



**Universidad**  
Internacional  
de Valencia

# **Creación de una aplicación web con FastAPI y Angular para la gestión de un salón de belleza**

Titulación:  
Master Universitario en  
Desarrollo de  
Aplicaciones y Servicios  
Web

Segunda Curso  
Académico  
2022-2023

Alumno/a: Posada  
González, Francisco  
Javier  
DNI: 51756913J

Director/a del TFT:  
Rubén Pérez Ibáñez

Convocatoria:

SEGUNDA

De:

 Planeta Formación y Universidades

# Índice general

<b>Índice de figuras</b>	<b>2</b>
<b>Índice de tablas</b>	<b>2</b>
<b>1 Introducción</b>	<b>2</b>
1.1 Motivación del proyecto . . . . .	3
<b>2 Objetivos</b>	<b>4</b>
2.1 Requisitos funcionales . . . . .	4
2.2 Requisitos tecnológicos . . . . .	5
<b>3 Marco tecnológico</b>	<b>6</b>
3.1 Introducción a los sistemas ERP . . . . .	6
3.1.1 Clasificación de los ERPs . . . . .	7
3.2 Herramientas y frameworks . . . . .	9
3.2.1 Frameworks de Python para backend . . . . .	9
3.2.2 FastAPI . . . . .	10
3.2.3 Frameworks para el frontend . . . . .	11
3.2.4 Angular . . . . .	12
<b>4 Metodología</b>	<b>13</b>
4.1 Gestión del proyecto . . . . .	13
4.1.1 Historias de usuario . . . . .	14
4.1.2 Mapa de historias de usuario . . . . .	18
4.1.3 Sprints . . . . .	18
4.2 Modelos para la interfaz - Mockup . . . . .	19
4.3 Diseño BBDD - Diagrama ER . . . . .	21
<b>5 Resultados</b>	<b>22</b>
5.1 Vistas de la aplicación real . . . . .	22
5.2 Uso de la aplicación . . . . .	23
5.2.1 Login . . . . .	23
5.2.2 Añadir cliente . . . . .	24
5.2.3 Buscar cliente . . . . .	25
5.2.4 Ver información del cliente . . . . .	26
5.2.5 Crear servicio . . . . .	27
5.2.6 Consultar y editar servicio . . . . .	28
5.2.7 Crear cita . . . . .	29
5.2.8 Actualizar estado cita . . . . .	32
5.3 API con FastAPI . . . . .	33
5.3.1 Endpoints - Usuario . . . . .	33
5.3.2 Endpoints - Servicios . . . . .	33
5.3.3 Endpoints - Clientes . . . . .	34
5.3.4 Endpoints - Citas . . . . .	34

5.4 Aspectos destacables de programación . . . . .	34
5.4.1 Angular - Separación en módulos . . . . .	35
5.4.2 Angular - Lazy loading . . . . .	36
5.4.3 Búsqueda de cliente en tiempo real . . . . .	37
5.4.4 Autenticación mediante JWT . . . . .	38
<b>6 Conclusiones</b>	<b>41</b>
<b>7 Trabajos futuros</b>	<b>42</b>
<b>A Apéndice I: Diseño lógico BBDD</b>	<b>44</b>
<b>B Apéndice II: Diseño físico BBDD</b>	<b>46</b>
<b>Referencias</b>	<b>52</b>

# Índice de figuras

4.1	Mapa historias de usuario. Fuente: Propia . . . . .	18
4.2	Tareas del sprint 1. Fuente: Elaboración propia . . . . .	19
4.3	Modelo de agenda(vista principal). Fuente: Elaboración propia. . . . .	19
4.4	Modelo de clientes. Fuente: Elaboración propia. . . . .	20
4.5	Modelo de Servicios. Fuente: Elaboración propia. . . . .	20
4.6	Diagrama ER. Fuente: Elaboración propia. . . . .	21
5.1	Vista Agenda. Fuente: Elaboración propia . . . . .	22
5.2	Vista Clientes. Fuente: Elaboración propia . . . . .	23
5.3	Vista Agenda. Fuente: Elaboración propia . . . . .	23
5.4	Login. Fuente: Elaboración propia . . . . .	24
5.5	Botón añadir cliente. Fuente: Elaboración propia . . . . .	24
5.6	Formulario añadir cliente. Fuente: Elaboración propia . . . . .	25
5.7	Cliente ya existente. Fuente: Elaboración propia . . . . .	25
5.8	Botón buscar cliente. Fuente: Elaboración propia . . . . .	26
5.9	Resultado búsqueda para la palabra "mar". Fuente: Elaboración propia. 26	
5.10	Resultado búsqueda para la palabra "marc". Fuente: Elaboración propia. 26	
5.11	Información del cliente. Fuente: Elaboración propia . . . . .	27
5.12	Botón añadir nuevo servicio. Fuente: Elaboración propia . . . . .	27
5.13	Formulario nuevo servicio. Fuente: Elaboración propia . . . . .	28
5.14	Mensaje servicio añadido. Fuente: Elaboración propia . . . . .	28
5.15	Lista de servicios. Fuente: Elaboración propia . . . . .	28
5.16	Formulario modificar servicio. Fuente: Elaboración propia . . . . .	29
5.17	Mensaje servicio modificado. Fuente: Elaboración propia . . . . .	29
5.18	Sección de citas. Fuente: Elaboración propia . . . . .	29
5.19	Formulario crear cita. Fuente: Elaboración propia . . . . .	30
5.20	Búsqueda cliente nueva cita. Fuente: Elaboración propia . . . . .	31
5.21	Cita reservada. Fuente: Elaboración propia . . . . .	31
5.22	Cita no disponible. Fuente: Elaboración propia . . . . .	31
5.23	Sección citas actualizada. Fuente: Elaboración propia . . . . .	31
5.24	Cambiar estado cita. Fuente: Elaboración propia . . . . .	32
5.25	Mensaje de alerta al cancelar cita. Fuente: Elaboración propia . . . . .	32
5.26	Mensaje de alerta al finalizar cita. Fuente: Elaboración propia . . . . .	32

# Índice de tablas

3.1	Clasificación de algunos ERPs. . . . .	7
4.1	Login de usuario . . . . .	14
4.2	Crear clientes . . . . .	14
4.3	Consultar clientes . . . . .	14
4.4	Modificar clientes . . . . .	15
4.5	Crear servicios . . . . .	15
4.6	Consultar servicios . . . . .	15
4.7	Modificar servicios . . . . .	16
4.8	Borrar servicios . . . . .	16
4.9	Crear citas . . . . .	16
4.10	Consultar citas . . . . .	17
4.11	Borrar citas . . . . .	17

## Resumen

La gestión eficiente de las tareas que se llevan a cabo en un empresa, ya sea grande o pequeña, es esencial para el éxito de esta. En este contexto, se ha identificado la necesidad de una herramienta digital que facilite la gestión de un salón de belleza.

Este TFM se enfoca en el diseño y desarrollo de una aplicación web destinada a abordar dicha necesidad. Aunque esta aplicación comparte algunas similitudes con los Sistemas de Planificación de Recursos Empresariales (ERPs), su enfoque es diferente. En lugar de ser un ERP completo, esta herramienta está diseñada para proporcionar una solución simple y efectiva para la gestión integral de un salón de belleza.

La aplicación propuesta tiene como objetivo principal ofrecer una plataforma que permita a los propietarios y administradores realizar tareas como la programación de citas, el seguimiento de servicios prestados y la gestión de inventario. Además, se adaptará y se enriquecerá con nuevas funcionalidades a medida que surjan necesidades adicionales.

Para llevar a cabo este proyecto, se adoptará una metodología ágil, permitiendo una adaptación continua a las cambiantes necesidades del usuario final. Se utilizarán tecnologías modernas como Angular para el frontend y FastAPI para el backend, garantizando así un alto rendimiento y una experiencia de usuario atractiva.

**Palabras clave:** fastapi, angular, ERPs, gestión salón de belleza, digitalización, tecnologías web modernas

## Agradecimientos

**A mi pareja, Paola, por la ayuda, el tiempo y el apoyo que me ha dado durante todo este tiempo, gracias!**

A mi familia, que siempre está ahí apoyándome en todo lo que hago.

A Rubén Pérez, mi tutor en este TFM. Por su dedicación, sus consejos y el tiempo dedicado. Ha sido un placer!

Gracias a todos!

# 1. Introducción

La gestión de una empresa o pequeño negocio es una actividad crucial para asegurar el buen futuro de este. Incorporar herramientas que faciliten la gestión de los recursos de manera eficiente y mejoren los procesos y servicios prestados, contribuye al bienestar del negocio. Gracias a la evolución de la tecnología, estas herramientas pueden ser implementadas mediante una aplicación web.

Entre los diversos sectores que se benefician de esta evolución tecnológica se encuentra la industria de los salones de belleza, donde la eficiencia en la gestión y la satisfacción del cliente son elementos cruciales para el éxito. En este contexto, este Trabajo Fin de Máster (TFM) se enfoca en el diseño y desarrollo de una aplicación web que permita la gestión integral de un salón de belleza. Aunque esta aplicación comparte algunas características con los Sistemas de Planificación de Recursos Empresariales (ERPs, por sus siglas en inglés), es importante resaltar que su enfoque no se centra en ser un ERP completo, sino más bien en brindar una solución que permita la gestión del negocio de una manera sencilla.

La aplicación propuesta tiene como objetivo principal ofrecer una plataforma que permita a los propietarios y administradores realizar tareas como la programación de citas, el seguimiento de servicios prestados, la gestión de inventario y la comunicación con los clientes. La aplicación pretende abarcar funcionalidades básicas que posteriormente se irán ampliando a medida que vayan surgiendo nuevas necesidades.

Dentro del contexto de desarrollo de aplicaciones web, se empleará un enfoque tecnológico moderno y eficiente. Se hará uso de tecnologías como Angular para el frontend y FastAPI para el backend. Estas herramientas, reconocidas por su rendimiento y facilidad de uso, permitirán la creación de una interfaz de usuario atractiva y una base sólida para la gestión de datos y lógica de negocio.

En resumen, este Trabajo de Fin de Máster se propone como una contribución al campo de la gestión de salones de belleza a través de la tecnología. Al proporcionar una herramienta específica y enfocada en la gestión de estos establecimientos, se busca mejorar la eficiencia operativa y la experiencia tanto de los propietarios como de los clientes. A medida que avancemos en los siguientes capítulos, exploraremos más a fondo los detalles de esta aplicación web.



## 1.1 Motivación del proyecto

La motivación detrás de este proyecto surge a raíz de identificar la falta de una herramienta que permita la gestión de un negocio familiar (Salón de belleza) de manera sencilla. Después de varias reuniones y de estudiar algunas posibilidades en el mercado, se ha tomado la decisión de llevar a cabo una aplicación web que permita la gestión del negocio. Actualmente el negocio gestiona sus operaciones de manera manual, lo que conlleva una serie de limitaciones y dificultades.

La creación de una aplicación web especializada se presenta como la respuesta ideal a este desafío. Esta herramienta no solo automatizará y mejorará significativamente la gestión de citas, clientes y servicios, sino que también proporcionará al negocio una presencia sólida y necesaria en el mundo digital. En la era actual, donde la visibilidad en línea es esencial para atraer y retener clientes, esta iniciativa permitirá al salón de belleza ampliar su alcance y ofrecer servicios de manera más efectiva.

En un plano personal, este proyecto representa una oportunidad única para mi crecimiento y desarrollo profesional. La posibilidad de aprender y aplicar tecnologías modernas en el desarrollo de una aplicación web es enormemente gratificante. La experiencia de construir una solución desde cero y ver cómo se materializa en una herramienta funcional y útil para los demás es un gran impulso para mi aprendizaje. Al explorar estas tecnologías y aplicarlas en un contexto real, espero adquirir habilidades prácticas que me serán útiles en mi desarrollo profesional y en la creación de soluciones innovadoras.

## 2. Objetivos

El objetivo de este TFM es desarrollar y poner en funcionamiento una aplicación web que permita la gestión de un salón de belleza, optimizando procesos y mejorando la experiencia tanto para los propietarios como para los clientes.

Para llevar a cabo este objetivo, se han identificado los **Requisitos funcionales** (sección 2.1) y los **Requisitos tecnológicos** (sección 2.2), los cuales son detallados en dichos apartados.

Estos requisitos son el resultado de un primer análisis que se realizó con el objetivo de conseguir un producto mínimo viable. Esto quiere decir que existe la posibilidad que durante el desarrollo o en futuras etapas de la aplicación, surjan nuevos requisitos o se modifiquen los ya existentes.

### 2.1 Requisitos funcionales

La aplicación permitirá a los usuarios registrados acceder a sus cuentas a través de una interfaz que haga el inicio de sesión. La aplicación contará por ahora con tres funcionalidades principales:

- **Gestión de las citas:** La aplicación permitirá gestionar las citas mediante una interfaz que le permita crear, modificar, borrar y consultar las citas.
- **Gestión de los clientes:** La aplicación permitirá gestionar los clientes mediante una interfaz en la cual se visualice una ficha de clientes. En esta se mostrará información relevante del cliente, como por ejemplo: datos del cliente, histórico de citas, servicios prestados, etc. Por supuesto, la gestión del cliente también incluye las operaciones de crear, modificar, borrar y consultar los clientes.
- **Gestión de servicios:** La aplicación permitirá gestionar los servicios mediante una interfaz en la cual el usuario podrá crear, modificar, borrar y consultar los servicios.

## 2.2 Requisitos tecnológicos

La aplicación estará desarrollada con el framework de Angular para la parte del front y con FastAPI para la parte de la API. La persistencia de los datos se hará mediante la base de datos relacional PostgreSQL. En el [Capítulo 3 - Marco tecnológico](#) se explicarán los detalles de las tecnologías usadas.

Respecto a la seguridad, la aplicación deberá implementar protocolos y medios que garanticen la seguridad de los datos. Además de solicitar un usuario y contraseña, la aplicación hará uso del estándar JWT para mejorar de seguridad en cada petición que se haga a la API. Por otro lado, la contraseña será encriptada y almacenada en la base de datos de esta manera, es decir, no se podrán guardar contraseñas en texto plano. Como sistema de autorización, se usarán roles que limitarán tanto el acceso a los datos como a funcionalidades a usuarios con determinados roles.

La aplicación tendrá que tener una interfaz fácil de usar y que sea agradable para el usuario. Además, la interfaz se diseñara de manera que sea reponsive", para que se pueda acceder a la aplicación desde dispositivos como móviles o tablets.

## 3. Marco tecnológico

### 3.1 Introducción a los sistemas ERP

La aplicación que se va a llevar a cabo en este TFM está dentro del ámbito de los sistemas que ayudan a la gestión de una empresa. Normalmente estos sistemas suelen denominarse ERPs.

Un sistema ERP (Enterprise Resource Planning, por sus siglas en inglés) es un software integral que permite gestionar los diversos procesos de una empresa. Este tipo de sistemas ayuda a mejorar la eficiencia empresarial, la toma de decisiones, el servicio al cliente, la reducción de costes y, en general, a un crecimiento empresarial.

Las principales características de los sistemas ERP son las siguientes:

- **Integración:** Unificación de múltiples funciones en una sola plataforma que permite compartir datos entre diferentes áreas.
- **Automatización:** Automatizan procesos repetitivos que minimizan la necesidad de la intervención humana.
- **Acceso centralizado:** Permiten acceder a la información desde cualquier ubicación y dispositivo autorizado, facilitando la colaboración y el trabajo remoto.
- **Seguridad:** Implementan medidas de seguridad robustas que protegen los datos empresariales.
- **Atención al cliente:** Permiten gestionar las relaciones con los clientes, como la administración de contactos y el historial de compras.
- **Gestión financiera:** Ofrecen herramientas para llevar un registro de las finanzas.
- **Gestión de inventario y logística:** Ayudan a supervisar los niveles de inventario, optimizar la cadena de suministro y garantizar la disponibilidad de productos.

Los sistemas ERPs suelen ser sistemas bastante amplios y, en ocasiones, bastante complejos, en los cuales se pueden encontrar características como la centralización de datos, flujos de trabajo definidos, información en tiempo real, gestión de los recursos humanos, planificación de la producción, etc.

### 3.1.1 Clasificación de los ERPs

Hay una gran cantidad de sistemas ERPs en el mercado, debido a que son sistemas muy usados por las empresas y, que además, suelen adaptarse o encajar en el propio mercado al que pertenece la empresa, existe una gran variedad que se pueden clasificar en base a los siguiente criterios: Tamaño de la empresa, Tipo de instalación, Licencia, Arquitectura y Grado de especialización.

A continuación se presenta una tabla con la clasificación de los ERPs mas usados en base a los criterios mencionados anteriormente.

Criterio	Tipo	ERP
Tamaño de la Empresa	Pequeñas y Medianas Empresas (PYMES)	Odoo
	Grandes Empresas y Multinacionales	SAP S/4HANA
Tipo de Instalación	ERP en la Nube (Cloud ERP)	NetSuite
	ERP On-Premises (Local)	Microsoft Dynamics AX
Licencia	ERP de Código Abierto	OpenERP (Odoo)
	ERP Propietario o Comercial	Oracle ERP Cloud
Arquitectura	Arquitectura Monolítica	SAP ERP
	Arquitectura Modular	Microsoft Dynamics 365
Grado de Especialización	ERP Generalista	Microsoft Dynamics NAV
	ERP Específico por Industria	Epicor ERP

Tabla 3.1: Clasificación de algunos ERPs.

La tabla anterior sirve para clasificar los ERPs y aporta ideas sobre los criterios que se pueden tener en cuenta a la hora de elegir un sistema de este tipo.

Otros puntos importante a la hora de elegir un ERP son los siguientes:

- **Costos:** Es importante analizar los costos continuos de implementación, personalización, capacitación y mantenimiento.
- **Integración:** Evaluar la capacidad del ERP para integrarse con otros sistemas y aplicaciones existentes en la empresa.
- **Flexibilidad y Personalización:** Determinar si el ERP puede adaptarse a los procesos específicos de la empresa, y si es posible personalizarlo según las necesidades.

- **Usabilidad:** Evaluar la facilidad de uso del sistema para los empleados de la empresa, considerando su experiencia y nivel de habilidad técnica.
- **Tecnología y Actualizaciones:** Evaluar la tecnología subyacente del ERP y la frecuencia de las actualizaciones para asegurarse de que esté alineada con las necesidades tecnológicas actuales y futuras.

En este TFM, no se busca llevar a cabo una implementación completa de un ERP. El objetivo principal es utilizar las características clave de los ERPs como referencia para desarrollar una aplicación web diseñada para la gestión eficiente de un salón de belleza.

A raíz de este análisis, han surgido diversas ideas que se planean implementar en esta aplicación. Estas incluyen la **gestión de ventas y clientes**, una característica común en todos los ERPs. Además, se considera la posibilidad de incorporar la **personalización**, una característica destacada en ERPs como Odoo. La **accesibilidad** que ofrecen Oracle ERP Cloud y NetSuite, al ser sistemas basados en la nube, también es un punto clave a considerar. Por último, se pretende abordar la **especialización** que caracteriza a Epicor ERP, ya que se enfoca en un sector específico.

La decisión de llevar a cabo esta aplicación web y no usar una comercial se basa en los siguientes motivos:

- El costo o licenciamiento de un ERP una empresa en crecimiento puede llegar a ser alto.
- Desarrollar una aplicación web personalizada permite adaptar la herramienta a la necesidades de la empresa.
- Se cuentan con conocimientos en desarrollo de software que permiten llevar a cabo este tipo de proyecto.

## 3.2 Herramientas y frameworks

Este proyecto implica la creación de una aplicación web moderna, desarrollada con enfoques dinámicos, interactivos y centrados en el usuario. Mediante la combinación de tecnologías innovadoras, se generan aplicaciones que van más allá de la mera entrega de información, al ofrecer interfaces interactivas que brindan a los usuarios una experiencia de uso fluida y enriquecedora.

Entre las numerosas características que pueden definir a una aplicación web moderna, podrían destacarse las siguientes:

- **Arquitectura SPA (Single Page Application):** Esta arquitectura, implementada mediante frameworks como React, Angular o Vue.js, permite una carga inicial única y actualizaciones dinámicas según las acciones del usuario. Esto resulta en una experiencia más fluida y rápida.
- **Comunicación a través de API RESTful:** La comunicación con un backend mediante una API REST establece una clara separación entre el frontend y el backend. Esto facilita el desarrollo en paralelo y la reutilización de servicios. (Fielding y Taylor, 2000) (Kumar y cols., 2023)
- **Seguridad en las Comunicaciones:** Implementar medidas de seguridad en las comunicaciones resulta fundamental. Esto se logra a través de técnicas de autenticación y autorización seguras, como los JSON Web Tokens (JWT) y el protocolo OAuth.
- **Diseño Responsivo:** Dado que el acceso a las aplicaciones web proviene no solo de dispositivos portátiles y PCs, sino también de dispositivos móviles, es crucial considerar el diseño responsivo. Esto asegura que la web se adapte adecuadamente a cualquier tipo de dispositivo.

Durante el desarrollo de este TFM se ha hecho un estudio de los posibles frameworks, tanto de frontend como de backend, con los que se puede llevar a cabo esta aplicación.

### 3.2.1 Frameworks de Python para backend

Para el backend se ha considerado usar el lenguaje Python debido a que es un lenguaje moderno, fácil de usar, versátil y cuenta con numerosos frameworks para el backend. A continuación se analizan las características de algunos de ellos como son Django, Flask y FastAPI.

Django es el framework de Python más popular, diseñado para generar directamente HTML en el backend, se puede considerar un framework full-stack. Está más orientado a la creación de aplicaciones web complejas y se basa en el patrón de diseño MVC. Tiene una curva de aprendizaje pronunciada y características como la

generación automática de documentación o la inyección de dependencias no están incluidas por defecto. Django (s.f.)

Flask es un "microframework" que permite crear aplicaciones rápidamente. Es rápido de aprender y comúnmente utilizado para aplicaciones pequeñas o que no requieren una base de datos o gestión de usuarios. (Flask, s.f.)

FastAPI es un framework orientado a la creación de APIs. Es relativamente nuevo en comparación con Flask (2010) o Django (2005). Sus principales características son la facilidad que ofrece para crear una API en poco tiempo, la documentación automática, la validación de datos y el buen rendimiento que ofrece. (FastAPI, s.f.)

En este TFM se ha considerado implementar el backend con FastAPI debido a las características que tiene y que se detallan en la sección **3.2.2 - FastAPI**.

### 3.2.2 FastAPI

FastAPI (FastAPI, s.f.) es un web framework moderno y rápido para construir APIs con Python 3.6+ basado en las anotaciones de tipos estándar (Type Hints) de Python.

Fue diseñado para facilitar la creación rápida de aplicaciones web y servicios API de manera eficiente con una sintaxis clara y expresiva.

Algunas de las principales características de FastAPI son:

- **Basado en estándares abiertos:** Uso del estándar OpenAPI (OpenAPI, s.f.) para la creación de APIs y JSON Schema (JSON, s.f.) para la documentación automática del modelo de datos.
- **Documentación automática:** Esta es una de las características que hace a FastAPI un framework tan potente y bien recibido por la comunidad. La documentación es **generada automáticamente** gracias a la especificación Swagger (Swagger, s.f.) y a Swagger UI. Además, dispone de una documentación alternativa generada por ReDoc (ReDoc, s.f.). La documentación ayuda a que los desarrolladores y los usuarios comprendan rápidamente cómo interactuar con la API.
- **Validación de datos:** La combinación de tipo de datos en Python (type hints, a partir de Python 3.6+) y de **Pydantic** (Pydantic, s.f.), permite llevar a cabo la validación de los datos de entrada, generando mensajes de error automáticos que son devueltos al cliente.
- **Seguridad y autenticación:** Proporciona mecanismos de seguridad integrados, como autenticación OAuth2 y manejo de tokens JWT (JSON Web Tokens), para garantizar la seguridad de las API.
- **Inyección de dependencias:** Permite la inyección de dependencias para gestionar la lógica de la aplicación de manera modular y clara. Todas las depen-



dencias son manejadas automáticamente por el framework.

- **Starlette y Pydantic:** Son dos de los principales componentes de FastAPI, ambos desempeñan un papel fundamental en las características y funcionalidades de FastAPI.
  - **Starlette:** es un servidor web asíncrono que maneja el enrutamiento y las solicitudes a la aplicación. Tiene soporte para WebSockets, capacidad para incorporar middlewares personalizados y una estructura para la gestión de errores. Todas estas capacidades hacen que Starlette sea fundamental para la gestión de solicitudes concurrentes y la comunicación bidireccional en tiempo real.
  - **Pydantic:** es una biblioteca que se utiliza para definir modelos de datos, validar la entrada y salida de datos automáticamente, y transformar datos entre representaciones Python y JSON. Con la capacidad de definir modelos utilizando tipos de Python nativos y anotaciones, Pydantic facilita la creación de objetos estructurados y su validación. Además, su uso es de gran ayuda en la generación de documentación interactiva, ya que muestra información sobre los modelos y los parámetros de rutas.

En resumen, FastAPI tiene características que lo convierten en un framework ideal para crear APIs de manera rápida. Estas APIs pueden ser utilizadas en conjunto con frameworks de frontend modernos, como Angular, React o Vue.js. A pesar de ser un framework relativamente nuevo, está siendo utilizado por empresas privadas como Netflix (Kevin Glisson, 2020) o Uber (Piero Molino, 2019), y se pueden encontrar menciones a él en artículos científicos contemporáneos (Bansal y Ouda, 2022)

### 3.2.3 Frameworks para el frontend

Angular, React y Vue son tres de los frameworks y bibliotecas más populares para el desarrollo de aplicaciones web modernas. Cada uno tiene sus propias ventajas y enfoques que los hacen adecuados para diferentes tipos de proyectos.

Angular ofrece un enfoque en arquitectura completa basada en componentes, servicios, módulos y enrutamiento. Utiliza TypeScript para mayor seguridad y mantenibilidad. Tiene un alto rendimiento que lo hace idóneo para aplicaciones complejas y una gran comunidad activa con amplios recursos disponibles. Está respaldado por Google.

React se centra en ser una biblioteca de vistas que se puede combinar con otras bibliotecas o herramientas según sea necesario. Es muy eficiente gracias a su modelo de representación virtual (Virtual DOM). Esto lo hace adecuado para aplicaciones con interfaces de usuario altamente dinámicas y cambiantes. React es respaldado por Facebook y tiene una comunidad extremadamente grande y activa, lo que garantiza una amplia gama de recursos y bibliotecas disponibles.

Vue Ofrece una arquitectura progresiva que permite adoptar solo las partes necesarias. Fácil de aprender, adecuado para proyectos medianos o pequeños. Buen rendimiento gracias a su sistema de reactividad y Virtual DOM. Comunidad en crecimiento con bibliotecas útiles disponibles.

En este TFM se ha decidido implementar el front con Angular debido a las características que tiene (que se detallan más en la sección **3.2.4 - Angular**) y porque se tiene experiencia trabajando con el.

### **3.2.4 Angular**

Angular es una plataforma y un framework para crear aplicaciones de una sola página (SPA) en el lado del cliente usando HTML y TypeScript. Desarrollado en TypeScript, este framework de código abierto es mantenido por Google.

Angular sigue el patrón MVC que ayuda a separar la lógica de negocio, la representación visual y la interacción del usuario en componentes individuales.

La arquitectura de Angular se basa en módulos, componentes, servicios, enrutamiento, entre otros. Al aprovechar este conjunto de herramientas y patrones, es posible crear aplicaciones con un alto nivel de calidad y rendimiento. Además, al adoptar un enfoque modular, facilitando la separación de preocupaciones y fomentando la creación de componentes reutilizables, se brinda a los desarrolladores la posibilidad de construir aplicaciones escalables y mantenibles.

En el marco de este proyecto, se desarrollará la parte frontal haciendo uso del framework Angular.

## 4. Metodología

La metodología seleccionada para abordar este Trabajo de Fin de Máster (TFM) es una metodología ágil. Esta elección se fundamenta en la necesidad de desarrollar el proyecto de manera eficiente, flexible y orientada a resultados concretos.

La metodología ágil se destaca por su enfoque iterativo e incremental, permitiendo dividir el proyecto en etapas o iteraciones manejables. Cada iteración se enfoca en un conjunto específico de tareas y objetivos, lo que ayuda a un progreso constante a una adaptación rápida a los cambios. En esta metodología se promueve la colaboración estrecha con el cliente y permite hacer ajustes en función de las necesidades cambiantes.

La elección de una metodología ágil viene siendo práctica habitual en proyectos de desarrollo de software como este debido al buen resultado que aportan a la gestión del proyecto. En este tipo de proyectos es importante tener una buena comunicación, saber adaptarse a los cambios y mantener una entrega continua de funcionalidades. Todas estas características se pueden lograr adoptando una metodología ágil.

A continuación se detalla la gestión del proyecto y las herramientas empleadas para ello.

### 4.1 Gestión del proyecto

Para la gestión del proyecto, se han utilizado herramientas basadas en la metodología Scrum. A partir de los requisitos del proyecto, se construyó un **Product Backlog** que incluye las **Historias de Usuario**. Luego, se generaron diferentes **Sprints** con las tareas que debían llevarse a cabo. La consecución exitosa de estos Sprints dará como resultado la aplicación web.

Las Historias de Usuario representan necesidades o requerimientos específicos desde la perspectiva del cliente o usuario final. En lugar de describir una lista exhaustiva de especificaciones técnicas, las Historias de Usuario se centran en los objetivos y expectativas de quienes utilizarán la aplicación.

En la sección **4.1.1 - Historias de usuario**, se detallan las Historias de Usuario que servirán como hoja de ruta para la creación de las funcionalidades requeridas por los usuarios.

### 4.1.1 Historias de usuario

En base a los requisitos de la aplicación se han identificado las siguientes historias de usuario:

Login de usuario			
<b>Descripción:</b> Como usuario quiero acceder a la aplicación a través de una interfaz en la que tenga que introducir el usuario y la contraseña			
<b>Validación:</b> Acceder con usuario y contraseña. Denegar acceso cuando el usuario o la contraseña sean incorrectos			
ID	Prioridad	Estimación	Dependencia
1	1	14h	-

Tabla 4.1: Login de usuario

Crear clientes			
<b>Descripción:</b> Como usuario, quiero poder crear un cliente a través de una interfaz			
<b>Validación:</b> Crear un cliente, consultar los datos del cliente recién creado			
ID	Prioridad	Estimación	Dependencia
2	1	10h	-

Tabla 4.2: Crear clientes

Consultar clientes			
<b>Descripción:</b> Como usuario, quiero poder consultar los clientes a partir de algún dato del cliente			
<b>Validación:</b> Obtener los datos de un cliente a través de un dato. Obtener el cliente a partir del listado de clientes			
ID	Prioridad	Estimación	Dependencia
3	1	10h	2

Tabla 4.3: Consultar clientes

Modificar clientes			
<b>Descripción:</b> Como usuario, quiero poder acceder a los datos del cliente y modificarlos			
<b>Validación:</b> Obtener los datos del cliente, modificarlos y consultar los datos actualizados			
ID	Prioridad	Estimación	Dependencia
4	1	10h	2

Tabla 4.4: Modificar clientes

Crear servicios			
<b>Descripción:</b> Como usuario, quiero poder crear un servicio a través de una interfaz			
<b>Validación:</b> Crear un servicio, consultar los datos del servicio recién creado			
ID	Prioridad	Estimación	Dependencia
6	2	10h	-

Tabla 4.5: Crear servicios

Consultar servicios			
<b>Descripción:</b> Como usuario, quiero poder consultar los servicios a partir de algún dato del servicio			
<b>Validación:</b> Obtener los datos de un servicio a través de un dato. Obtener el servicio a partir del listado de servicios			
ID	Prioridad	Estimación	Dependencia
7	2	10h	7

Tabla 4.6: Consultar servicios

Modificar servicios			
<b>Descripción:</b> Como usuario, quiero poder acceder a los datos del servicio y modificarlos			
<b>Validación:</b> Obtener los datos del servicio, modificarlos y consultar los datos actualizados			
ID	Prioridad	Estimación	Dependencia
8	2	10h	7

Tabla 4.7: Modificar servicios

Borrar servicios			
<b>Descripción:</b> Como usuario, quiero poder borrar un servicio			
<b>Validación:</b> Obtener el servicio a borrar. Visualizar que en el listado de servicios ya no aparece			
ID	Prioridad	Estimación	Dependencia
9	2	10h	7

Tabla 4.8: Borrar servicios

Crear citas			
<b>Descripción:</b> Como usuario, quiero poder crear una cita para un cliente			
<b>Validación:</b> Crear una cita. Asociar la cita a un cliente. Asociar un servicio a un cliente			
ID	Prioridad	Estimación	Dependencia
10	1	10h	2,3,6,7

Tabla 4.9: Crear citas

Consultar citas			
<b>Descripción:</b> Como usuario, quiero poder consultar una cita			
<b>Validación:</b> Obtener los datos de la cita a partir de un dato de la cita. Obtener los datos de la cita a partir de un listado de citas			
ID	Prioridad	Estimación	Dependencia
11	1	10h	2,3,6,7,10

Tabla 4.10: Consultar citas

Borra citas			
<b>Descripción:</b> Como usuario, quiero poder borrar una cita			
<b>Validación:</b> Obtener la cita a borrar. Visualizar que en el listado de citas ya no aparece			
ID	Prioridad	Estimación	Dependencia
13	1	10h	2,3,6,7,10

Tabla 4.11: Borrar citas

## 4.1.2 Mapa de historias de usuario

Con las historias de usuario se ha creado un mapa de historias de usuario donde se reflejan las tareas que implica cada una de ellas.

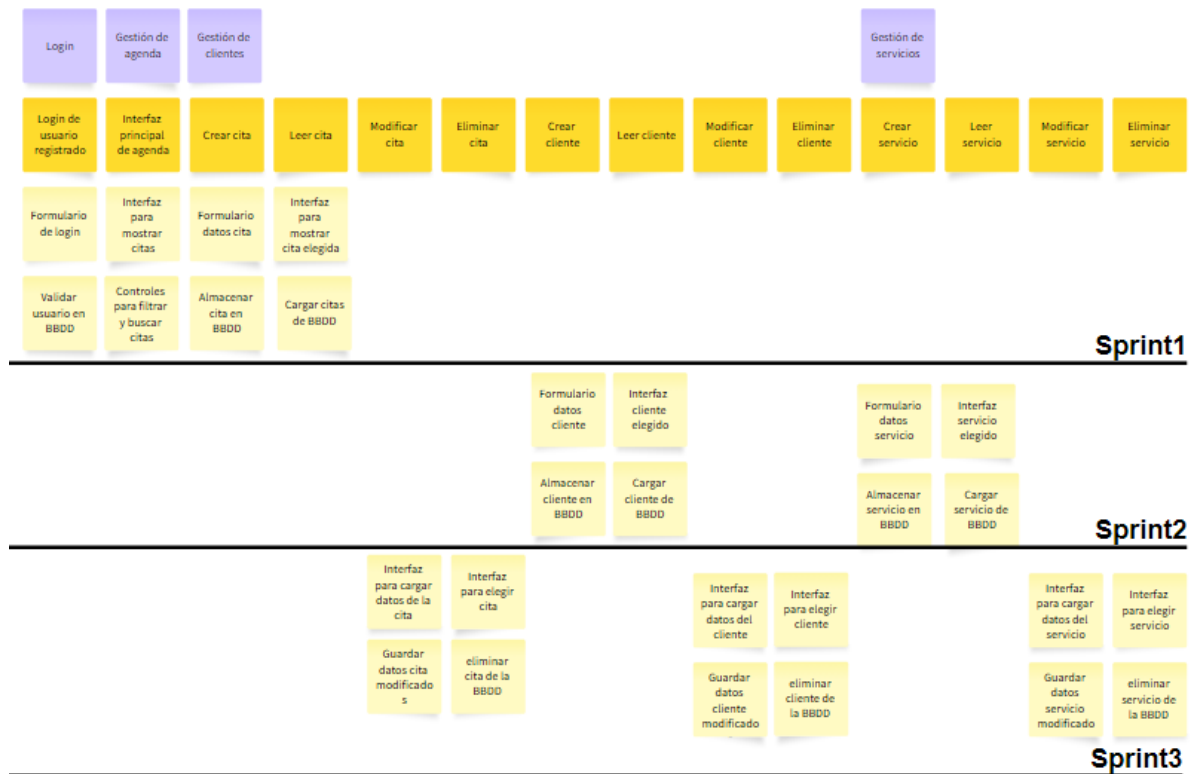


Figura 4.1: Mapa historias de usuario. Fuente: Propia

## 4.1.3 Sprints

A partir del mapa de historias de usuario se han generado 3 Sprints para llevar a cabo todas las tareas.

- **Sprint 1:** Se centrará en el login de usuarios registrados y en tener la interfaz para gestionar las citas (crearlas y listarlas)
- **Sprint 2:** Se llevarán a cabo las tareas de crear y listar tanto Clientes como Servicios.
- **Sprint 3:** Ultimo sprint en el que se podrán modificar y eliminar Citas, Clientes y Servicios.

Al terminar cada uno de estos sprints se tendrá una versión de la aplicación que el cliente podrá probar. La idea es que a medida que la aplicación se va construyendo,



el cliente pueda ir probándola para obtener feedback y poder corregir o cambiar funcionalidades.

En la **Figura 4.2** se puede ver el detalle del sprint1:



Figura 4.2: Tareas del sprint 1. Fuente: Elaboración propia

## 4.2 Modelos para la interfaz - Mockup

Para el diseño de la interfaz se ha hecho un modelado de la estructura con las vistas con las vistas principales de la aplicación

La vista principal está formada por un barra de navegación mediante la cual se puede acceder a la Agenda, los clientes y los servicios, como se puede ver en la figura 4.3

La sección de agenda será el "home" de la aplicación. En ella se podrá visualizar las citas asignadas a cada empleado y los periodos de tiempo libres por día seleccionado en el calendario.



Figura 4.3: Modelo de agenda(vista principal). Fuente: Elaboración propia.

Para los clientes, se plantea una vista en la que se puedan ver todos los datos relacionados con el, como puede ser los datos principales, las próximas citas, y un historial de citas/servicios. Además, se cuentan con controles que permiten buscar o añadir clientes.



Logo    Agenda    **Cientes**    Servicios    User

---

Buscar    Añadir

**Datos**

Nombre    Telefono    Email    Fecha nacimiento

**Proximas citas**

Fecha    Hora    Servicios

**Servicios**

Fecha    Descripción servicio

Figura 4.4: Modelo de clientes. Fuente: Elaboración propia.

La sección de servicios consta de una lista con todos los servicios que se ofrecen y se da la posibilidad de añadir nuevos servicios y modificar los ya existentes



Logo    Agenda    Clientes    **Servicios**    User

---

Añadir

Servicio	Precio	
Servicio A	€€	Editar
Servicio B	€€	Editar
Servicio C	€€	Editar
Servicio D	€€	Editar

Figura 4.5: Modelo de Servicios. Fuente: Elaboración propia.

## 4.3 Diseño BBDD - Diagrama ER

La información que se genera y se almacena en la aplicación se ha modelado mediante una base de datos relacional. En base a los requerimientos de la aplicación se han identificado las entidades con sus atributos y la relaciones entre estas. Con esos componentes se ha construido el diagrama entidad-relación de la [Figura 4.6](#)

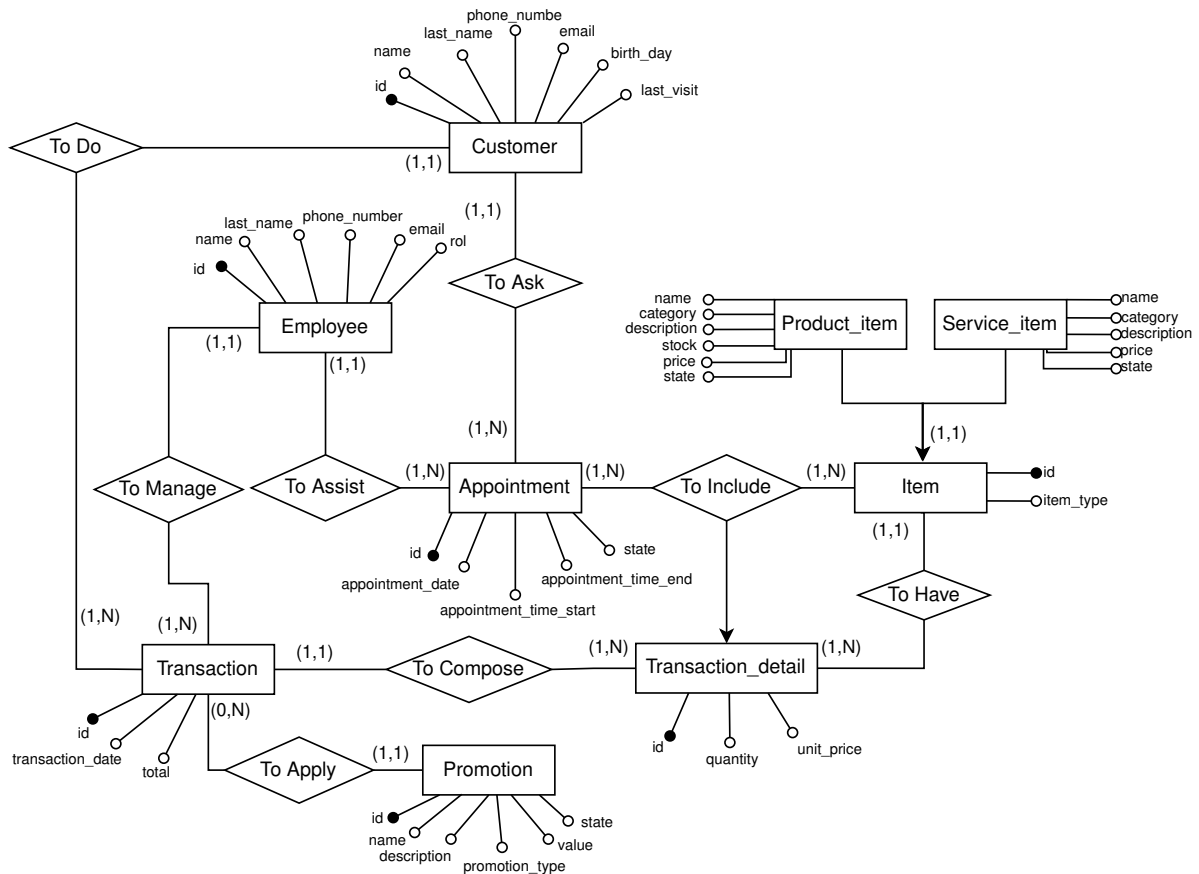


Figura 4.6: Diagrama ER. Fuente: Elaboración propia.

En este diagrama entidad-relación están incluidas las entidades que forman parte de las funcionalidades actuales de la aplicación y también están incluidas entidades que se usarán en funcionalidades futuras. Algunas de estas funcionalidades son la de registrar las transacciones, obtener el detalle de estas o aplicar promociones.

Las otras fases del diseño de la base de datos se detallan en el [Apéndice I: Diseño lógico BBDD](#) y en el [Apéndice II: Diseño físico BBDD](#). El sistema de gestión de base de datos que se ha usado es para implementar esta base de datos relacional es PostgreSQL.

## 5. Resultados

En este capítulo se exponen los resultados obtenidos durante el desarrollo de la aplicación. En el capítulo se muestran y se comentan las vistas reales de la aplicación, como se usa la aplicación y los aspectos más destacables de la programación.

### 5.1 Vistas de la aplicación real

Los modelos de las vistas de la [sección 4.2 - Modelos para la interfaz - Mockup](#) sirven como referencia tanto para el desarrollador como para el cliente, ya que se consigue una primera aproximación de lo que será la aplicación a nivel visual.

A continuación se presentan las vistas reales de la aplicación, las cuales han variado poco respecto a los modelos iniciales. También se explicará a grandes rasgos las funcionalidades principales de estas vistas. El detalle de todas ellas se verá en la [sección 5.2 - Uso de la aplicación](#).

En la vista de la agenda, se ha replicado prácticamente el modelo inicial. La vista consta de los filtros de los empleados, la sección donde se muestran las citas y los huecos libres, y el calendario. Las citas que se muestran son las que coinciden para ese día y ese empleado.

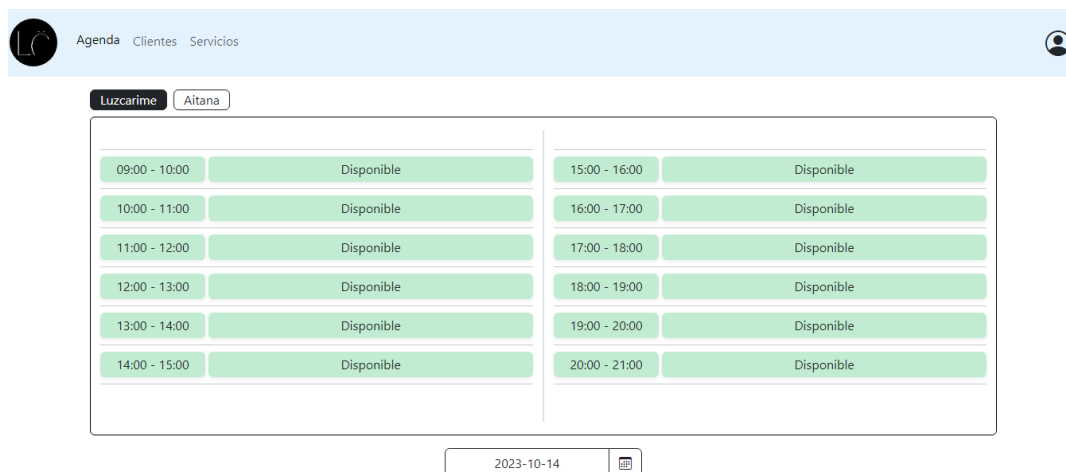


Figura 5.1: Vista Agenda. Fuente: Elaboración propia

En la vista de los clientes se muestran primero dos tarjetas para que el usuario pueda elegir entre buscar a un cliente o añadir un nuevo cliente

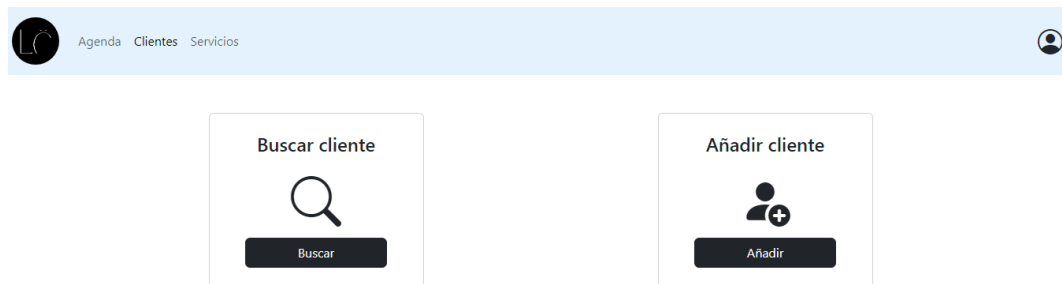


Figura 5.2: Vista Clientes. Fuente: Elaboración propia

La vista de servicios consta de un botón que permite añadir servicios nuevos y una lista con los servicios ya existentes. En cada elemento de la lista hay un botón que permite acceder al servicio para consultar más información y para modificar este si así se desea.



Figura 5.3: Vista Agenda. Fuente: Elaboración propia

## 5.2 Uso de la aplicación

EL uso de la aplicación y las funcionalidades que esta ofrece, están basadas en los requisitos funcionales que se tomaron al inicio del proyecto. Estas primeras funcionalidades sirven como base para futuras funcionalidades o mejoras que se quieran implantar en la aplicación.

### 5.2.1 Login

El login permite el acceso a la aplicación después de que el usuario introduzca un correo y una contraseña válidos. Una vez hecho el login, se carga la vista de la agenda, que actúa como vista principal y de llegada a la aplicación (Figura 5.3).

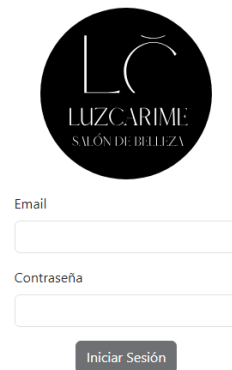


Figura 5.4: Login. Fuente: Elaboración propia

### 5.2.2 Añadir cliente

Para añadir un nuevo cliente, se accede primero a la vista de cliente desde la barra de navegación. En esta vista encontramos el botón añadir cliente (**Figura 5.5 - Botón añadir cliente. Fuente: Elaboración propia**)

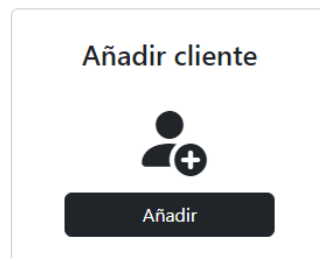



Figura 5.5: Botón añadir cliente. Fuente: Elaboración propia

Después de pulsar el botón Añadir, se accede al formulario que permite introducir los datos del cliente. Este formulario solo tiene dos campos obligatorios, el nombre y el número de teléfono. Los demás campos sirven como información para identificar al cliente pero no son obligatorios. Una vez rellenado el formulario y después de darle al botón de enviar, se indica mediante un mensaje que el usuario ha sido añadido correctamente.

## Añadir nuevo cliente

Nombre	Apellido
<input type="text" value="María"/>	<input type="text" value="López"/>
Alias	Teléfono
<input type="text" value="Vecina"/>	<input type="text" value="612612612"/>
Correo Electrónico	Fecha de Nacimiento
<input type="text" value="maria@example.com"/>	<input type="text" value="20/04/1985"/> 

¡Cliente registrado!

Figura 5.6: Formulario añadir cliente. Fuente: Elaboración propia

El número de teléfono de cliente se usa como dato para identificarlo, por tanto solo puede un número de teléfono por cliente. Si al introducir los datos se detecta que ese número ya existe, se muestra un mensaje indicándolo.

## Añadir nuevo cliente

Nombre	Apellido
<input type="text" value="María"/>	<input type="text" value="López"/>
Alias	Teléfono
<input type="text"/>	<input type="text" value="612612612"/>
Correo Electrónico	Fecha de Nacimiento
<input type="text"/>	<input type="text" value="dd/mm/aaaa"/> 

Cliente ya existente con ese número de teléfono

Figura 5.7: Cliente ya existente. Fuente: Elaboración propia

### 5.2.3 Buscar cliente

Se puede buscar un cliente para consultar su información. Para ello, se tiene que acceder a la sección de clientes desde la barra de navegación. Una vez dentro, se

pulsa sobre el botón Buscar.

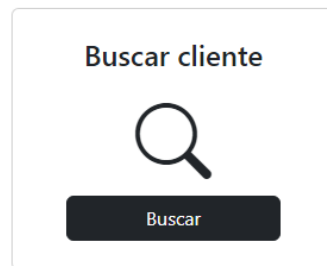


Figura 5.8: Botón buscar cliente. Fuente: Elaboración propia

Después de pulsar el botón se accede a la vista de búsqueda de clientes. Esta vista se compone de un campo para introducir el nombre del cliente. A medida que se van introduciendo letras se va mostrando una lista con los resultados encontrados.

La búsqueda que se ha implementado aquí es la que se conoce como "búsqueda con comodín" ó "búsqueda wildcard"

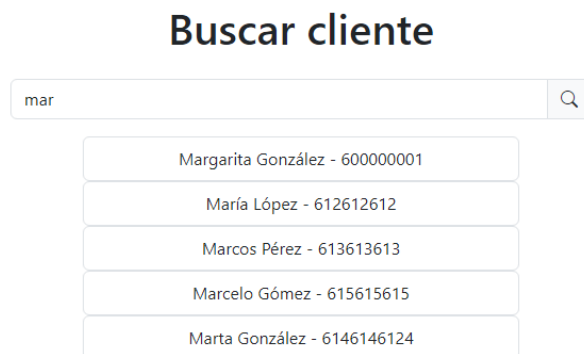


Figura 5.9: Resultado búsqueda para la palabra "mar". Fuente: Elaboración propia.

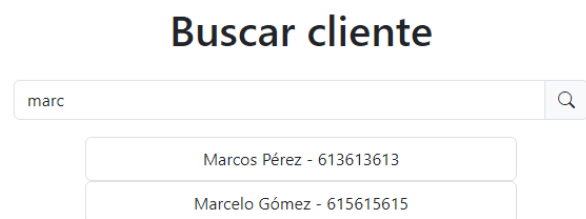


Figura 5.10: Resultado búsqueda para la palabra "marc". Fuente: Elaboración propia.

Al pulsar sobre uno de los elementos de la lista, se accede a la información de cliente que se explica en el punto (5.2.4 - [Ver información del cliente](#))

## 5.2.4 Ver información del cliente

En la vista de información del cliente se muestran las secciones de Datos de cliente, Próximas citas e Historial de citas/servicios. Con los datos incluidos en estas secciones se puede acceder rápidamente a toda la información relacionada con un cliente.



## Datos del cliente

**Nombre:** María López   **Tel:** 612612612  
**Email:** maria@example.com   **Alias:** Vecina

## Próximas citas

No hay próximas citas

## Historial de citas/servicios

No hay historial de servicios

Figura 5.11: Información del cliente. Fuente: Elaboración propia

### 5.2.5 Crear servicio

La creación de un servicio se lleva a cabo mediante la vista de Servicios, a la cual se accede desde la barra de navegación. Dentro de esta vista se debe pulsar sobre el botón "Añadir nuevo servicio "

Añadir nuevo servicio

Figura 5.12: Botón añadir nuevo servicio. Fuente: Elaboración propia

Al pulsar sobre el botón se accede al formulario que permite la creación del servicio. El formulario consta de los campos Nombre, Categoría, Descripción y Precio. Obligatorios son el nombre, la categoría y el precio, este último debe ser mayor que cero.

## Añadir nuevo servicio

Nombre  Categoría

Descripción

Precio

## Añadir nuevo servicio

Nombre  Categoría

Descripción

Precio

¡Servicio añadido!

Figura 5.13: Formulario nuevo servicio.  
Fuente: Elaboración propia

Figura 5.14: Mensaje servicio añadido.  
Fuente: Elaboración propia

El servicio añadido se puede consultar en la vista principal de los servicios, como se explica en el punto (5.2.6 - Consultar y editar servicio)

### 5.2.6 Consultar y editar servicio

Para consultar y editar un servicio se debe acceder a la vista principal de los servicios. Al acceder a la vista, se muestra una tabla con todos los servicios que se han ido añadiendo. En la tabla se indica el nombre del servicio, el precio de este y un botón de "Modificar". Cuando se quiere modificar un servicio, se presiona el botón y se accede al formulario para modificar el servicio.

Servicio	Precio	
Manicura Semipermanente	10€	<input type="button" value="Modificar"/>
Manicura Gel	20€	<input type="button" value="Modificar"/>
Pedicura Exprés	20€	<input type="button" value="Modificar"/>
Manicura Acrílico	20€	<input type="button" value="Modificar"/>
Manicura Spa	20€	<input type="button" value="Modificar"/>

Figura 5.15: Lista de servicios. Fuente: Elaboración propia

El formulario para modificar el servicio consta de los mismos campos que se usaron para crearlo, más un desplegable que permite cambiar el estado del servicio. Cuando se crea el servicio, este se crea con el estado "Disponible" y en este formulario se da la opción de modificar el estado a "Retirado ". Este estado es relevante para la funcionalidad de las citas que se verá más adelante.

### Modificar servicio

Nombre	Categoría
<input type="text" value="Manicura"/>	<input type="text" value="Spa"/>
Descripción	
<input type="text" value="Servicio de manicura Spa"/>	
Precio	Estado
<input type="text" value="20"/>	<input type="text" value="Disponible"/>
<input type="button" value="Enviar"/>	

Figura 5.16: Formulario modificar servicio. Fuente: Elaboración propia

### Modificar servicio

Nombre	Categoría
<input type="text" value="Manicura"/>	<input type="text" value="Spa"/>
Descripción	
<input type="text" value="Servicio de manicura Spa"/>	
Precio	Estado
<input type="text" value="15"/>	<input type="text" value="Disponible"/>
<input type="button" value="Enviar"/>	

¡Servicio modificado!

Figura 5.17: Mensaje servicio modificado. Fuente: Elaboración propia

## 5.2.7 Crear cita

La creación de una cita se hace desde la vista de Agenda, a la cual se accede desde la barra de navegación. Para reservar una cita se debe hacer clic sobre uno de los huecos marcados como "Disponible "o sobre el intervalo de hora en el que se desee reservar. Los huecos libres que se muestran están relacionados con el empleado que está seleccionado y con el día que indica el calendario.

Luzcarime

Aitana

09:00 - 10:00	Disponible	15:00 - 16:00	Disponible
10:00 - 11:00	Disponible	16:00 - 17:00	Disponible
11:00 - 12:00	Disponible	17:00 - 18:00	Disponible
12:00 - 13:00	Disponible	18:00 - 19:00	Disponible
13:00 - 14:00	Disponible	19:00 - 20:00	Disponible
14:00 - 15:00	Disponible	20:00 - 21:00	Disponible

2023-10-14

Figura 5.18: Sección de citas. Fuente: Elaboración propia

Al hacer clic sobre uno de los huecos disponibles, se accede al formulario de la cita.

## Nueva Cita

Cliente					
Empleado	Luzcarime				
Servicio	Elegir... <span style="float: right;">▼</span>				Añadir
Fecha	2023-10-10	Desde	--:-- <span style="float: right;">🕒</span>	Hasta	--:-- <span style="float: right;">🕒</span>

Enviar

Figura 5.19: Formulario crear cita. Fuente: Elaboración propia

Este formulario consta de los siguiente campos:

- **Cliente:** Campo para introducir el nombre del cliente. Tiene la misma funcionalidad de búsqueda explicada en la sección (5.2.3 - [Buscar cliente](#)). A medida que se introduce el nombre del cliente, sale una lista de la cual se debe seleccionar uno.
- **Empleado:** Empleado que atenderá la cita. Este campo corresponde con el empleado seleccionado en la pantalla anterior.
- **Servicio:** Desplegable con los servicios. Se debe elegir un servicio y pulsar el botón de añadir. Pueden elegir varios servicios y evita que se elijan servicios ya seleccionados.
- **Fecha:** Fecha de la cita. La fecha corresponde con la elegida en la pantalla anterior.
- **Desde:** Hora de inicio de la cita.
- **Hasta:** Hora de fin de la cita.

El botón de enviar no se habilita hasta que no se haya comprobado que la hora de inicio y la hora fin están dentro de un intervalo de tiempo libre.

Cuando se tienes todos los campos validados, se habilita el botón del enviar y se procede a reservar la cita. Se indica mediante un mensaje que la cita ha sido reservada.

### Nueva Cita

Cliente

Empleado

Servicio

Manicura Se

Mar

Margarita González - 600000001  
María López - 612612612  
Marcos Pérez - 613613613  
Marcelo Gómez - 615615615

Fecha

2023-10-11

Desde

--:--

Hasta

--:--

Enviar

### Nueva Cita

Cliente

Margarita González

Empleado

Luzcarime

Servicio

Manicura Semipermanente

Añadir

Manicura Semipermanente

X

Fecha

2023-10-11

Desde

10:00

Hasta

11:00

Enviar

¡Cita Reservada!

Figura 5.20: Búsqueda cliente nueva cita. Fuente: Elaboración propia

Figura 5.21: Cita reservada. Fuente: Elaboración propia

Si al elegir las horas de inicio y de fin no se encuentra disponible ese intervalo de tiempo, se muestra un mensaje y no se habilita el botón.

Fecha

2023-10-10

Desde

09:00

Hasta

10:00

Cita no disponible

Enviar

Figura 5.22: Cita no disponible. Fuente: Elaboración propia

Una vez reservada la cita, al volver a la vista de la agenda, se puede ver que esta se ha actualizado con la cita que se acaba de añadir.

Luzcarime

Aitana

09:00 - 10:00	Margarita González - Manicura Semipermanente	15:00 - 16:00	Disponible
10:00 - 11:00	Disponible	16:00 - 17:00	Disponible
11:00 - 12:00	Disponible	17:00 - 18:00	Disponible
12:00 - 13:00	Disponible	18:00 - 19:00	Disponible
13:00 - 14:00	Disponible	19:00 - 20:00	Disponible
14:00 - 15:00	Disponible	20:00 - 21:00	Disponible

Figura 5.23: Sección citas actualizada. Fuente: Elaboración propia

## 5.2.8 Actualizar estado cita

Se puede actualizar el estado de las citas para marcarlas como canceladas o finalizadas. Para ello se debe ir a la agenda y localizar la cita a la cual se quiere cambiar el estado (hacer clic sobre la cita en cuestión).

Una vez dentro de la cita, se muestra un formulario con los datos de la cita y los botones de Cancelar Cita y Finalizar

**Información de la cita**

Cliente	Margarita González				
Empleado	Luzcarime				
Servicios	Manicura Semipermanente				
Fecha	2023-10-10	Desde	09:00	Hasta	10:00

Cancelar Cita
Finalizar

Figura 5.24: Cambiar estado cita. Fuente: Elaboración propia

En función de la acción que se quiera realizar se pulsará un botón u otro. En ambos casos, después de pulsar el botón se pide al usuario que confirme la acción mediante un mensaje de alerta.

Cancelar Cita
Finalizar
Cancelar Cita
Finalizar

¿Está seguro que sea cancelar?

Sí
No

¿Está seguro que sea finalizar?

Sí
No

Figura 5.25: Mensaje de alerta al cancelar cita. Fuente: Elaboración propia

Figura 5.26: Mensaje de alerta al finalizar cita. Fuente: Elaboración propia

La implementación del botón de finalizar está pensada para una futura mejora en la que después de pulsar finalizar, se vaya a una vista con el resumen del servicio para que el usuario pueda añadir/quitar información del servicio prestado y dejar un registro de ello.

## 5.3 API con FastAPI

La API ha sido implementada con el framework FastAPI. Esta se encarga de recibir las peticiones del front, procesar los datos y devolver una respuesta. Para esta aplicación se han construido una serie de endpoint asociados a cada una de las funcionalidades. A continuación se describen los endpoints para la gestión de usuarios, los servicios, los clientes y las citas.

Las peticiones a la api deben ir acompañadas de un JWT que permite autentificar al usuario. Solo hay un endpoint que no requiere de este JWT y es usado para crear un usuario la primera vez que se usa la api.

### 5.3.1 Endpoints - Usuario

- **(POST) - /api/users/:** Endpoint que permite crear un usuario a partir de un email y una contraseña.
- **(POST) - /api/login/acces-token:** Mediante este endpoint se obtiene el JWT que permite autentificar al usuario. Cuando se realiza una llamada, esta debe incluir el usuario y la contraseña, estos datos se validan en la base de datos y si son correctos se devuelve un JWT. Ahora, Cada petición que realice el usuario a la api deberá incluir este JWT para poder tener acceso.

### 5.3.2 Endpoints - Servicios

- **(GET) - /api/items/services/:** Endpoint que permite obtener la lista de servicios. Al hacer la llamada se deben pasar los parámetros "skip" y "limit" para limitar los resultados obtenidos o para usarlos en la paginación.
- **(POST) - /api/items/services/:** Endpoint que se usa para la creación de un servicio a partir de los datos del servicio que se pasan en la llamada. Si el servicio se ha creado correctamente, se devuelve el mismo servicio junto con el id de la base de datos.
- **(PUT) - /api/items/services/id:** Mediante este endpoint se pueden modificar los datos de un servicio. Al hacer la llamada, se deben pasar los datos del servicio a modificar. Si la modificación se ha realizado correctamente, se devuelven los datos del servicio.
- **(GET) - /api/items/services/id:** Endpoint que permite obtener los datos de un servicio a partir del id del servicio.

### 5.3.3 Endpoints - Clientes

- **(POST) - /api/customer/:** Endpoint que permite crear un cliente en base a los datos que se pasan en el body de la petición.
- **(PUT) - /api/customer/id:** Endpoint que permite editar los datos de un cliente. En la petición deben pasarse los datos nuevos y el id de cliente. Si los datos se modifican correctamente, se devuelven los datos modificados
- **(GET) - /api/customer/nameLike/:** Endpoint que permite hacer una búsqueda de tipo wildcard en la base de datos para obtener una lista de cliente en función del nombre introducido.
- **(POST) - /api/customer/appointmentsInfo/id:** Mediante este endpoint se obtienen los datos principales del cliente y los datos de las citas pasadas y de las citas futuras.

### 5.3.4 Endpoints - Citas

- **(POST) api/appointment/:** Endpoint mediante el cual se puede reservar una cita en función de los datos pasados en la llamada. Si la cita se almacena correctamente en la base de datos, se devuelven los datos de la cita.
- **(PUT) api/appointment/id:** Endpoint para modificar el estado de una cita. Por defecto la cita se guarda con el estado "scheduled " y puede modificarse a los estados "canceled" o "finished".
- **(GET) api/appointment/dateEmployee/:** Mediante este endpoint se obtienen las citas asociadas a un empleado para un día concreto. Al hacer la petición se debe pasar la fecha y el id del empleado y, como respuesta, se recibe una estructura de datos en la que están los intervalos de tiempo disponibles y los intervalos de tiempo ocupados. Además, en los intervalos de tiempo ocupados (citas reservadas) también se incluyen los datos de la cita.

## 5.4 Aspectos destacables de programación

Durante el proyecto se han realizado desarrollos de código, tanto en el back como en el front, que requieren del uso de librerías o de técnicas específicas para lograr una funcionalidad, como por ejemplo el uso de los JWT. También se han tenido en cuenta técnicas recomendadas en el uso del framework para mejorar la eficiencia de este, como puede ser el uso de módulos en Angular.

A continuación se detallan algunos aspectos de la programación que han sido relevantes en el proyecto.



### 5.4.1 Angular - Separación en módulos

La aplicación se divide en tres partes: Agenda, Clientes y Servicios, cada una con sus características únicas. En lugar de tener un solo módulo principal con toda la app, se utiliza un módulo separado para cada una de estas funciones, siguiendo las recomendaciones de Angular (Angular-Modules, s.f.). Esta estrategia facilita la organización, el mantenimiento y la escalabilidad del código, además de mejorar el rendimiento de la carga inicial de la aplicación.

En el código del AppModule (**Código 5.1**) se puede ver que únicamente se usan los componentes necesarios en dicho módulo. Por otro lado, en el módulo de agenda (**Código 5.2**) se puede ver que gestiona sus propios componentes y que se comunica con el módulo principal (CommonModule). Esto mismo se aplica en los módulos de Clientes y Servicios.

```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    ErrorComponent,
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    HttpClientModule,
    NgbModule,
    AppRoutingModule
  ],
  providers: [
    AuthService,
    CookieService,
    {provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor,
      multi:true}
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Código 5.1: app.module.ts

```
@NgModule({
  declarations: [
```

```

    AgendaComponent ,
    NavbarComponent ,
    FilterComponent ,
    TimeBarComponent ,
    AppointmentComponent ,
    AppointmentAddComponent ,
    AppointmentInfoComponent ,
    HomeComponent ,
  ],
  imports: [
    CommonModule ,
    DatePickerComponent ,
    ReactiveFormsModule ,
    AgendaRoutingModule
  ]
})
export class AgendaModule { }

```

Código 5.2: agenda.module.ts

### 5.4.2 Angular - Lazy loading

La estrategia de separa la aplicación en módulos (5.4.1 Angular - Separación en módulos) permite hacer uso la técnica de "Lazy Loading". Esta técnica evita que se carguen todos los módulos al arrancar la aplicación, por el contrario, los módulos son cargados cuando son necesarios. En la aplicación, el único módulo que se carga al arrancar es la Agenda, los demás módulos se cargan cuando el usuario accede a los Clientes o a los servicios.

```

const mainRoutes: Routes = [
  {path: '', component: MainComponent, children:[
    {path: 'agenda', loadChildren:()=>import('../agenda/
      agenda.module').then(m=>m.AgendaModule)},
    {path: 'customer',loadChildren:()=>import('../customer/
      customer.module').then(m=>m.CustomerModule)},
    {path: 'service',loadChildren:()=>import('../service/
      service.module').then(m=>m.ServiceModule)}
  ]},
];

@NgModule({

```

```
imports: [RouterModule.forChild(mainRoutes)],
exports: [RouterModule]
})
export class MainRoutingModule { }
```

Código 5.3: lazy-loading

### 5.4.3 Búsqueda de cliente en tiempo real

Cuando se desea buscar un cliente se da la posibilidad de hacer una búsqueda en tiempo real. A medida que el usuario va introduciendo el nombre, se van mostrando en una lista los clientes obtenidos en la base de datos.

La detección de cambios en un formulario se controla mediante un observable que aplica un retraso de 200 ms para evitar búsquedas excesivas, ejecutando una búsqueda cuando el usuario ha terminado de ingresar su consulta.

```
@Component({
  selector: 'app-customer-search',
  templateUrl: './customer-search.component.html',
  styleUrls: ['./customer-search.component.css']
})
export class CustomerSearchComponent {

  searchControl = new FormControl();
  customers: CustomerInDB[] = [];
  errorMessage = "";
  constructor(private customerService: CustomerService){}

  ngOnInit(): void {
    try {
      this.searchControl.valueChanges
        .pipe(debounceTime(200))
        .subscribe((query) => {this.search(query)});
    } catch (error) { this.handleError(error); }
  }

  search(query: string) {
    if(query !== ''){
      this.customerService.fetchCustomers(query).subscribe({
        next:(info) =>{this.customers = info},
        error: (error) =>{ throw error }
      })
    }
  }
}
```

```
        }else{this.customers=[];}  
    }  
  
    handleError(error:any){  
        this.errorMessage=error  
    }  
}
```

Código 5.4: Búsqueda en tiempo real

### 5.4.4 Autenticación mediante JWT

Para controlar el acceso a la aplicación se ha implementado la autenticación mediante JWT. Para ello, se ha creado un endpoint en el servidor que se encarga de generar el token. Este endpoint llama a una función que maneja el proceso de autenticación de usuario y emite un token de acceso compatible con OAuth2. El token de acceso permitirá que el cliente realice solicitudes seguras a la API en nombre del usuario.

```
@router.post("/login/access-token", response_model=schemas.  
    Token)  
def login_access_token(  
    db: Session = Depends(deps.get_db), form_data:  
        OAuth2PasswordRequestForm = Depends()  
) -> Any:  
    user = crud.user.authenticate(  
        db, email=form_data.username, password=form_data.  
            password  
    )  
    if not user:  
        raise HTTPException(status_code=400, detail="Incorrect email or password")  
    elif not crud.user.is_active(user):  
        raise HTTPException(status_code=400, detail="Inactive user")  
    access_token_expires = timedelta(minutes=settings.  
        ACCESS_TOKEN_EXPIRE_MINUTES)  
    return {  
        "access_token": security.create_access_token(  
            user.id, expires_delta=access_token_expires  
        ),  
        "token_type": "bearer",  
    }
```

Código 5.5: API - Obtencion JWT

Para gestionar el JWT en el front, se ha creado un servicio con las funciones para obtener el token, guardarlo en una cookie y obtenerlo de la cookie.

La función que obtiene el token de la cookie la invoca un interceptor. Este interceptor se encarga de interceptar todas las peticiones http del front y buscar la cookie donde está el JWT, si la encuentra, clona la petición con la cookie y continua con la llamada.

```
export var loginEndpoint = "http://localhost:8000/api/login/
access-token";
const headers = new HttpHeaders()
  .set('Accept', 'application/json') // Cabecera Accept
  .set('Content-Type', 'application/x-www-form-
urlencoded'); // Cabecera Content-Type

@Injectable()
export class AuthService{

  constructor(
    private http: HttpClient,
    private cookieService: CookieService) {

  }

  login(credentials:UserCredentials):Observable<any>{
    let formData = new URLSearchParams();
    formData.append('username',credentials.username);
    formData.append('password',credentials.password);
    return this.http.post(loginEndpoint,formData,{
      headers})
  }

  setJwtCookie(token:string){

    this.cookieService.set('jwtToken',token);
  }

  getJwtCookie(): string | null {

    const jwtExist = this.cookieService.check('jwtToken'
    );
    return jwtExist ? this.cookieService.get('jwtToken')
      : null;
  }
}
```

Código 5.6: Servicio para JWT

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor }
  from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthService } from '../services/auth.service';

@Injectable()
export class JwtInterceptor implements HttpInterceptor {

  constructor(
    private authservice: AuthService
  ) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler
    ): Observable<HttpEvent<unknown>> {

    const token = this.authservice.getJwtCookie();
    if (token){

      request=request.clone({
        setHeaders:{
          Authorization: 'Bearer ${token}'
        }
      });
    }
    return next.handle(request);
  }
}
```

Código 5.7: Interceptor

## 6. Conclusiones

En este Trabajo de Fin de Máster se ha logrado desarrollar una aplicación web destinada a la gestión de un salón de belleza. Esta aplicación ofrece una mejora significativa en la gestión del negocio al proporcionar un sistema que simplifica la organización y el acceso a la información de manera eficiente.

Se han implementado una serie de funcionalidades clave destinadas a mejorar la gestión de los clientes, permitiendo la recopilación y el almacenamiento de información relevante que puede aprovecharse para ofrecer una atención más personalizada. Además, se ha incorporado una sección dedicada a la programación y gestión de citas, lo que contribuye a la optimización de la agenda del salón. Por último, se ha desarrollado una funcionalidad que simplifica la gestión de los servicios ofrecidos.

Cabe destacar que cada una de las funcionalidades integradas en la aplicación ha sido diseñada y desarrollada con recursos limitados, lo que subraya la eficiencia y la efectividad del uso de tecnologías de desarrollo web modernas, como FastAPI y Angular. Estas herramientas han demostrado ser esenciales para simplificar el proceso de desarrollo, permitiendo la creación de una aplicación de alto valor incluso en un entorno con recursos limitados.

En resumen, este Trabajo de Fin de Máster ha culminado en la creación de una aplicación web que digitaliza y mejora sustancialmente la gestión de un salón de belleza. A pesar de que la aplicación actual cuenta con funcionalidades básicas, está diseñada para su expansión gradual con el tiempo, con el objetivo de lograr una aplicación más completa y versátil que permita una gestión integral del negocio en el futuro.

## 7. Trabajos futuros

El esfuerzo invertido en este Trabajo de Fin de Máster (TFM) ha culminado en el desarrollo de una aplicación web con funcionalidades iniciales sólidas, que cumplen con el propósito inicial: la creación de una plataforma que facilite la administración de un negocio. Sin embargo, este proyecto marca el punto de partida, y existen diversas oportunidades para ampliar, mejorar y optimizar aún más la aplicación en futuros trabajos. A continuación, se detallan algunas áreas y tareas que pueden considerarse para trabajos futuros:

- **Desarrollar nuevas funcionalidades**
  - **Gestión de empleados:** Actualmente la creación y la gestión de los empleados se realiza desde la api. Esta funcionalidad no se ha llevado todavía hasta el front. Habría que crear un panel de administrador para que se pueda realizar esta gestión.
  - **Gestión de productos:** A esta funcionalidad no se le ha dado prioridad porque actualmente no se gestionan una gran cantidad de productos, sin embargo es una funcionalidad que está en la lista de requisitos y que se llevará a cabo más adelante.
  - **Resumen del servicio:** Se pretende tener un registro más completo de los servicios prestados con el fin de tener información relevante que ayude a la toma de decisiones. Aunque el en diseño de la base de datos ya se contempló esta opción, no se ha llevado a cabo por falta de tiempo.
  - **Control de gastos:** Se pretende desarrollar una funcionalidad en la que se puedan registrar los gastos para poder tener un control más detallados de estos.
  - **Envío de recordatorio de citas:** Aprovechando que ya se tiene una base de datos con la información de las citas de los clientes, se pretende automatizar el envío de notificaciones a los clientes para informarles de sus próximas citas.
- **Docker:** La aplicación ha sido desarrollada de manera tradicional, pero se pretende hacer uso de tecnologías de contenedores, como es docker, para facilitar el despliegue, la escalabilidad y el propio desarrollo de la aplicación.
- **Certificados SSL:** Para que la comunicación sea segura y se puedan proteger los datos, se han de instalar certificados SSL para hacer uso del protocolo HTTPS.
- **Desplegar en la nube:** Se pretende usar la nube para hacer el despliegue de la aplicación en producción. La adopción de la nube es una estrategia que se



toma cada vez más para el despliegue de las aplicaciones, teniendo en cuenta todos los servicios que ofrece (escalabilidad, seguridad, mantenimiento, etc) y el reducido coste de estos, hoy en día se ha convertido en una herramienta fundamental para la transformación digital.

## A. Apéndice I: Diseño lógico BBDD

Customer(id, name, last\_name, phone\_number, email, birth\_day, last\_visit)  
CP: {id}

Employee (id, name, last\_name, phone\_number, email, rol)  
CP: {id}

Appoinment (id, id\_customer, id\_employee, appointment\_date,  
appointment\_time\_star,appointment\_time\_end, state)  
CP: {id}

CAj1: Appointment.id\_customer es clave ajena de Customer.id  
Nulos: No  
Actualizar: Propagar  
Borrar:Restringir

CAj2: Appointment.id\_employee es clave ajena de Employee.id  
Nulos No  
Actualizar: Propagar  
Borrar: Restringir

Item(id, item\_type)  
CP: {id}

Product\_item (id\_item, name, category, descripcion, stock, price, state)  
CAj: Product\_item.id\_item es clave ajena de Item.id  
Nulos: No  
Actualizar: Propagar  
Borrar: Restringir

Service\_item (id\_item, name, category, description, price, state)  
CAj: Service\_item.id\_item es clave ajena de item.id  
Nulos:No  
Actualizar: Propagar  
Borrar: Restringir

Appointment\_item (id,id\_item, id\_appointment) - Relacion To Include  
CP: {id,id\_item, id\_appointment}

CAj1: Appointment\_item.id\_appointment es clave ajena de Appointment.id  
Nulos No

Actualizar: Propagar  
Borrar: Restringir

CAj2: Appointment\_item.id\_item es clave ajena de Item.id  
Nulos: No  
Actualizar: Propagar  
Borrar: Restringir

Transaction\_detail(id, id\_item, id\_transaction, quantity, unit\_price)  
CP: {id}

CAj: Transaction\_detail.id\_item es clave ajena de Item.id  
Nulos: No  
Actualizar: Propagar  
Borrar: Restringir

Transaction: (id, id\_customer, id\_employee, id\_promotion, transaction\_date)  
CP: {id}

CAj1: Transaction.id\_customer es clave ajena de Customer.id  
Nulos: No  
Actualizar: Propagar  
Borrar: Restringir

CAj2: Transaction.id\_employee es clave ajena de Employee.id  
Nulos: No  
Actualizar: Propagar  
Borrar: Restringir

Caj3: Transaction.id\_promocion es clave ajena de Promotion.id  
Nulos: Sí  
Actualizar: Propagar  
Borrar: Restringir

Promotion: (id, name, description, promotion\_type, value, state)  
CP: {id}

## B. Apéndice II: Diseño físico BBDD

A continuación, se presenta el diseño físico de la base de datos. Para la construcción y administración de la base de datos, se ha utilizado la herramienta Alembic. Alembic es una herramienta que se emplea en conjunto con la biblioteca SQLAlchemy, la cual actúa como un ORM para la interacción con bases de datos en Python.

```
def upgrade() -> None:
    op.create_table('customer',
        sa.Column('id', sa.Integer(), autoincrement=True,
            nullable=False),
        sa.Column('name', sa.String(), nullable=False),
        sa.Column('last_name', sa.String(), nullable=True),
        sa.Column('alias', sa.String(), nullable=True),
        sa.Column('phone_number', sa.String(), nullable=True),
        sa.Column('email', sa.String(), nullable=True),
        sa.Column('birth_day', sa.Date(), nullable=True),
        sa.Column('last_visit', sa.Date(), nullable=True),
        sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_customer_birth_day'), 'customer',
        ['birth_day'], unique=False)
    op.create_index(op.f('ix_customer_email'), 'customer', ['email'], unique=True)
    op.create_index(op.f('ix_customer_id'), 'customer', ['id'], unique=False)
    op.create_index(op.f('ix_customer_last_name'), 'customer', ['last_name'], unique=False)
    op.create_index(op.f('ix_customer_last_visit'), 'customer', ['last_visit'], unique=False)
    op.create_index(op.f('ix_customer_name'), 'customer', ['name'], unique=False)
    op.create_index(op.f('ix_customer_phone_number'), 'customer', ['phone_number'], unique=True)
    op.create_table('item',
        sa.Column('id', sa.Integer(), autoincrement=True,
            nullable=False),
        sa.Column('item_type', sa.String(), nullable=False),
        sa.PrimaryKeyConstraint('id')
```

```

)
op.create_index(op.f('ix_item_id'), 'item', ['id'],
               unique=False)
op.create_index(op.f('ix_item_item_type'), 'item', ['
    item_type'], unique=False)
op.create_table('promotion',
sa.Column('id', sa.Integer(), autoincrement=True,
          nullable=False),
sa.Column('name', sa.String(), nullable=True),
sa.Column('promotion_type', sa.String(), nullable=True),
sa.Column('description', sa.String(length=200), nullable
          =True),
sa.Column('value', sa.Float(), nullable=True),
sa.Column('state', sa.String(), nullable=True),
sa.PrimaryKeyConstraint('id')
)
op.create_index(op.f('ix_promotion_description'), '
    promotion', ['description'], unique=False)
op.create_index(op.f('ix_promotion_id'), 'promotion', ['
    id'], unique=False)
op.create_index(op.f('ix_promotion_name'), 'promotion',
               ['name'], unique=False)
op.create_index(op.f('ix_promotion_promotion_type'), '
    promotion', ['promotion_type'], unique=False)
op.create_index(op.f('ix_promotion_state'), 'promotion',
               ['state'], unique=False)
op.create_table('user',
sa.Column('id', sa.Integer(), autoincrement=True,
          nullable=False),
sa.Column('name', sa.String(), nullable=False),
sa.Column('last_name', sa.String(), nullable=False),
sa.Column('phone_number', sa.String(length=9), nullable=
          False),
sa.Column('email', sa.String(), nullable=False),
sa.Column('hashed_password', sa.String(), nullable=False
          ),
sa.Column('rol', sa.String(), nullable=False),
sa.Column('is_active', sa.Boolean(), nullable=True),
sa.Column('is_superuser', sa.Boolean(), nullable=True),
sa.PrimaryKeyConstraint('id')
)
op.create_index(op.f('ix_user_email'), 'user', ['email'
    ], unique=True)
op.create_index(op.f('ix_user_id'), 'user', ['id'],

```

```

    unique=False)
op.create_index(op.f('ix_user_last_name'), 'user', ['
    last_name'], unique=False)
op.create_index(op.f('ix_user_name'), 'user', ['name'],
    unique=False)
op.create_index(op.f('ix_user_phone_number'), 'user', ['
    phone_number'], unique=True)
op.create_table('appointment',
sa.Column('id', sa.Integer(), autoincrement=True,
    nullable=False),
sa.Column('id_customer', sa.Integer(), nullable=False),
sa.Column('id_employee', sa.Integer(), nullable=False),
sa.Column('appointment_date', sa.Date(), nullable=True),
sa.Column('appointment_time_start', sa.Time(), nullable=
    True),
sa.Column('appointment_time_end', sa.Time(), nullable=
    True),
sa.Column('state', sa.String(), nullable=True),
sa.ForeignKeyConstraint(['id_customer'], ['customer.id'
    ], onupdate='CASCADE', ondelete='RESTRICT'),
sa.ForeignKeyConstraint(['id_employee'], ['user.id'],
    onupdate='CASCADE', ondelete='RESTRICT'),
sa.PrimaryKeyConstraint('id')
)
op.create_index(op.f('ix_appointment_appointment_date'),
    'appointment', ['appointment_date'], unique=False)
op.create_index(op.f('
    ix_appointment_appointment_time_end'), 'appointment',
    ['appointment_time_end'], unique=False)
op.create_index(op.f('
    ix_appointment_appointment_time_start'), 'appointment
    ', ['appointment_time_start'], unique=False)
op.create_index(op.f('ix_appointment_id'), 'appointment'
    , ['id'], unique=False)
op.create_index(op.f('ix_appointment_id_customer'), '
    appointment', ['id_customer'], unique=False)
op.create_index(op.f('ix_appointment_id_employee'), '
    appointment', ['id_employee'], unique=False)
op.create_index(op.f('ix_appointment_state'), '
    appointment', ['state'], unique=False)
op.create_table('product_item',
sa.Column('id', sa.Integer(), autoincrement=True,
    nullable=False),
sa.Column('item_id', sa.Integer(), nullable=False),

```

```

sa.Column('name', sa.String(), nullable=True),
sa.Column('category', sa.String(), nullable=True),
sa.Column('description', sa.String(length=200), nullable
    =True),
sa.Column('stock', sa.Integer(), nullable=True),
sa.Column('price', sa.Float(), nullable=True),
sa.Column('state', sa.String(), nullable=True),
sa.ForeignKeyConstraint(['item_id'], ['item.id'],
    onupdate='CASCADE', ondelete='RESTRICT'),
sa.PrimaryKeyConstraint('id')
)
op.create_index(op.f('ix_product_item_category'), '
    product_item', ['category'], unique=False)
op.create_index(op.f('ix_product_item_description'), '
    product_item', ['description'], unique=False)
op.create_index(op.f('ix_product_item_id'), '
    product_item', ['id'], unique=False)
op.create_index(op.f('ix_product_item_item_id'), '
    product_item', ['item_id'], unique=False)
op.create_index(op.f('ix_product_item_name'), '
    product_item', ['name'], unique=False)
op.create_index(op.f('ix_product_item_state'), '
    product_item', ['state'], unique=False)
op.create_table('service_item',
sa.Column('id', sa.Integer(), autoincrement=True,
    nullable=False),
sa.Column('item_id', sa.Integer(), nullable=False),
sa.Column('name', sa.String(), nullable=True),
sa.Column('category', sa.String(), nullable=True),
sa.Column('description', sa.String(length=200), nullable
    =True),
sa.Column('price', sa.Float(), nullable=True),
sa.Column('state', sa.String(), nullable=True),
sa.ForeignKeyConstraint(['item_id'], ['item.id'],
    onupdate='CASCADE', ondelete='RESTRICT'),
sa.PrimaryKeyConstraint('id')
)
op.create_index(op.f('ix_service_item_category'), '
    service_item', ['category'], unique=False)
op.create_index(op.f('ix_service_item_description'), '
    service_item', ['description'], unique=False)
op.create_index(op.f('ix_service_item_id'), '
    service_item', ['id'], unique=False)
op.create_index(op.f('ix_service_item_item_id'), '

```

```

        service_item', ['item_id'], unique=False)
    op.create_index(op.f('ix_service_item_name'), '
        service_item', ['name'], unique=False)
    op.create_index(op.f('ix_service_item_state'), '
        service_item', ['state'], unique=False)
    op.create_table('transaction',
        sa.Column('id', sa.Integer(), autoincrement=True,
            nullable=False),
        sa.Column('id_customer', sa.Integer(), nullable=False),
        sa.Column('id_employee', sa.Integer(), nullable=False),
        sa.Column('id_promotion', sa.Integer(), nullable=True),
        sa.Column('transaction_date', sa.DateTime(), nullable=
            True),
        sa.ForeignKeyConstraint(['id_customer'], ['customer.id'
            ], onupdate='CASCADE', ondelete='RESTRICT'),
        sa.ForeignKeyConstraint(['id_employee'], ['user.id'],
            onupdate='CASCADE', ondelete='RESTRICT'),
        sa.ForeignKeyConstraint(['id_promotion'], ['promotion.id
            '], onupdate='CASCADE', ondelete='RESTRICT'),
        sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_transaction_id'), 'transaction'
        , ['id'], unique=False)
    op.create_index(op.f('ix_transaction_id_customer'), '
        transaction', ['id_customer'], unique=False)
    op.create_index(op.f('ix_transaction_id_employee'), '
        transaction', ['id_employee'], unique=False)
    op.create_index(op.f('ix_transaction_id_promotion'), '
        transaction', ['id_promotion'], unique=False)
    op.create_index(op.f('ix_transaction_transaction_date'),
        'transaction', ['transaction_date'], unique=False)
    op.create_table('appointment_item',
        sa.Column('id', sa.Integer(), autoincrement=True,
            nullable=False),
        sa.Column('id_item', sa.Integer(), nullable=False),
        sa.Column('id_appointment', sa.Integer(), nullable=False
        ),
        sa.ForeignKeyConstraint(['id_appointment'], ['
            appointment.id'], onupdate='CASCADE', ondelete='
            RESTRICT'),
        sa.ForeignKeyConstraint(['id_item'], ['item.id'],
            onupdate='CASCADE', ondelete='RESTRICT'),
        sa.PrimaryKeyConstraint('id', 'id_item', 'id_appointment
        ')

```



```

)
op.create_index(op.f('ix_appointment_item_id'), '
    appointment_item', ['id'], unique=False)
op.create_index(op.f('ix_appointment_item_id_appointment
    '), 'appointment_item', ['id_appointment'], unique=
    False)
op.create_index(op.f('ix_appointment_item_id_item'), '
    appointment_item', ['id_item'], unique=False)
op.create_table('transaction_detail',
sa.Column('id', sa.Integer(), autoincrement=True,
    nullable=False),
sa.Column('id_transaction', sa.Integer(), nullable=False
    ),
sa.Column('id_item', sa.Integer(), nullable=False),
sa.Column('quantity', sa.Integer(), nullable=True),
sa.Column('unit_price', sa.Float(), nullable=True),
sa.ForeignKeyConstraint(['id_item'], ['item.id'],
    onupdate='CASCADE', ondelete='RESTRICT'),
sa.ForeignKeyConstraint(['id_transaction'], ['
    transaction.id'], onupdate='CASCADE', ondelete='
    RESTRICT'),
sa.PrimaryKeyConstraint('id')
)
op.create_index(op.f('ix_transaction_detail_id'), '
    transaction_detail', ['id'], unique=False)
op.create_index(op.f('ix_transaction_detail_id_item'), '
    transaction_detail', ['id_item'], unique=False)
op.create_index(op.f('
    ix_transaction_detail_id_transaction'), '
    transaction_detail', ['id_transaction'], unique=False
    )

```

Código B.1: Ejemplo de código Python

# Referencias

- Angular-Modules. (s.f.). *Feature modules*. <https://angular.io/guide/feature-modules>. (Consultado el 10 de agosto de 2023)
- Bansal, P., y Ouda, A. (2022). Study on integration of fastapi and machine learning for continuous authentication of behavioral biometrics. En *2022 international symposium on networks, computers and communications (isncc)* (p. 1-6). doi: 10.1109/ISNCC55209.2022.9851790
- Django. (s.f.). *Django*. [https://es.wikipedia.org/wiki/Django\\_\(framework\)](https://es.wikipedia.org/wiki/Django_(framework)). (Consultado el 23 de julio de 2023)
- FastAPI. (s.f.). *Fastapi*. <https://fastapi.tiangolo.com/es/>. (Consultado el 2 de junio de 2023)
- Fielding, R., y Taylor, R. (2000). Principled design of the modern web architecture. En *Proceedings of the 2000 international conference on software engineering. icse 2000 the new millennium* (p. 407-416). doi: 10.1145/337180.337228
- Flask. (s.f.). *Flask*. <https://es.wikipedia.org/wiki/Flask>. (Consultado el 11 de julio de 2023)
- JSON. (s.f.). *Json-schema*. <https://json-schema.org/>. (Consultado el 24 de agosto de 2023)
- Kevin Glisson, F. M., Marc Vilanova. (2020). *Netflix technology blog. introducing dispatch*. <https://netflixtechblog.com/introducing-dispatch-da4b8a2a8072>. (Consultado el 1 de agosto de 2023)
- Kumar, K., Jain, A. K., Tiwari, R. G., Jain, N., Gautam, V., y Trivedi, N. K. (2023). Analysis of api architecture: A detailed report. En *2023 ieee 12th international conference on communication systems and network technologies (csnt)* (p. 880-884). doi: 10.1109/CSNT57126.2023.10134658
- OpenAPI. (s.f.). *Openapi specification*. <https://github.com/OAI/OpenAPI-Specification>. (Consultado el 14 de junio de 2023)
- Piero Molino, S. S. M., Yaroslav Dudin. (2019). *Uber. ludwig v0.2 adds new features and other improvements to its deep learning toolbox*. <https://www.uber.com/en-ES/blog/ludwig-v0-2/>. (Consultado el 1 de agosto de 2023)
- Pydantic. (s.f.). *Pydantic*. <https://docs.pydantic.dev/latest/>. (Consultado el 4 de junio de 2023)
- ReDoc. (s.f.). *Redoc*. <https://github.com/Redocly/redoc>. (Consultado el 14 de junio de 2023)
- Swagger. (s.f.). *Swagger*. <https://github.com/swagger-api/swagger-ui>. (Consultado el 14 de junio de 2023)