interfaz de usuario. Toda la presentación de la aplicación web y la interfaz se incluye en el componente del servidor web.

- Controlador: Esta capa contiene toda la lógica de negocio, desde el diseño de la interfaz API Rest que responde a las peticiones HTTP del cliente hasta la definición de las clases de Python y su adaptación al modelo SQL por medio del ORM Flask-SQLAlchemy. De la misma forma, incluye la lógica del servidor web.
  Dentro de la capa de controlador incluimos el servidor web con toda la lógica de negocio.

- Modelo: La capa de modelo incluye la definición de los datos para la aplicación. Aunque con SQLAlchemy se define el modelo directamente desde Python y las tablas se crean desde cero en el servidor de base de datos, seguiremos considerando la base de datos como parte de la capa de modelo.

La capa de modelo contiene el servidor de base de datos que a su vez contiene las tablas que hacen referencia a las clases que utilizará la aplicación.

Tanto la capa de Modelo como la de Controlador están incluidas dentro de la nube en esta arquitectura ya que todos sus componentes se encuentran contenidos en la plataforma Cloud. El servidor web se hospeda en un container y la base de datos con sus tablas en un servidor de base de datos virtual.

4.3.2. Lenguaje de programación

La elección del lenguaje de programación se ha justificado principalmente como elección personal del estudiante. Dado lo estudiado en la asignatura del grado Programación Multiparadigma para las Tic, el estudiante eligió Python como lenguaje de programación para poder extender su conocimiento en el diseño de aplicaciones web con Python.

Otros lenguajes de programación aptos y ampliamente utilizados para el desarrollo web son Java con *frameworks* como Spring o Hibernate y Ruby con Ruby on Rails.

Python también ofrece características que justifican su elección frente a otros lenguajes:

(i) Ecosistema y reutilización: cuenta con una gran cantidad de librerías documentadas y *frameworks* que ofrecen un gran abanico de posibilidades a nivel de elegir qué utilidades se necesitan para nuestra aplicación.

(ii) Sintaxis y simplicidad: es considerado uno de los lenguajes con la sintaxis más sencilla que se puede encontrar, obligando al usuario el uso de tabulaciones y facilitando la declaración de variables sin tener que establecer un tipo de dato, pero a su vez ofreciendo protección de formato transparente para el usuario.

(iii) Compilación: es un lenguaje de programación de alto nivel interpretado, lo que significa que no necesita compilación de código antes de ejecutarse. Una aplicación de Python se puede codificar en su totalidad en un bloc de notas y ejecutarse sin necesidad de compilación.

4.3.3. Framework de desarrollo web

Los principales *frameworks* para el desarrollo web en Python son Flask y Django. Ambos *frameworks* son ampliamente utilizados y disponen de documentación detallada para su uso.

La elección de un *framework* se basa en las necesidades específicas del proyecto y en las preferencias del desarrollador. Flask es un *framework* sencillo que permite construir una aplicación web básica con muy pocas líneas de código. Debido a la poca experiencia del estudiante en el entorno del desarrollo web, Flask presenta una solución más sencilla que Django y permite iniciarse en el mundo del desarrollo web con menos complicaciones. [11]

Se trata de un *framework* minimalista que permite acomodar la mayoría de las necesidades del proyecto de forma incremental. Debido a la naturaleza didáctica del proyecto de fin de grado, se ha considerado preferible optar por un *framework* más sencillo y escalable que facilite su comprensión.

Tras consultar bibliografía y recomendaciones de uso de otros desarrolladores web, se ha llegado a la conclusión de que Flask se acomoda mejor al modelo de aplicación pequeña con soluciones del tipo API REST. Por tanto, se ha optado por Flask al tratarse del *framework* más adecuado para la solución de esta aplicación web.

4.3.4. Conexión con la base de datos y solución de la persistencia

SQLAlchemy es un *Toolkit* y ORM que permite la relación entre clases de Python y las tablas de una base de datos SQL. Funciona por medio de una instancia del tipo SQLAlchemy, una vez instanciada realizamos todas las operaciones de interacción con la base de datos desde ahí.

Para utilizar SQLAlchemy junto a Flask utilizaremos la extensión Flask-SQLALchemy que permite definir modelos de base de datos directamente desde Python. Utilizaremos la misma instancia de SQLAlchemy para definir todos los modelos y sus atributos. Dentro de la clase SQLAlchemy existe el método Column que se utiliza para definir una columna y sus argumentos (el tipo de dato, si es único, si es clave primaria, si puede ser nulo, entre otros). [6]

SQLAlchemy también ofrece una solución de persistencia en la que se pueden realizar cambios sobre los datos mediante una sesión y utilizando métodos como *session.add* para añadir nuevos datos, *session.delete* para eliminarlos y cambiando los valores de los atributos directamente desde la instancia del objeto. Una vez realizados los cambios se utilizaría el método *session.commit* para formalizar los cambios en la base de datos. En caso de haber algún problema como claves duplicadas o datos incorrectos podemos realizar un *session.rollback* para deshacer los cambios realizados en la sesión.

4.3.5. Plataforma Cloud

Existen múltiples plataformas que prestan servicios en la nube, entre ellas Google Engine, Microsoft Azure y Amazon Web Services. Las diferencias que presentan son la integración con otros servicios, por ejemplo, Microsoft Azure incluye integración con otros servicios de Microsoft y ofrece servicios de Software como plataforma con los productos de Microsoft. Por otra parte, una de sus principales diferenciaciones en lo que respecta el ámbito económico, son los precios de los servicios prestados lo que supone un condicionante importante a la hora de la elección.

Para esta solución se ha elegido Microsoft Azure por dos motivos principales. El primero es la experiencia del estudiante en Microsoft Azure, habiendo adquirido tres certificaciones de Microsoft y habiendo trabajado en la plataforma Azure en la asignatura del grado Introduction To Artificial Intelligence In The Cloud. El segundo es el acuerdo de Microsoft con la Universidad Politécnica de Madrid, que proporciona a los estudiantes 100$ de crédito para realizar pruebas en el entorno de

Azure. En cuanto a servicios prestados Azure incluye todo lo necesario para el desarrollo de la solución.

4.4. Fase de modelado

La fase de modelado como su propio nombre indica consistirá en el desarrollo del modelo de datos de la aplicación. Se elaborará el diagrama de clases y se especificarán los atributos de cada uno de los objetos que componen el modelo de datos de la aplicación.

4.4.1. Base de datos

Como modelo de base de datos se trabajará una base de datos relacional que comenzará siendo un servidor de base de datos MySQL y con la migración a la nube pasará a ser un servidor Azure SQL.

Al tener una estructura de base de datos clara, organizada mediante tablas, filas, columnas y, además, queriendo garantizar la integridad de los datos por medio de claves primarias y foráneas, se ha optado por una base de datos relacional frente a otros modelos de base de datos.

4.4.2. Modelo de datos

En la fase de modelado se ha diseñado un modelo de datos que a continuación se justificará:

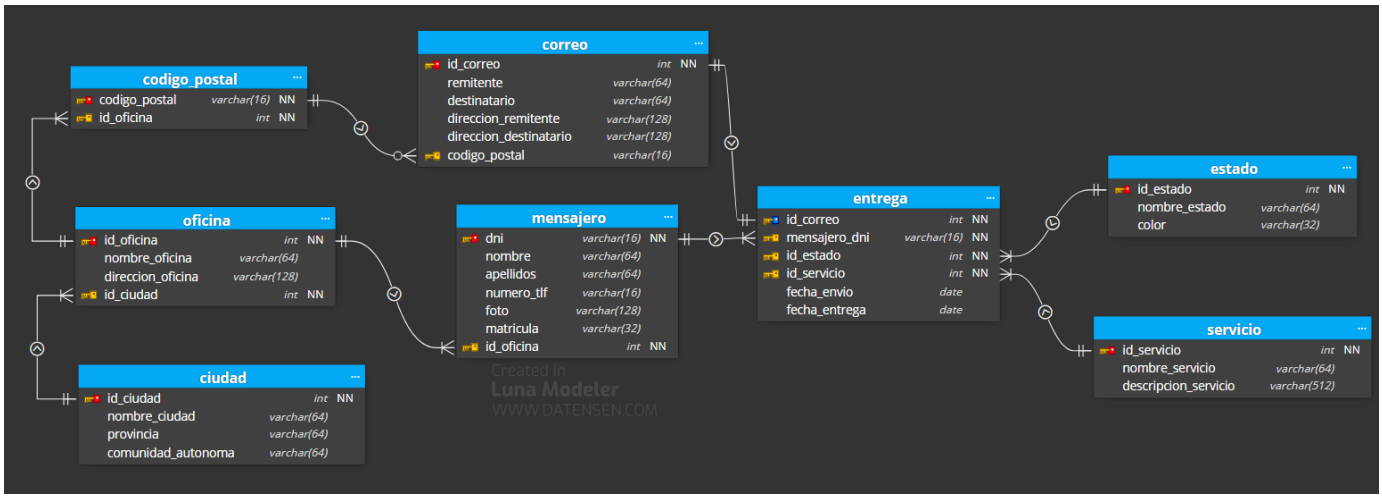


Ilustración 12. Modelo de datos de la aplicación.

El modelo de la base de datos se ha especificado y después se ha definido directamente en Python usando Flask-SQLAlchemy. En este diagrama se ha utilizado la herramienta Luna Modeler de datensen para obtener el diagrama Entidad-Relación directamente desde la base de datos final. Para obtenerlo, el software, lanza un script que realiza peticiones SQL sobre la base de datos para obtener el diagrama por ingeniería inversa. [12]

Ilustración 13. Especificación de la clase ciudad.

La clase ciudad utiliza un identificador como clave primaria dado que es posible que nombres de municipio coincidan entre distintas provincias y, llegado a cierto punto, si la empresa quisiese dar cubrimiento internacional hay ciudades en distintos países con el mismo nombre. El resto de los atributos son campos *varchar* con una longitud estándar.

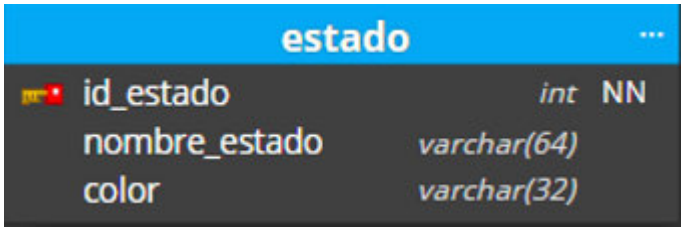


Ilustración 14. Especificación de la clase estado.

Los estados de entrega serán los estados en los que se puede encontrar un paquete. Los dos principales son “En reparto” y “Entregado”. Se le da un nombre y un color para poder hacer una representación gráfica del estado.

Ilustración 15. Especificación de la clase servicio.

Los servicios de entrega son los servicios proporcionados por la empresa para realizar la entrega de los correos. Ejemplos de servicio serán “Entrega a domicilio” o “Entrega Express”, creados con el propósito de darles prioridad a las entregas. Al igual que estado, se utiliza una id en caso de que distintos nombres de servicio puedan llegar a coincidir y como atributos tiene el nombre y la descripción del servicio.

Ilustración 16. Especificación de la clase oficina.

La oficina es la clase que representa las oficinas físicas de la empresa. Las oficinas tienen un identificador como clave primaria, un nombre y una dirección y cada oficina tiene que estar ubicada en una ciudad, por lo que el campo de id_ciudad es una clave foránea de la clase ciudad. Podría darse el caso de que hubiese vayas oficinas en la misma ciudad si por ejemplo hablamos de una gran ciudad, por eso no se limita la relación a *one to one* y se define como *many to one*.

Ilustración 17. Especificación de la clase código postal.

Para la tabla de código postal utilizamos el propio número de código postal como clave primaria y cada código postal está atendido por una oficina, por tanto, lo relacionamos con la tabla de oficina por medio de la clave foránea id_oficina.

Ilustración 18. Especificación de la clase correo.

La clase de correo referencia los correos o paquetes físicos con su información de origen, entrega, remitente y destinatario, es decir, la información física que va escrita en el sobre. El código postal es importante también para saber que oficina cubre la entrega del correo y para saber la zona de entrega, por tanto, se usa el atributo de código postal como clave foránea a la tabla de código postal.

Ilustración 19. Especificación de la clase mensajero.

Mensajero es la persona física que hace la labor entregar los paquetes. Sus atributos aportan información sobre la persona y su DNI es la clave primaria. Cada mensajero está asignado a una oficina y las tablas se relacionan con el atributo id_oficina como clave foránea.

Ilustración 20. Especificación de la clase entrega.

Entrega es la tabla que relaciona los correos con el mensajero que los entrega. Cada correo debe tener un solo mensajero asignado, un estado de la entrega y un servicio contratado. Además, tiene como atributos una fecha de envío y una fecha de entrega que se asignará una vez entregado el correo.

4.5. Fase de desarrollo

Una vez definido el modelo de datos y configurada una base de datos, se debe desarrollar la capa de lógica de negocio y construir la solución REST API. En esta fase se desarrollará el servidor web para que atienda a las peticiones del usuario desde el navegador. También se definirán las vistas de usuario y se trabajará la interfaz por medio de páginas web definidas en HTML.

4.5.1. Desarrollo en Python y envoltorios virtuales

Lo primero que se debe hacer al comenzar el desarrollo de un proyecto en Python es la creación de un envoltorio virtual en el que trabajar. De esa forma, se aísla la instalación de las dependencias y se puede replicar el entorno de desarrollo para el futuro despliegue. Más adelante, se generará el fichero de requisitos que incluya las dependencias utilizadas en el proyecto y sus versiones.

4.5.2. Algoritmos de procesamiento de información

Uno de los requisitos funcionales de la aplicación es que utilice información geográfica real. Podemos considerar información geográfica real los códigos postales y los municipios.

Para la obtención de los municipios de España usaremos información oficial del Gobierno de España. Desde la página web de datos del gobierno de España se puede obtener un archivo JSON con información sobre los municipios de España, sus provincias y sus comunidades autónomas.

La estructura del documento JSON que contiene los municipios es la siguiente:

```
[
  {
    "parent_code": "xx",
    "label": "Nombre de la comunidad autónoma",
    "code": "xx",
    "provinces": [
      {
        "parent_code": "xx",
        "code": "xx",
        "label": "Nombre de la provincia",
        "towns": [
          {
            "parent_code": "x",
            "code": "x",
            "label": "Nombre del municipio/ciudad"
          }
        ]
      }
    ]
  }
]
```

Para la obtención de los códigos postales y los municipios que abarcan se consulta una página web que actualiza los códigos postales según las versiones anuales de la Guía de Códigos Postales. La estructura del documento CSV que contiene los códigos postales es la siguiente:

```
codigo_postal,municipio_id,municipio_nombre
```

Los códigos postales, en muchos casos, vienen duplicados ya que puede darse el caso de que el mismo código postal forme parte de múltiples municipios. En el algoritmo se toma cada código postal y se asigna a la oficina del municipio en orden de aparición dado que cada código postal solo puede estar asignado a una oficina. Si encontramos un código postal ya asignado lo ignoraremos.

A continuación, se explica el pseudocódigo de procesamiento de la información para añadir toda la información de estos dos archivos a la base de datos:

Añadir las ciudades a la base de datos:

- Abrimos el fichero JSON
- Para cada comunidad guardamos el nombre en una variable
    - Para cada provincia guardamos el nombre en una variable
        - Para cada ciudad guardamos el nombre en una variable
        - Creamos una clase ciudad con los atributos nombre de ciudad, provincia y comunidad guardados.

Añadir las oficinas a la base de datos:

- Para cada ciudad
- Creamos una oficina con la información de la oficina y el nombre del municipio/ciudad

Añadir los códigos postales a la base de datos:

- Abrimos el fichero CSV como una tabla de datos
- Para cada fila en la tabla de datos (o para cada código postal)
    - Comprobamos si ya existe el código postal en la base de datos
    - Si no existe
        - Buscamos una oficina en ese municipio/ciudad y guardamos su identificador.
        - Si encontramos la oficina creamos un código postal al que atiende esa oficina.
        - Si no lo encontramos pasamos a la siguiente fila.
    - Si existe pasamos a la siguiente fila.

Se utiliza la biblioteca de Python, Pandas para tratar el fichero CSV como una tabla de datos ya que facilita la iteración por filas de una tabla de datos y la consulta de los campos utilizando el nombre de la columna como una cadenas de caracteres.

Para ello utilizaremos el método *pandas.read_csv* al que se le pasa como argumento la ruta de un fichero CSV y devuelve una variable de tipo tabla de datos desde la cual se pueden ejecutar métodos como *iterrows()* para poder iterar por las filas de la tabla.

4.5.3. Aplicación web

Como ya se ha mencionado antes, el desarrollo de la aplicación web se realizará utilizando el *framework* de desarrollo web Flask. Para atender a las solicitudes del cliente desde la aplicación web, se diseñará una solución de tipo API Rest que atienda a las peticiones realizadas por el usuario desde su navegador web.

Se puede distinguir entre dos tipos de peticiones que hacen referencia a métodos de HTTP. Los dos tipos principales de peticiones serán GET y POST. Las peticiones de tipo GET serán las que soliciten una representación del recurso especificado y el método se la devolverá. Las peticiones de tipo POST serán las que soliciten realizar la adición o modificación de alguno de los tres tipos de datos principales de la aplicación.

Para la solución de la persistencia y la definición de los tipos de datos, se ha usado la extensión SQLAlchemy adaptada a Flask en el plugin Flask-SQLAlchemy. Esta ORM permite definir las clases directamente en Python y la creación del modelo de datos desde los objetos definidos en el código.

Para la interfaz de usuario se comenzará utilizando un sencillo prototipo basado en HTML básico y en la renderización de las páginas HTML con Python para la inserción de la información. Una vez probado el correcto funcionamiento de la API con este prototipo se actualizará a una plantilla de diseño web basada en componentes de Bootstrap 5.

4.5.3.1. Back-end

El *back-end* de una aplicación web se refiere a la parte de la aplicación que se encarga de la lógica y la funcionalidad que ocurre en la capa de negocio. Es responsable de procesar y gestionar los datos, interactuar con la base de datos, manejar la lógica de negocio, autenticar usuarios, y generar la respuesta que será enviada al *front-end* para su visualización en el navegador del usuario.

4.5.3.1.1. Definición de las clases como objetos Python con Flask-SQLAlchemy

Con el plugin Flask-SQLAlchemy, utilizamos el ORM que nos ofrece para definir las clases directamente en la base de datos utilizando la instancia de SQLAlchemy llamada db.

```
class Ciudad(db.Model):

    id_ciudad = db.Column(db.Integer, primary_key=True)
    nombre_ciudad = db.Column(db.String(64))
    provincia = db.Column(db.String(64))
    comunidad_autonoma = db.Column(db.String(64))
    oficinas = db.relationship('Oficina', backref='ciudad', lazy=True)
```

```
class Servicio(db.Model):
    id_servicio = db.Column(db.Integer, primary_key=True)
    nombre_servicio = db.Column(db.String(64))
    descripcion_servicio = db.Column(db.String(512))
    entregas = db.relationship('Entrega', backref='servicio', lazy=True)
```

```
class Mensajero(db.Model):
    dni = db.Column(db.String(16), primary_key=True, nullable=False)
    nombre = db.Column(db.String(64))
    apellidos = db.Column(db.String(64))
    numero_tlf = db.Column(db.String(16))
    foto = db.Column(db.String(128))
    matricula = db.Column(db.String(32))
    id_oficina = db.Column(db.Integer, db.ForeignKey("oficina.id_oficina"),\
    nullable=False)
    entregas = db.relationship('Entrega', backref='mensajero', lazy=True)
```

```
class Codigo_postal(db.Model):

    codigo_postal = db.Column(db.String(16), primary_key=True)
    id_oficina = db.Column(db.Integer, db.ForeignKey("oficina.id_oficina"),\
    nullable=False)
    oficinas = db.relationship('Oficina', backref='codigo_postal', lazy=True)
```

```
class Correo(db.Model):
    id_correo = db.Column(db.Integer, primary_key=True)
    remitente = db.Column(db.String(64))
    destinatario = db.Column(db.String(64))
    direccion_remitente = db.Column(db.String(128))
    direccion_destinatario = db.Column(db.String(128))
    codigo_postal = db.Column(db.String(16),\
    db.ForeignKey("codigo_postal.codigo_postal"), nullable=False)
    entrega = db.relationship('Entrega', back_populates="correo")
```

```
class Entrega(db.Model):
    id_correo = db.Column(db.Integer, db.ForeignKey("correo.id_correo"),\
    primary_key=True, nullable=False)
    correo = db.relationship('Correo', back_populates="entrega")
    mensajero_dni = db.Column(db.String(16), db.ForeignKey("mensajero.dni"),\
    nullable=False)
    id_estado = db.Column(db.Integer, db.ForeignKey("estado.id_estado"),\
    nullable=False)
    id_servicio = db.Column(db.Integer, db.ForeignKey("servicio.id_servicio"),\
    nullable=False)
    fecha_envio = db.Column(db.Date())
    fecha_entrega = db.Column(db.Date())
```

```
class Estado(db.Model):
    id_estado = db.Column(db.Integer, primary_key=True)
    nombre_estado = db.Column(db.String(64))
    color = db.Column(db.String(32))
```

```
class Oficina(db.Model):
    id_oficina = db.Column(db.Integer, primary_key=True)
    nombre_oficina = db.Column(db.String(64))
    direccion_oficina = db.Column(db.String(128))
    id_ciudad = db.Column(db.Integer, db.ForeignKey("ciudad.id_ciudad"),\
    nullable=False)
    codigos_postales = db.relationship('Codigo_postal', backref='oficina', lazy=True)
    mensajeros = db.relationship('Mensajero', backref='oficina', lazy=True)
```

4.5.3.1.2. API REST

El funcionamiento de una API está basado en dar respuesta a peticiones o mensajes HTTP. Estos mensajes se lanzan desde el navegador al servidor web y el servidor web responde. La respuesta puede ser una página HTML y/o una acción por parte del servidor.

En el caso de la solución se trabajará en una API Rest con un modelo CRUD para los tres tipos de datos principales. El funcionamiento de la API es muy similar para los tres tipos de datos, simplemente, se adaptan los parámetros de las peticiones al tipo de dato que corresponden.

Los *endpoints* de una API son métodos que aceptan peticiones y devuelven respuestas, por tanto, para la API se tienen cuatro tipos de *endpoints* (ver, añadir, eliminar y actualizar) para cada uno de los tres tipos de datos. Por ello, se definen 12 métodos o *endpoints* en esta API además de otros que renderizan las páginas HTML de error.

En Flask los *endpoints* o rutas se definen utilizando la etiqueta *@app.route()* lo cual especifica la regla o URL que se utiliza para llegar a los distintos lugares de la página web o para realizar las distintas acciones sobre la base de datos. [4]

A continuación, se detallará cada uno de los métodos clasificados por su tipo de operación CRUD y se enunciarán las distintas rutas o *endpoints* de la aplicación.

- LEER (READ)

Este método atiende a una petición HTTP de tipo GET y devolverá la página HTML renderizada con toda la información necesaria.

Más adelante, en el apartado de la interfaz se justificarán las decisiones de diseño, para este apartado. Cabe mencionar que la interfaz que muestra cada uno de los tipos de datos principales permite acceder al resto de funciones CRUD (adición, actualización y eliminación). Por tanto, para las páginas HTML necesitaremos información relacionada con la creación de los datos. Las peticiones de este tipo que atiende la API son:

/ver_correos
/ver_mensajeros
/ver_entregas

- CREAR (CREATE)

Este método atiende a una petición de tipo POST. El método obtiene la información de los campos de un formulario como parámetros y crea una instancia de un objeto. Ese objeto se añade a la sesión y, a continuación, con un *commit* se añade a la base de datos. Para entender el funcionamiento de este método y del posterior *update*, es necesario indagar en la clase de Flask *request*, especialmente en su atributo *form*, que contiene los datos de un formulario de una petición tipo POST o PUT.

Las peticiones de este tipo que atiende la API son:

/add_correo
/add_mensajero
/add_entrega

- ELIMINAR (DELETE)

Para la eliminación de un dato se utilizará un método GET con un parámetro que contiene el id del objeto que se quiere eliminar de la base de datos. Además, el método redirigirá al *endpoint* de lectura de ese tipo de datos. Por tanto, el pseudocódigo para el proceso de la eliminación de un objeto es:

- Se obtiene el objeto dado su id, se envía como parámetro de la petición y es un argumento del método.
- Se elimina el objeto de la sesión y se hace un *commit* para eliminarlo de la base de datos.
- Se redirige al método de lectura lo que hace que se actualice la página con los datos actualizados y con el objeto eliminado.

Las peticiones de este tipo que atiende la API son:

/delete_correo/<id>
/delete_mensajero/<id>
/delete_entrega/<id>

- ACTUALIZAR (UPDATE)

Este método atiende a dos tipos de peticiones y dependiendo del tipo lleva a cabo distintas acciones. Si la petición que recibe es del tipo GET, debe contener como parámetro la id del dato que se desea actualizar y se devolverá una página HTML con un formulario que contiene toda la información de ese dato y permite su actualización. Si el tipo de petición es POST deberá incluir como parámetro la id del dato que se desea actualizar y el método obtendrá de los campos de un formulario los nuevos datos y los actualizará en la sesión. El pseudocódigo del método es el siguiente:

- Se obtiene el objeto dado su id, se envía como parámetro de la petición y es un argumento del método.
- Se lanza una *query* a la base de datos y se obtiene el objeto.
- Si la petición es de tipo POST se actualizan los campos del objeto y se redirige a la página de visualización de los datos con los datos actualizados.
- Si hemos llegado hasta aquí (la petición no era de tipo POST, por tanto, es un GET) se renderiza la página web de actualización de un objeto con toda la información necesaria para ser mostrada.

Las peticiones de este tipo que atiende la API son:

/update_correo/<id>
/update_mensajero/<id>
/update_entrega/<id>

La API también contempla otras peticiones GET para mostrar páginas HTML. Estas páginas se cargan utilizando un método que trabaja como /<template> cuya utilidad es cargar distintas páginas HTML que no requieran renderizado y por tanto generaliza su uso. Por ejemplo, se puede cargar el menú principal de la aplicación con /index o la página de autenticación con /login.html

El método también realiza el control de errores devolviendo códigos 404 si no encuentra la página y 500 para errores de servidor internos ante cualquier excepción no contemplada.

4.5.3.1.3. Autenticación

Flask-Login es una extensión de Flask que facilita la gestión de la autenticación de usuarios en aplicaciones web. Proporciona un conjunto de herramientas que permiten gestionar sesiones de usuario lo que permite saber si un usuario está autenticado o no. Esto es esencial para proteger ciertos *endpoints* y recursos dentro de la aplicación, asegurando que solo los usuarios autorizados puedan acceder. [13]

Además, Flask-Login facilita la implementación de funcionalidades de inicio y cierre de sesión mediante funciones como *login_user()* y *logout_user()*. Esto no solo mejora la seguridad, sino que también agiliza la experiencia del usuario al interactuar con la aplicación.

Para la protección de las rutas, Flask-Login utiliza el decorador *@login_required*. Esto asegura que solo los usuarios autenticados puedan acceder a ciertas partes de la aplicación, añadiendo una capa adicional de seguridad sin complicaciones innecesarias.

Flask-Login también incluye soporte para recordar la sesión de usuario mediante cookies seguras. Esta funcionalidad permite a los usuarios permanecer autenticados incluso después de cerrar el navegador, mejorando la conveniencia y usabilidad de la aplicación.

El modelo de datos para los usuarios y las sesiones de usuario se encuentra definido en el *template* de desarrollo que explicaremos en los siguientes apartados de la memoria.

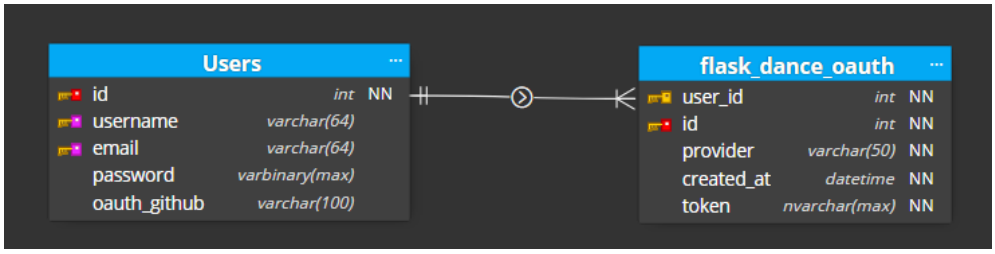


Ilustración 21. Modelo de datos para la autenticación de usuarios.

4.5.3.2. Front-end

El *front-end* de una aplicación web se refiere a la parte de la interfaz con la que interactúa directamente el usuario. Incluye todos los elementos visibles y las funcionalidades que el usuario puede ver y con las que puede interactuar en el navegador web, como el diseño, la presentación visual, la estructura de la página, los formularios, los botones y otros elementos interactivos.

4.5.3.2.1. Plantillas HTML (Templates)

Las plantillas son ficheros definidos en HTML con datos estáticos y además *placeholders* para datos dinámicos. Las plantillas se renderizan individualmente para producir páginas HTML finales que se presentan al usuario en su navegador web. Flask utiliza la librería Jinja para renderizar las plantillas.

Jinja ofrece una sintaxis similar a Python que se utiliza para realizar todas las funciones relacionadas con la interacción entre plantillas e información devuelta por la Rest API. En el caso del proyecto se utilizan las plantillas para iterar por listas de objetos Python devueltas por la API y acceder a sus atributos representándolos en una tabla HTML. [14]

Para la aplicación web se han usado plantillas para mostrar cada una de las páginas HTML que representan los tres tipos de dato principales. Por tanto, el pseudocódigo para el funcionamiento de una plantilla que muestra la información de todos los objetos de tipo correo es el siguiente:

- En la plantilla HTML se define una tabla siendo la primera fila los campos y en la que para cada correo se crea una fila nueva y se rellena cada una de las columnas con los atributos del objeto.
- El usuario invoca mediante una llamada GET al método ver_correos de la API, el cual hace una consulta a la base de datos y devuelve una lista de objetos de tipo correo.
- Se renderiza la información estática de la plantilla y para cada objeto correo de la lista:
  - Obtenemos sus atributos y los rellenamos en las distintas casillas de la fila.
- Se devuelve una página HTML con información que para el usuario es estática y se representa en su navegador web.

Flask utiliza el métodos *render_template()* para renderizar las plantillas HTML y se le pasa el fichero que contiene la plantilla y los objetos que necesitará para renderizarla, que por lo general, son la respuesta a la consulta a la base de datos.

Los métodos *url_for()* y *redirect()* de Flask se utilizan para crear el flujo de la aplicación web y no devolver páginas web en *endpoints* de la API que no deben devolver páginas web. Por ejemplo, los *endpoints* que hacen referencia a *Create* para crear nuevas instancias de objetos en la base de datos, redirigen al *endpoint* de lectura ya que tras añadir un objeto queremos que se redirija a la página que muestra la información con la página actualizada.

4.5.3.2.2. Plantilla de diseño web VOLT y Bootstrap 5

Siguiendo el paradigma de la reutilización en Python, existen múltiples plantillas de desarrollo web que ofrecen una solución funcional y un conjunto de componentes listos para utilizar y estilizados para presentar un diseño profesional.

Trabajar con una plantilla web significa que se reduce el trabajo de diseño, pero no el de desarrollo, ya que, aunque ofrece una solución de diseño con componentes reutilizables, hay que adaptarlos a los requisitos de la aplicación. La mayoría de las plantillas, incluyen páginas de ejemplo que despliegan todos los componentes disponibles y ofrecen el código HTML para usarlos en nuestros propios diseños.

Bootstrap 5 es un *framework* HTML, CSS y JavaScript utilizado para crear páginas web adaptativas, es decir, que su diseño se ajuste automáticamente al tamaño de la ventana del navegador y al dispositivo desde el que se accede. Al ya estar incluido su uso en las plantillas de diseño web, no necesitamos conocerlo en profundidad ya que solo utilizaremos componentes.

En el caso específico de la aplicación a desarrollar, se ha trabajado sobre el *template* VOLT producido por Themesberg que muestra varias páginas de componentes de ejemplo y ofrece plantillas de diseño con páginas vacías. De las páginas de componentes se han utilizado los botones, las tablas y los formularios.

VOLT, también incluye una estructura de aplicación web que tan solo se tiene que adaptar para ajustarse a las necesidades del proyecto. Se incluye el sistema de autenticación y registro de usuarios, la estructura básica de la aplicación y un sistema para realizar la conexión con distintos sistemas de base de datos.

Para la aplicación en concreto se ha tenido que adaptar la plantilla añadiendo todas las clases que constituyen el modelo de datos, adaptar la conexión para realizarse con una base de datos hospedada en Azure y añadir todas las rutas que definen los *endpoints* de la API Rest, además de muchos otros pequeños cambios para el correcto funcionamiento. A nivel de *front-end* se han creado las 4 páginas que constituyen las vistas principales de la aplicación utilizando los componentes que se ofrecen como ya se ha mencionado. [15]

4.6. Fase de migración

Una vez probado el funcionamiento de todos los componentes de la aplicación de manera local, se migrarán los componentes a la nube y se desplegará el servidor web en la plataforma Azure. En esta fase se realizará todo lo relativo a la migración de los servicios a la nube, cambiando parámetros como la conexión entre base de datos y web server y solucionando los problemas que conlleve el hospedaje en la nube.

4.6.1. Hospedaje de la base de datos

El primer paso de la migración de la aplicación web será la creación de un servidor de base de datos en la nube. En Microsoft Azure se necesita crear un grupo de recursos que contenga todos los componentes. Para la base de datos se deben desplegar dos componentes: Una SQL *database* y un SQL Server que lo contenga.

El SQL Server constituye el componente que virtualiza la plataforma que contiene la base de datos. Lo más importante para este componente es la configuración del firewall, añadiendo reglas que permitan la conexión al servidor desde ciertas IPv4s. En el caso de la aplicación se han configurado dos reglas, una que permite la conexión desde la IP pública de la red privada en la que se han realizado todas las pruebas locales y la IPv4 virtual del componente de la aplicación web del que hablaremos más adelante.

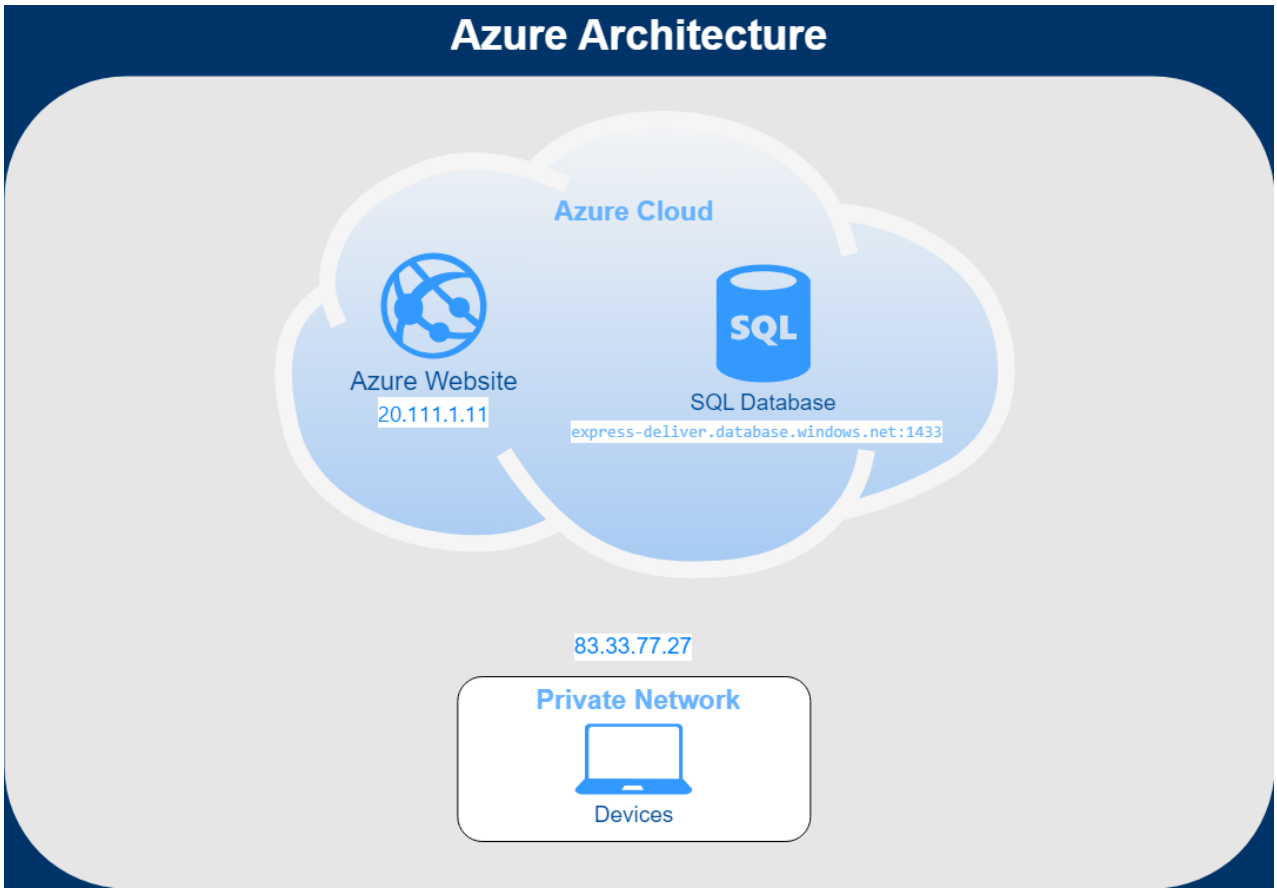


Ilustración 22. Diagrama de la arquitectura de red en la nube

El SQL *database* es el componente base de datos de la aplicación. La instancia de SQLAlchemy se conecta con este componente para crear la sesión y para gestionar toda la información de la base de datos. Para la conexión se utiliza la interfaz a nivel de llamada ODBC, por lo que es necesario el plugin *pyodbc* en la máquina del web server para realizar las pruebas de funcionamiento. Es importante mencionar que se debe añadir esta librería al fichero de *requirements* para que el despliegue del web server funcione. Debido a que Azure nos ofrece la cadena de conexión ODBC directamente, se utilizará el módulo *urllib* de Python y el método *urllib.parse* para crear la instancia del objeto params que contendrá los parámetros de la conexión, como la url, el usuario y la contraseña. Además, se tiene que especificar que es una cadena de conexión ODBC en el parámetro SQL_ALCHEMY_DATABASE_URI.

El código es el siguiente:

```
# Configure Database URI:

        # Parseamos los parámetros de la conexión
        params = urllib.parse.quote_plus("Driver={ODBC Driver 18 for SQL Server};\
        Server=tcp:express-deliver.database.windows.net,1433;Database=express-deliver-db;\
         Uid=express-admin;Pwd=contraseña;Encrypt=yes;TrustServerCertificate=no;Connection\
        Timeout=30;")

# Initialization

        # Instanciamos Flask
        app = Flask(__name__)

        # Existe un fichero de configuración proporcionado por la plantilla para otro tipo de
        servidores de base de datos como MySQL.
        app.config.from_object(config)

        # Aquí tenemos que sobrescribir ciertos parámetros del fichero de configuración
        app.config['SQLALCHEMY_DATABASE_URI'] = "mssql+pyodbc:///?odbc_connect=%s" % params

        register_extensions(app)
```

4.6.2. Despliegue del servidor web

Microsoft incluye una guía detallada paso por paso de cómo realizar el despliegue en Azure de aplicaciones web Flask o Django. A continuación, se sintetizará la guía y se adapta al caso de uso:

(i) El primer paso será crear un contenedor de aplicación web. Para ello desde la interfaz de usuario, se navegará por las distintas pestañas y se creará un contenedor de aplicación web con Python como plataforma. También se tendrá que elegir una región para lo cual seleccionaremos la que más nos convenga por distancia y precio.

(ii) Se activará el parámetro de configuración SCM_DO_BUILD_DURING_DEPLOYMENT del contenedor de la web app. Este paso se realizará por medio de un comando desde la terminal de PowerShell.

(iii) Se creará un archivo comprimido de la aplicación que incluya todo lo necesario para el despliegue incluyendo el fichero de *requirements*. A continuación, se ejecutará el comando de despliegue indicando el *path* de nuestro fichero zip.

(iv) Azure se encarga de realizar el despliegue de la aplicación y de la instalación de todas las dependencias necesarias.

(v) Se puede ejecutar un comando en PowerShell para capturar el flujo de los logs de la aplicación en tiempo real y poder llevar a cabo el control de errores.

4.7. Seguridad

Los puntos más importantes para la seguridad de la aplicación son los siguientes:

(i) Acceso a la base de datos: Haciendo referencia a la ilustración 22 que constituye el diagrama de la arquitectura red, el firewall del servidor de la base de datos en Azure está configurado con reglas para que tan solo puedan conectar en el servidor tanto la IPv4 pública de la red de la casa del alumno como la IPv4 virtual del contenedor del servidor web.

(ii) Acceso a la aplicación web: El acceso a los *endpoints* de gestión de información de la aplicación web requiere una autenticación por parte del usuario. El modelo de usuario se guarda en la misma base de datos que la información empresarial. El registro de usuarios lo realizará un administrador del sistema. Cualquier usuario puede acceder a la aplicación web, pero solo un usuario autenticado puede acceder a todas las funciones de la aplicación.

Todas las demás implicaciones de seguridad son responsabilidad de Microsoft Azure, la única responsabilidad de desarrollo es asegurarse de que nadie pueda acceder al sistema.

5. Resultados

Desde la máquina del servidor web, antes de comenzar con el desarrollo, se ha utilizado la interfaz MySQL Workbench para realizar una prueba de conexión exitosa con la máquina que hospeda el servidor de base de datos. Esta prueba confirma el correcto funcionamiento del servidor de base de datos de prueba que será accesible durante el desarrollo de la aplicación web desde otra máquina, simulando así, una base de datos hospedada en la nube.

5.1. Escenarios

El desarrollo del prototipo de la aplicación se ha realizado de forma modular para aislar cada uno de los componentes y capas de la aplicación, evitando errores y asegurando el correcto funcionamiento. Además, se ha llevado a cabo de manera secuencial, lo que significa que cada parte de la aplicación no se ha desarrollado hasta que la anterior funcionase y se sometiese a pruebas de funcionamiento.

Siguiendo estos principios se pueden dividir los resultados en dos escenarios:

(i) Escenario de pruebas local: En este escenario se ha trabajo usando máquinas virtuales y probando la base de datos y la aplicación web de forma local. La conexión entre los componentes de la aplicación se ha realizado en una red virtual.

(ii) Escenario de pruebas en la nube o real: Una vez probado el correcto funcionamiento de la aplicación de forma local se ha realizado el proceso de migración de los componentes y se realizan pruebas de funcionamiento con los componentes hospedado en la nube.

A continuación, se describirán las pruebas realizadas y los resultados obtenidos en cada uno de los escenarios de pruebas.

5.2. Escenario de pruebas local

En este primer escenario se ha desarrollado la aplicación web en un entorno de pruebas local basado en máquinas virtuales utilizando el software de virtualización Oracle VirtualBox. Una máquina simula ser el servidor de base de datos, con un servidor de base de datos MySQL y otra máquina que lleva a cabo la función de servidor web, en la que también se han realizado las tareas de desarrollo con Visual Studio Code y todas las herramientas de desarrollo de la aplicación web.

5.2.1. Base de datos

Se ha instalado un servidor MySQL en una máquina virtual de la red local que funcionará a modo de servidor de base de datos. La instalación se ha realizado mediante el asistente de instalación de MySQL en el puerto por defecto 3306. Se ha creado un usuario *root* con todos los permisos y se ha permitido el uso remoto de este usuario.

5.2.2. Aplicación web/Máquina de desarrollo

5.2.2.1. Fase 1

La primera fase de la aplicación web consistirá en el desarrollo del servidor web y de un correcto funcionamiento de la API Rest. Temporalmente se crearán unas plantillas HTML básicas que presenten la información obtenida para poder probar el correcto funcionamiento del *back-end* del servidor web.

La máquina de desarrollo y servidor web contendrá el entorno de desarrollo y todos los paquetes necesarios para hospedar el servidor web localmente. Para el desarrollo se ha utilizado un envoltorio virtual para asegurar la compatibilidad y la replicabilidad del entorno para un futuro despliegue.

Las plantillas HTML utilizan un motor de plantillas con sintaxis basada en jinja para insertar líneas de código similares a la sintaxis de Python. Estas líneas de código se utilizan para la representación de la información directamente desde objetos Python, pudiendo acceder a sus atributos o iterando por listas de objetos.

Para esta primera fase no se han utilizado estilos para las plantillas ya que solo se pretende obtener la información y representarla a modo de prueba de funcionamiento. A continuación, se muestran capturas de la aplicación web que prueban el funcionamiento.

Ilustración 23. Interfaz de usuario de HTML básico, índice.

El índice de la aplicación web muestra un botón para realizar el procesamiento de la información de los ficheros JSON y CSV de códigos postales y municipios y poblar con esa información la base de datos. Después se muestran cuatro enlaces que hacen referencia a los métodos GET de la API Rest para la obtención de la plantilla que presenta la información de los distintos objetos que componen la aplicación.

Ilustración 24. Interfaz de usuario de HTML básico, ver correos.

Cada una de las vistas devuelve una página HTML que contiene un formulario para la inserción de datos y una tabla que representa la información de los objetos. El formulario contiene los campos del objeto y al utilizar el botón de "Añadir" se llama al método *create* de la API que añade un nuevo objeto a la base de datos por medio de una petición HTTP de tipo POST. Asimismo, al final del método se redirige a la misma página con la información actualizada.

Ilustración 25. Interfaz de usuario de HTML básico, añadir un correo.

De igual manera, para cada elemento se incluyen dos columnas que contienen dos hipervínculos, uno para actualizar y uno para eliminar. El vínculo de actualizar invoca al método de la API *update* lo que para un GET devuelve una plantilla que incluye un formulario con la información del objeto que se quiere actualizar y con un botón. Al pulsar el botón se vuelve a invocar al método *update* pero esta vez con un POST lo que hace que se actualice la información de un objeto con la nueva información.

Ilustración 26. Interfaz de usuario de HTML básico, actualizar un correo

En este caso se ha actualizado el código postal satisfactoriamente. El otro vínculo de la columna "eliminar" contiene una invocación al método de la API *delete* el cual mediante el identificador del objeto utiliza un POST para obtener el objeto de la base de datos y posteriormente eliminarlo de la misma.

Ilustración 27. Interfaz de usuario de HTML básico, eliminar un correo.

Todo el funcionamiento descrito es el mismo para los tres tipos de dato principales, correo, mensajero y entrega, lo cual concluye la fase de pruebas de la API Rest y del back-end del servidor web.

Además del correcto funcionamiento del back-end de la aplicación web, se prueba también que la definición del modelo de datos se haya realizado correctamente ya que, ante cualquier error en la definición, Flask nos avisará al iniciarse el servidor web y cargar la instancia de SQLAlchemy.

5.2.2.2. Fase 2

Para la segunda fase de pruebas locales se ha implementado la plantilla de diseño web VOLT basada en Bootstrap 5. Para esta fase se pretende verificar el funcionamiento total de la aplicación de forma local al igual que en la fase uno, pero reutilizando todo sobre una plantilla.

La primera novedad que se ha introducido en la fase 2 de desarrollo ha sido la página de log-in incluida en la plantilla de diseño. Al acceder a la aplicación web será lo primero que se muestre.

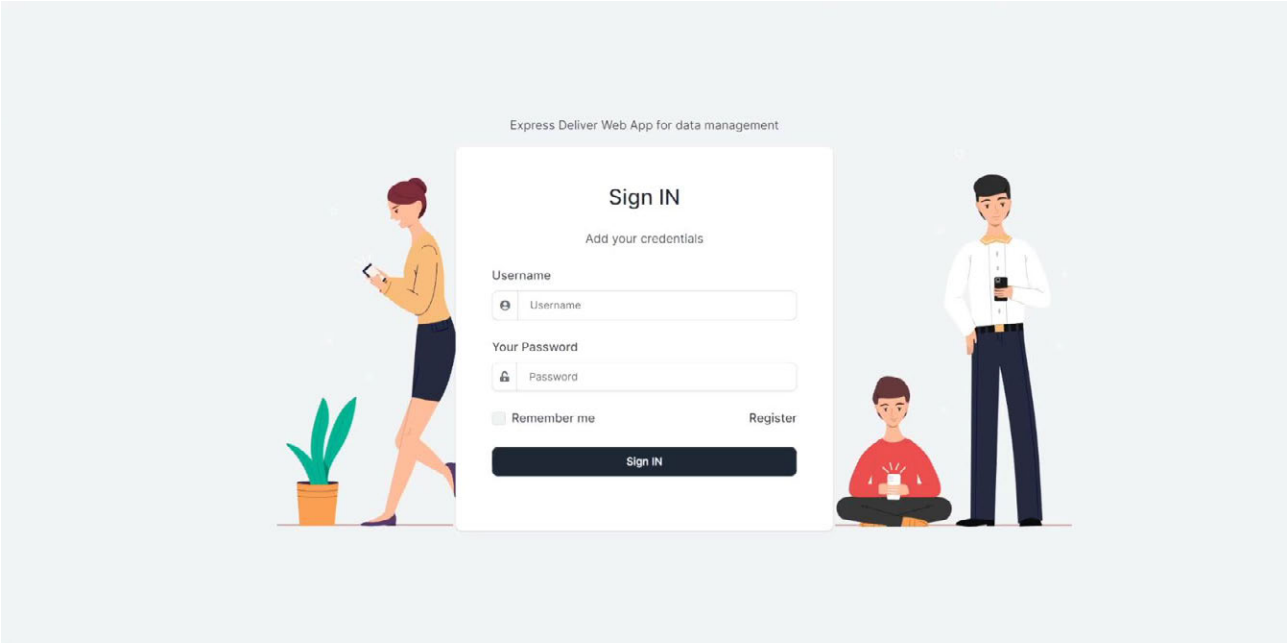


Ilustración 28. Interfaz de usuario final, autenticación.

Una vez autenticado el usuario se redirige al verdadero índice de la aplicación web desde el que acceder a todas las vistas de los tipos de dato principales mediante una barra lateral.

Ilustración 29. Interfaz de usuario final, índice.

Correos, mensajeros y entregas cuentan con una página desde la que se puede visualizar una tabla con todos los datos y un formulario para añadir nuevos datos. El botón de añadir correo invoca al método de *add_correo*, el botón de editar correo invoca al método *update_correo* pasándole el identificador del propio correo desde el que se invoca y el botón de eliminar invoca a *delete_correo* con el id.

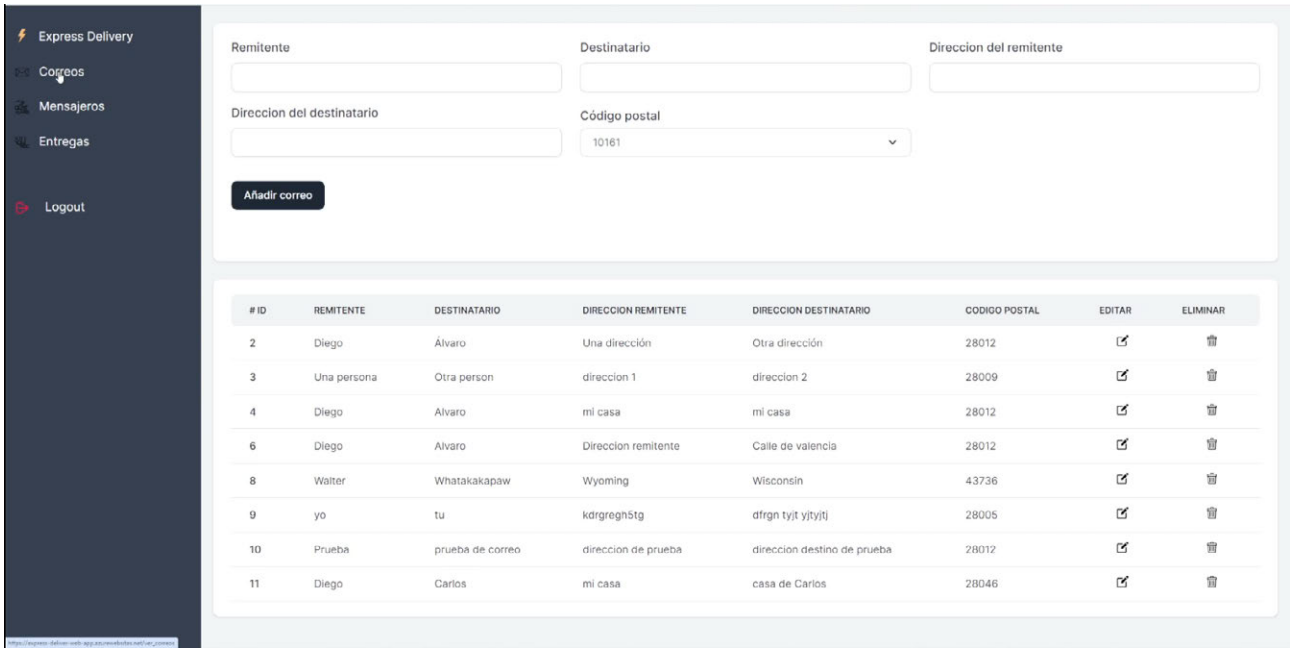


Ilustración 30. Interfaz de usuario final, ver correos.

Al añadir nuevos datos utilizando el formulario de adición y pulsando el botón de añadir, la tabla se actualiza dinámicamente con los datos ya que el método de adición utiliza el método *redirect*() para volver a renderizar la vista con los datos actualizados. El funcionamiento es el mismo para mensajeros y entregas.

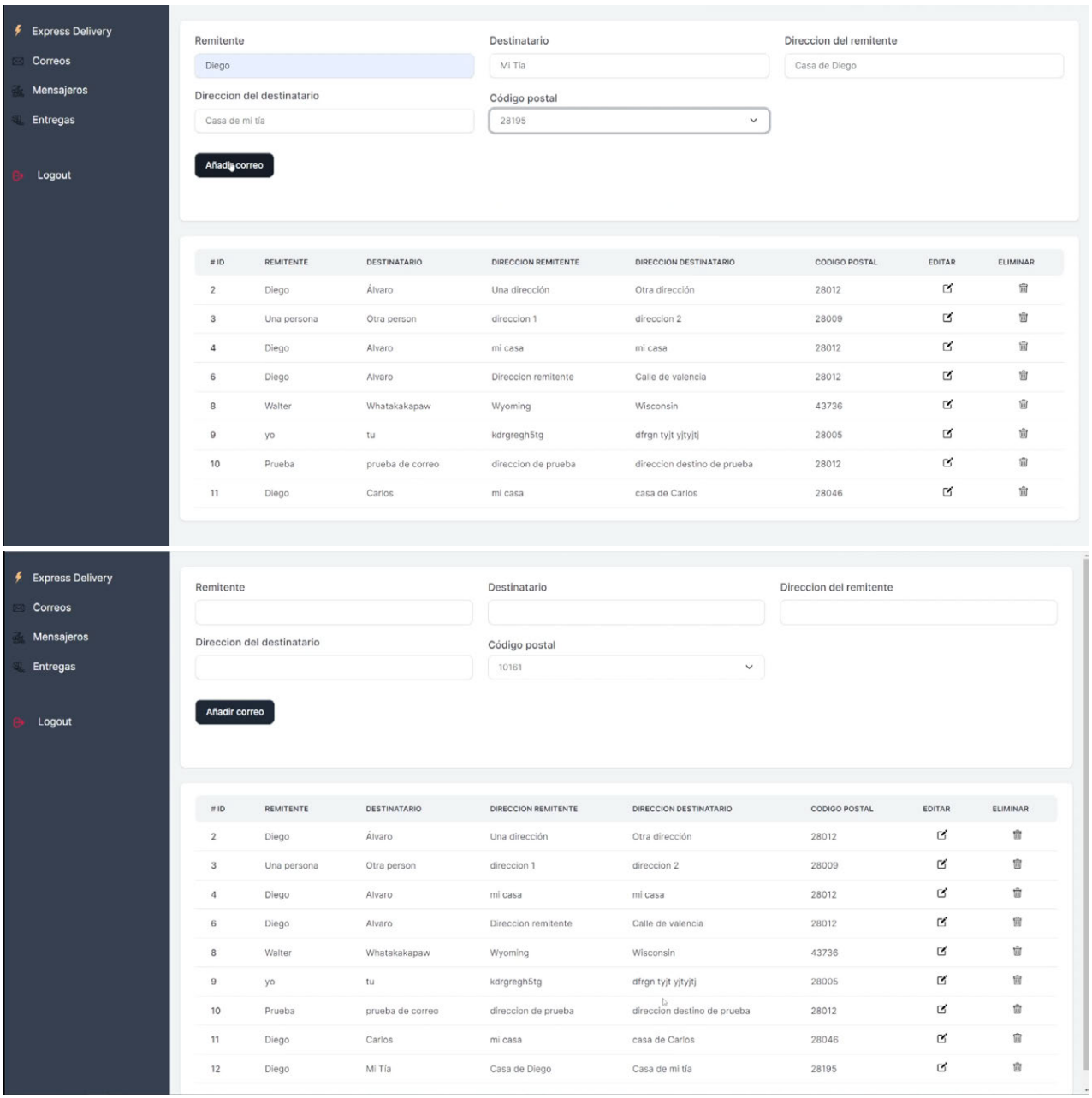


Ilustración 31. Interfaz de usuario final, añadir correo.

En la ilustración se puede observar cómo utilizamos el formulario de adición, pulsamos el botón y a continuación se añade el nuevo correo que hemos creado a la tabla.

Haciendo click en el botón de editar se redirigirá a otra vista con los datos del objeto que queremos actualizar. El botón de volver sirve para descartar cambios y volver a la vista. El botón de actualizar sirve para hacer efectivos los cambios mandando la petición POST y volviendo a la vista anterior.

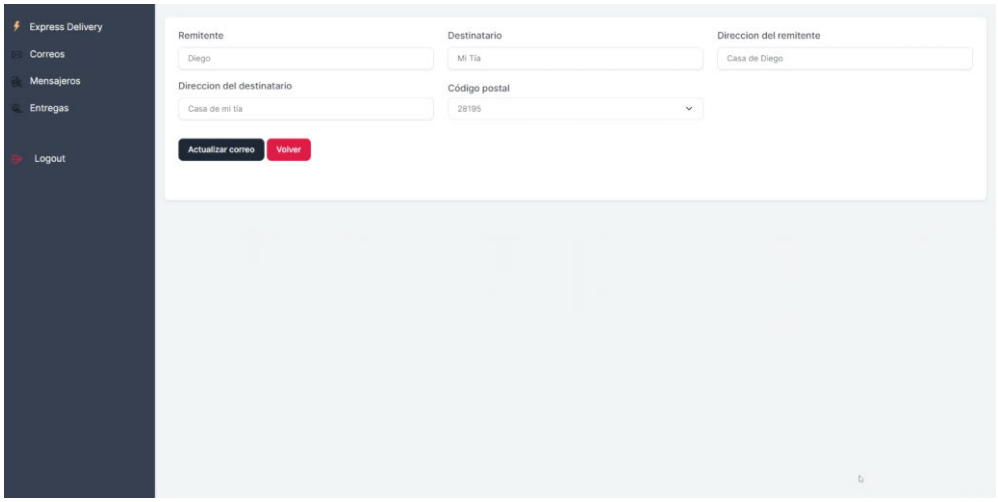


Ilustración 32. Interfaz de usuario final, actualizar correo.

Mensajeros y Entregas cuentan con vistas similares pero adaptadas a los atributos y a los datos que se necesitan. A continuación, se muestran

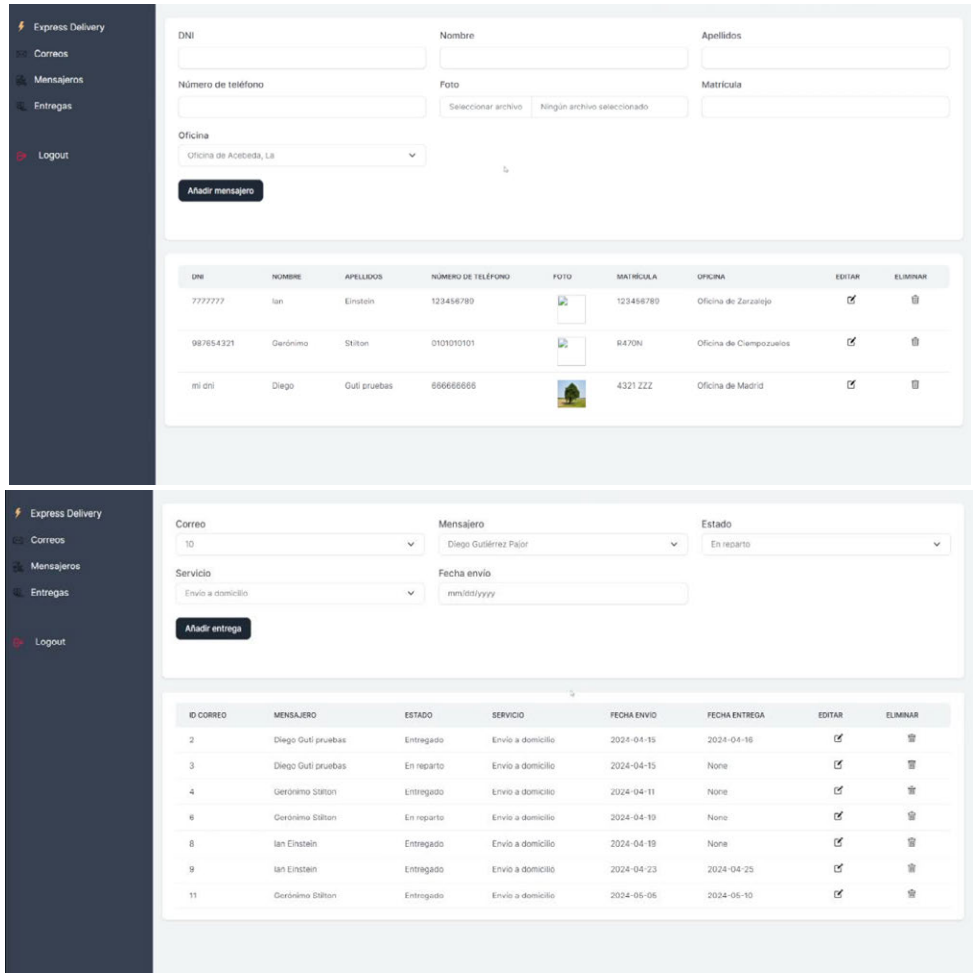


Ilustración 33. Interfaz de usuario final, vistas de mensajeros y entregas.

5.3. Escenario real/Escenario Cloud

Tras probar el funcionamiento de todos los componentes y de la aplicación web hospedada localmente en una máquina virtual, se probará el funcionamiento de la aplicación web hospedada en la nube. Primeramente, se hospedará tan solo la base de datos y una vez probada la cadena de conexión se realizará la prueba completa con todo el sistema hospedado en el servicio Cloud.

El grupo de recursos contendrá los siguientes componentes: un SQL Server que contiene un SQL database y para la aplicación web un contenedor App Service.

Ilustración 34. Componentes de la arquitectura Cloud.

Tanto servidor SQL como base de datos se han configurado utilizando el CLI de Azure y siguiendo los pasos que Azure especifica en los que se elige un plan de costes, unos recursos de procesamiento, la localización geográfica y la configuración de opciones como cargas de trabajo, tipo de autenticación, firewall…

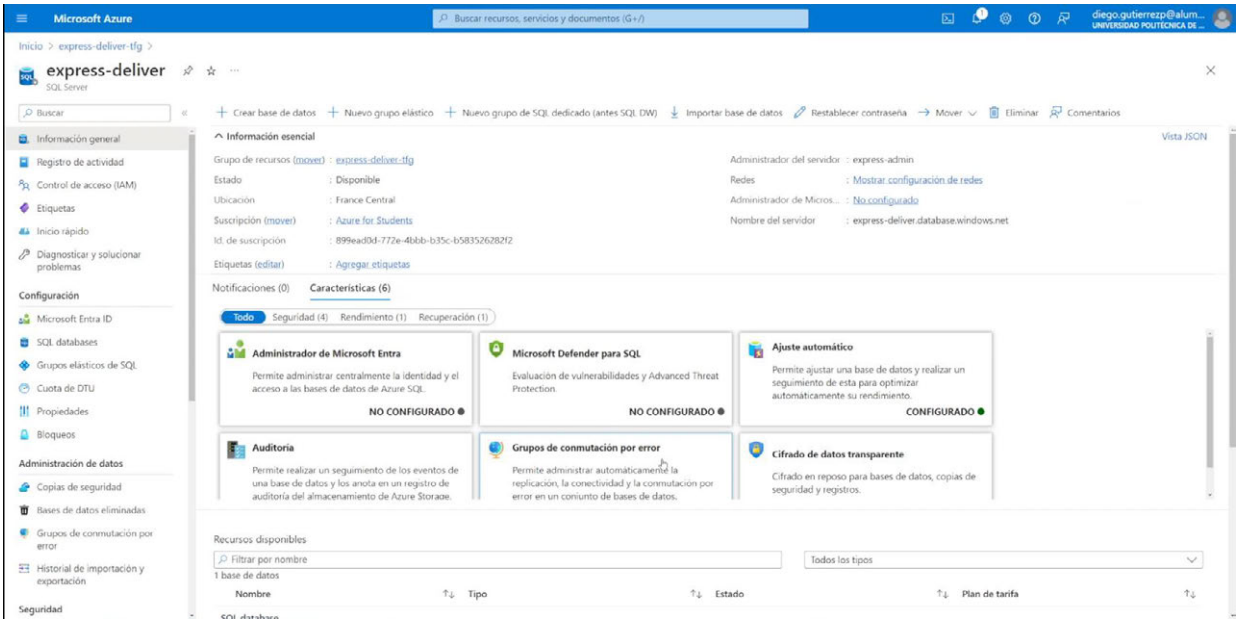


Ilustración 35. Componente Cloud SQL Server.

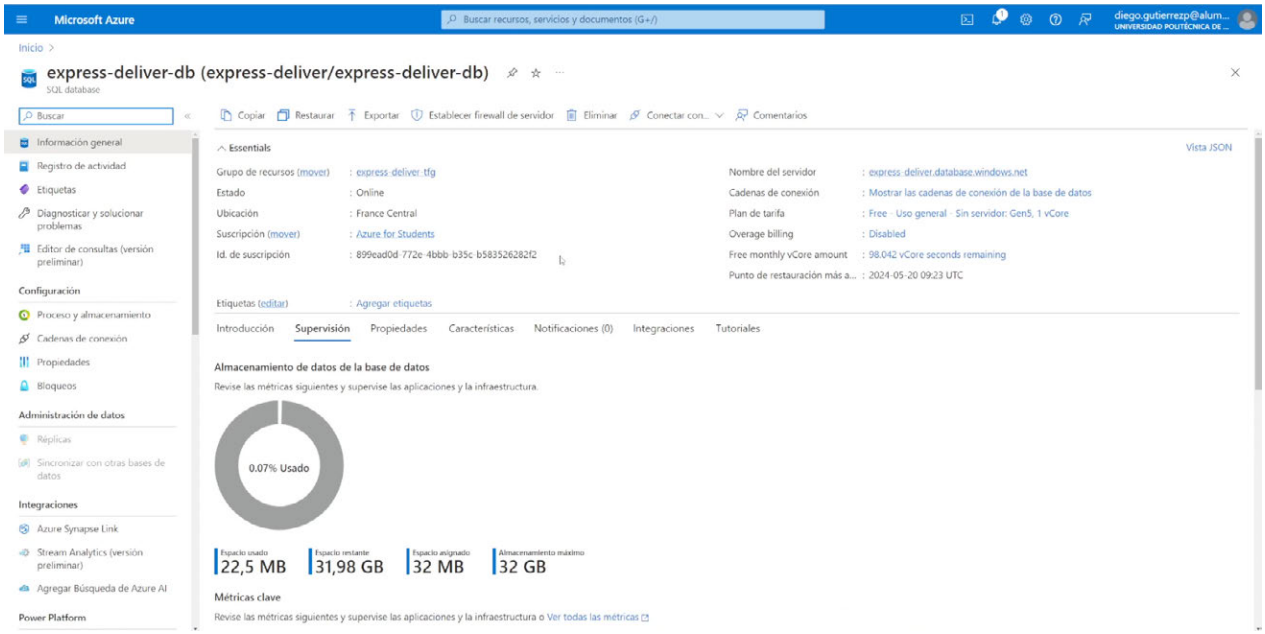


Ilustración 36. Componente Cloud SQL database.

El componente del App Service se ha configurado tanto mediante el CLI de Azure como utilizando comandos desde el intérprete de comandos de Azure para PowerShell. Desde Powershell se ha realizado el despliegue de la aplicación web subiendo un archivo comprimido con todo lo necesario para el despliegue.

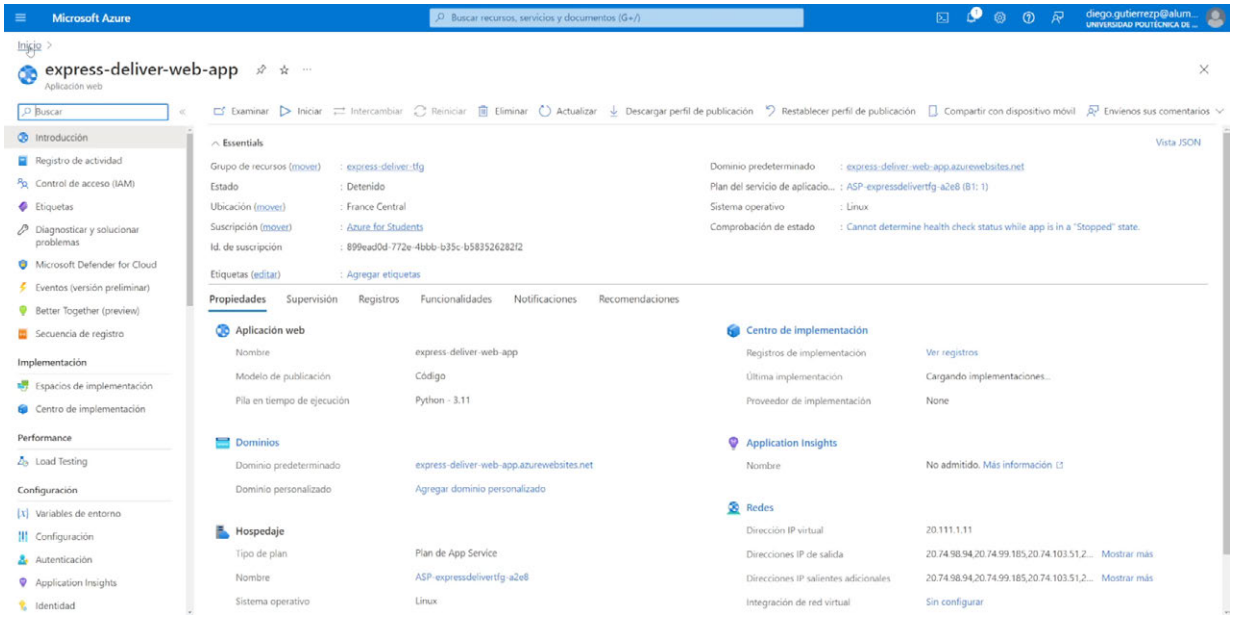


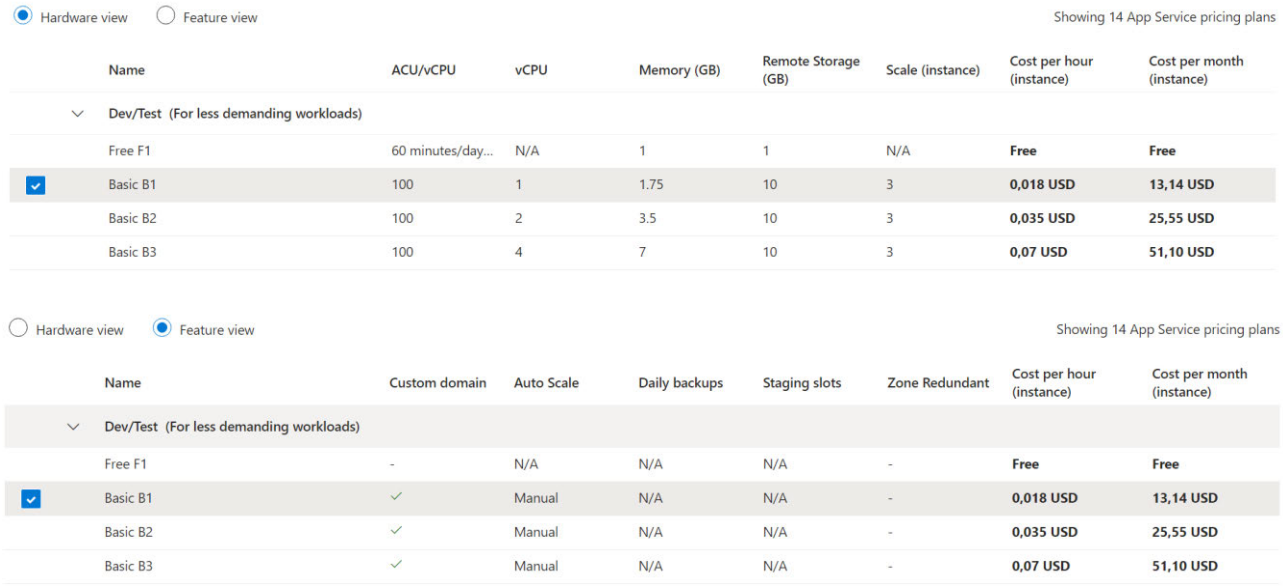
Ilustración 37. Componente Cloud Web App.

Tras iniciar la aplicación web desde Azure y la verificación de su funcionamiento se verifica el correcto funcionamiento y se da por concluido el desarrollo.

6. Presupuesto

El presupuesto incluye los siguientes ítems: desarrollo, diseño, despliegue, costes de licencia y mantenimiento. El desarrollo, diseño y despliegue incluyen el sueldo del ingeniero y los costes que conlleva para el proyecto. Al tratarse de un proyecto llevado a cabo por una consultoría o socio tecnológico un valor aproximado podría tratarse de 60 €/h.

Para los costes de licencia todo el proyecto se ha llevado a cabo utilizando herramientas y software gratuito. La mayor parte del coste monetario del proyecto es por tanto el coste del hospedaje en la nube para la realización de pruebas y el mantenimiento de la arquitectura Cloud que se puede clasificar como coste de línea base de licencia. Debido a los acuerdos de la UPM con la plataforma Cloud Microsoft Azure, el proyecto no ha conllevado ningún coste real para el estudiante, pero se detallarán los costes en caso de no disponer de un crédito.



Hardware view Feature view Showing 14 App Service pricing plans

	Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Scale (instance)	Cost per hour (instance)	Cost per month (instance)
	Dev/Test (For less demanding workloads)							
	Free F1	60 minutes/day...	N/A	1	1	N/A	**Free**	**Free**
☑	Basic B1	100	1	1.75	10	3	**0,018 USD**	**13,14 USD**
	Basic B2	100	2	3.5	10	3	**0,035 USD**	**25,55 USD**
	Basic B3	100	4	7	10	3	**0,07 USD**	**51,10 USD**

Hardware view Feature view Showing 14 App Service pricing plans

	Name	Custom domain	Auto Scale	Daily backups	Staging slots	Zone Redundant	Cost per hour (instance)	Cost per month (instance)
	Dev/Test (For less demanding workloads)							
	Free F1	-	N/A	N/A	N/A	-	**Free**	**Free**
☑	Basic B1	✓	Manual	N/A	N/A	-	**0,018 USD**	**13,14 USD**
	Basic B2	✓	Manual	N/A	N/A	-	**0,035 USD**	**25,55 USD**
	Basic B3	✓	Manual	N/A	N/A	-	**0,07 USD**	**51,10 USD**

Ilustración 38. Hardware and features review para el componente de aplicación web.

Para la base de datos se ha utilizado un plan gratuito de base de datos que ofrece 32GB de almacenamiento y 100,000 segundos de núcleos virtuales gratis.

Ilustración 39. Coste de mantenimiento del componente de base de datos.

El análisis de costes con los costes acumulados en total y la predicción anual de mantenimiento es el siguiente:

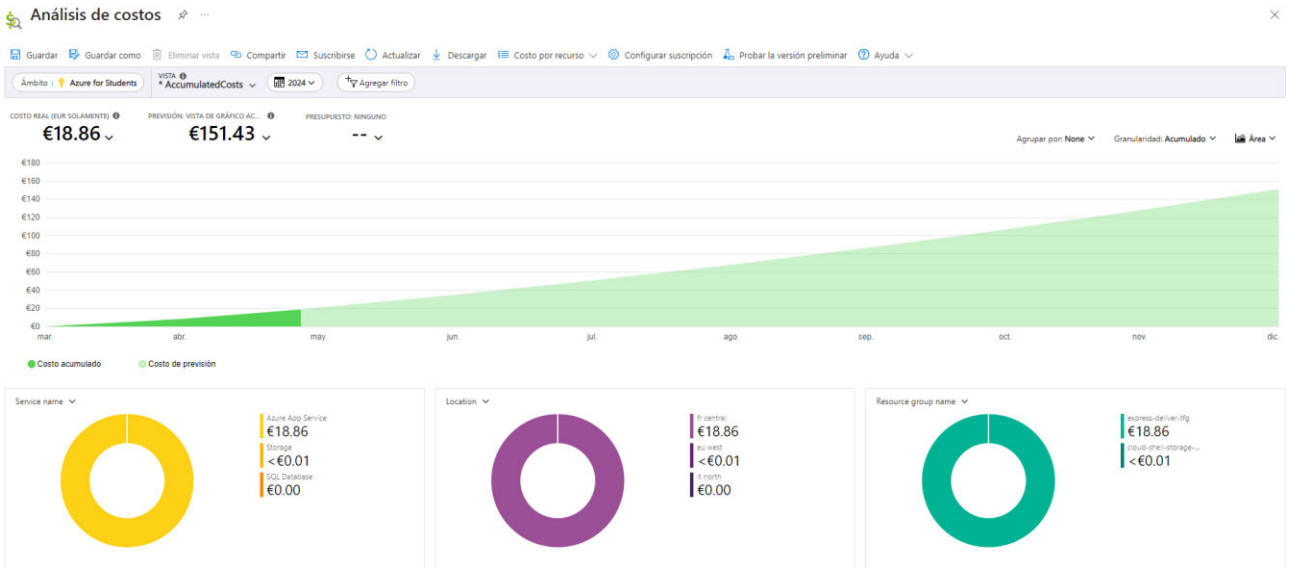


Ilustración 40. Análisis de costes y mantenimiento de la aplicación web

Se puede observar que la mayoría del presupuesto se lo lleva el contenedor del App Service que contiene la App Web. Para reducir costes podemos hacer un *downgrade* al componente del servidor web al plan gratuito.

En relación con el mantenimiento de la aplicación se podría tratar de un equipo que resolviese incidencias y podría conllevar un coste de 20 €/h.

7. Impacto del proyecto

7.1. Implicaciones y aportaciones

(i) Implicaciones Ambientales: Aunque el transporte de paquetes tiene un impacto ambiental, este proyecto puede contribuir a reducir dicho impacto mediante la optimización de rutas de entrega y la utilización de vehículos eficientes. Al mejorar la planificación logística y reducir el número de viajes necesarios, se puede disminuir la emisión de gases de efecto invernadero y el consumo de combustibles fósiles.

(ii) Implicaciones Económicas: Desde una perspectiva económica, una aplicación de mensajería exitosa puede generar oportunidades de empleo y contribuir al crecimiento económico. Al mejorar la eficiencia de los servicios de mensajería, las empresas pueden reducir costos operativos y aumentar la satisfacción del cliente.

(iii) Implicaciones Tecnológicas: Este proyecto destaca por la integración de tecnologías modernas que permiten una gestión eficiente y segura de la mensajería física. Este enfoque no solo demuestra la viabilidad técnica de la solución, sino que también sirve para contrastarse con el comúnmente utilizado MVC en Java que usan la gran mayoría de empresas.

(iv) Implicaciones Industriales: En el ámbito industrial, la aplicación puede ser utilizada para mejorar la gestión de la entrega de correspondencia y paquetes. Esto puede llevar a procesos más eficientes y a una mejor gestión de la logística, lo que a su vez puede impulsar la innovación y mejorar la competitividad.

7.2. Aportación a los Objetivos de Desarrollo Sostenible (ODS)

El proyecto contribuye a varios ODS, particularmente: [16]

ODS 9: Industria, Innovación e Infraestructura: Promueve la innovación y mejora la infraestructura logística al proporcionar una plataforma eficiente para la gestión de información en una aplicación web sin la necesidad de una infraestructura física.

ODS 8: Trabajo Decente y Crecimiento Económico: Facilita la creación de empleo en el sector logístico al crear una interfaz sencilla y evita la búsqueda de personal especializado. Además, mejora la eficiencia y productividad en diversos sectores económicos.

ODS 11: Ciudades y Comunidades Sostenibles: El sistema de entregas se puede optimizar, creando rutas de entrega y minimizando el tiempo en carretera lo que reduce las emisiones y contribuye a la creación de ciudades más sostenibles.

ODS 12: Producción y Consumo Responsables: La arquitectura en la nube promueve prácticas más sostenibles en la gestión de sistemas, reduciendo el uso de recursos y por tanto, las emisiones.

8. Conclusiones

8.1 Conclusiones

Tras la realización de este proyecto fin de carrera, se ha abordado de manera integral el problema propuesto relacionado con el desarrollo de un prototipo de aplicación web de mensajería hospedado en la nube. El problema inicial consistía en crear una plataforma de gestión de información eficiente, segura y escalable que pudiera satisfacer las necesidades administrativas de la empresa. Para resolverlo, se planteó una solución que involucraba el uso de tecnologías modernas como el lenguaje de programación Python con Flask para el *back-end*, SQLAlchemy para la gestión de la base de datos, y diversas herramientas de *front-end*.

A lo largo del desarrollo del proyecto, se integraron estas tecnologías y métodos, permitiendo alcanzar los objetivos establecidos. Los resultados obtenidos fueron satisfactorios, mostrando una plataforma funcional que permite a los usuarios gestionar y realizar un seguimiento de un modelo de información basado en correos, mensajeros y entregas. Además, se lograron implementar características adicionales como la autenticación de usuarios, la persistencia de datos y una interfaz de usuario responsiva.

El trabajo realizado valida la viabilidad del sistema planteado, ofreciendo una solución funcional con un modelo de infraestructura hospedada en la nube que aboga por la arquitectura Cloud contra el despliegue de infraestructura física.

De la misma forma, este proyecto me ha permitido derivar conocimiento nuevo respecto al desarrollo web y tecnologías Cloud que han permitido al estudiante verificar su conocimiento adquirido en la carrera y su capacidad de aprendizaje.

8.2. Trabajos futuros

Durante el desarrollo, se identificaron varios aspectos del problema que no se habían considerado inicialmente, como la necesidad de implementar sesiones de usuario y gestionar distintos permisos para distintos usuarios. Actualmente, en la aplicación solo existe un tipo de usuario que puede trabajar con todos los tipos de datos, pero para una versión final de la aplicación lo conveniente sería que cada uno de los mensajeros registrados se autenticasen y pudiesen ver sus entregas asignadas y, de esas entregas, actualizar tan solo su fecha de entrega y cambiar su estado a correo entregado.

También se podría incluir un buscador por características que permitiese filtrar cada tipo de dato utilizando uno de los campos, por ejemplo, para poder buscar correos destinados a cierto código postal o filtrar por entregas cuyo estado sea en reparto.

9. Referencias

[1] M. Richards, Software Architecture Patterns, O'Reilly, 2015.

[2] M. Fowler, Patterns of Enterprise Application Architecture, Martin Fowler, 2002.

[3] J. Geewax, API Design Patterns, Shelter Island NY: MANNING Shelter Island, 2021.

[4] «Flask,» [En línea]. Available: https://flask.palletsprojects.com/en/2.1.x/. [Último acceso: 23 Abril 2024].

[5] «Django,» [En línea]. Available: https://docs.djangoproject.com/en/stable/. [Último acceso: 23 Abril 2024].

[6] «SQLAlchemy,» [En línea]. Available: https://www.sqlalchemy.org/. [Último acceso: 23 Abril 2024].

[7] «Flask-sqlalchemy,» [En línea]. Available: https://flask-sqlalchemy.palletsprojects.com/en/2.x/. [Último acceso: 23 Abril 2024].

[8] I. NumFOCUS, «pandas,» [En línea]. Available: https://pandas.pydata.org/. [Último acceso: 3 junio 2024].

[9] J. B. y. A. G. R. Buyya, Cloud Computing: Principles and Paradigms, Wiley, 2011.

[10] Microsoft, «Microsoft Learn, AZ-900 Azure Fundamentals, Aspectos básicos de Microsoft Azure,» [En línea]. Available: https://learn.microsoft.com/es-es/training/courses/az-900t00. [Último acceso: 24 Abril 2024].

[11] M. Grinberg, Flask Web Development: Developing Web Applications with Python, O'Reilly Media, 2018.

[12] Datensen, «Luna Modeler,» [En línea]. Available: https://www.datensen.com/blog/database-design-tool-for-sql-server/. [Último acceso: 15 junio 2024].

[13] maxcountryman, «Flask-Login,» [En línea]. Available: https://flask-login.readthedocs.io/en/latest/. [Último acceso: 3 junio 2024].

[14] Pallets, «Jinja Documentation,» [En línea]. Available: https://jinja.palletsprojects.com/en/3.1.x/. [Último acceso: 15 junio 2024].

[15] Themesberg, «Volt Bootstrap5 Template,» [En línea]. Available: https://demo.themesberg.com/volt/. [Último acceso: 3 junio 2024].

[16] N. Unidas, «Objetivos de Desarrollo Sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Último acceso: 3 junio 2024].

10. Bibliografía

- W. S. Vincent, Django for Beginners Build websites with Python & Django, 2018 - 2020.
- M. Hartl, Ruby on Rails Tutorial: Learn Web Development with Rails, Addison-Wesley Professional, 2016.
- J. Z. w. E. Marcotte, Designing with web standards third edition, New Riders.