

# PFC: Hotel Management WebApp

---

Florida Universitaria

Víctor Giménez Gil

2024/2025



Florida

Universitaria



GENERALITAT  
VALENCIANA

CONSELLERIA D'EDUCACIÓ,  
INVESTIGACIÓ, CULTURA I ESPORT

## Tabla de contenido

1. Resumen del proyecto.....	3
2. Justificación del proyecto .....	4
3. Desarrollo del proyecto.....	6
3.1. Análisis del mercado y posible modelo de negocio:.....	6
3.2. Metodologías utilizadas: .....	7
3.3. Descripción de los componentes de la aplicación.....	8
3.3.1. Arquitectura General .....	8
3.3.2. Frontend (Angular) .....	9
3.3.3. Mockups y Diseño de Interfaz .....	11
3.3.4. Backend (Symfony) .....	12
3.3.5. Base de Datos .....	15
3.4. Problemas/dificultades encontradas: .....	17
4. Conclusiones .....	20
5. Líneas futuras de trabajo .....	21
6. Bibliografía.....	23

# 1. Resumen del proyecto

## RESUMEN:

Este proyecto consiste en una aplicación web para la gestión integral de un hotel, desarrollada con tecnologías modernas como Angular (frontend) y Symfony (backend).

La aplicación está dividida en dos áreas: una parte pública para clientes, que incluye información del hotel y reservas, y una parte privada para la gestión de reservas y administración de clientes.

El objetivo principal es ofrecer una herramienta eficiente, moderna y fácil de usar tanto para los huéspedes del hotel como para el personal administrativo, mejorando la experiencia de usuario y optimizando los procesos internos del hotel.

## ABSTRACT:

This project is a comprehensive web application for hotel management, developed using modern technologies such as Angular (frontend) and Symfony (backend).

The application is divided into two sections: a public area for clients, including hotel information and reservations, and a private area for reservation and client management.

The main goal is to provide an efficient, modern, and user-friendly tool for both hotel guests and administrative staff, enhancing the user experience and optimizing the hotel's internal processes.

## 2. Justificación del proyecto

La necesidad de una herramienta digital eficiente para la gestión hotelera motivó la creación de este proyecto.

En la actualidad, la digitalización es clave para la competitividad en el sector turístico. Muchos hoteles, especialmente los pequeños y medianos, pueden carecer de sistemas de gestión modernos, integrales o fáciles de usar, o depender de múltiples herramientas desconectadas.

Este proyecto busca ofrecer una solución unificada que aborde estas carencias.

Está dirigido a hoteles que buscan mejorar la experiencia del cliente, desde la consulta de información y la reserva online hasta la gestión interna de dichas reservas y la administración de los datos de los clientes, y optimizar sus procesos internos haciéndolos más ágiles y menos propensos a errores.

**Los objetivos principales del proyecto se pueden desglosar en:**

- **Objetivo General:**
  - Desarrollar una aplicación web completa para la gestión hotelera que sea intuitiva, eficiente y escalable.
- **Objetivos Específicos:**
  - Facilitar la gestión de reservas, incluyendo altas, bajas, modificaciones, y el proceso de check-in/check-out para el personal del hotel.
  - Ofrecer una experiencia de usuario mejorada para los clientes a través de una interfaz pública moderna, atractiva y funcional, permitiendo consultar información del hotel, tipos de habitaciones, precios y realizar reservas de forma sencilla.
  - Implementar un sistema de autenticación seguro para el área privada, diferenciando roles de usuario (ej. empleado, administrador) si fuera necesario para futuras ampliaciones.
  - Proporcionar un sistema escalable, robusto y fácil de mantener, utilizando tecnologías actuales y buenas prácticas de desarrollo.
  - Asegurar la portabilidad y facilidad de instalación de la aplicación mediante el uso de Docker y Docker Compose.

### 3. Desarrollo del proyecto

#### 3.1. Análisis del mercado y posible modelo de negocio:

Se identificaron diversas aplicaciones similares en el mercado de software de gestión hotelera (PMS - Property Management Systems). Estas soluciones varían desde sistemas complejos y costosos, orientados a grandes cadenas hoteleras, hasta herramientas más sencillas o especializadas en nichos. Muchas soluciones existentes pueden ser complejas de implementar para hoteles pequeños o medianos, o carecer de interfaces de usuario modernas y amigables que se adapten bien a dispositivos móviles.

Este proyecto se diferencia por su enfoque en la experiencia del usuario (UX), tanto para el cliente final como para el personal del hotel, y el uso de tecnologías modernas (Angular y Symfony) que aseguran un buen rendimiento, mantenibilidad y escalabilidad. La interfaz busca ser limpia, intuitiva y adaptada a las necesidades actuales.

El modelo de negocio más viable para este tipo de aplicación sería un **SaaS (Software as a Service)**. Esto permitiría ofrecer la aplicación a hoteles pequeños y medianos mediante una suscripción mensual o anual, eliminando la necesidad de una gran inversión inicial en software y hardware por parte del hotel. Se podrían ofrecer diferentes planes según el tamaño del hotel, número de habitaciones o funcionalidades requeridas. Este modelo también facilita las actualizaciones y el mantenimiento centralizado de la aplicación.

La propuesta de valor se centraría en la facilidad de uso, la eficiencia operativa que aporta, una experiencia de cliente mejorada y un coste de entrada accesible.

### 3.2. Metodologías utilizadas:

Para garantizar un desarrollo ágil, colaborativo y adaptable a los cambios, se utilizó la metodología **SCRUM**. Aunque el proyecto fue desarrollado individualmente, se aplicaron los principios de SCRUM para la organización del trabajo:

- **Product Backlog:** Se elaboró un listado priorizado de todas las funcionalidades y requisitos de la aplicación (tanto para la parte pública como la privada).
- **Sprints:** El desarrollo se dividió en iteraciones cortas (Sprints, por ejemplo, de dos semanas) donde se abordaba un conjunto de funcionalidades del Product Backlog.
- **Sprint Planning:** Al inicio de cada Sprint, se seleccionaban las tareas a realizar.
- **Daily Scrum (simulado):** Se realizaba una revisión diaria del progreso, los obstáculos y la planificación para el día.
- **Sprint Review:** Al finalizar cada Sprint, se revisaba el incremento funcional desarrollado.
- **Sprint Retrospective (simulado):** Se reflexionaba sobre el proceso para identificar mejoras para los siguientes Sprints.

Esta adaptación de SCRUM permitió un desarrollo incremental, facilitando la gestión de la complejidad del proyecto, la detección temprana de problemas y la adaptación a nuevas ideas o requisitos que pudieran surgir.

La elección de SCRUM también se alinea con las directrices de evaluación del módulo de proyecto, que fomenta la aplicación de metodologías ágiles.

### 3.3. Descripción de los componentes de la aplicación

#### 3.3.1. Arquitectura General

La aplicación sigue una arquitectura cliente-servidor desacoplada, una práctica estándar en el desarrollo web moderno. Esta arquitectura promueve la separación de responsabilidades, la escalabilidad y la flexibilidad.

- **Cliente (Frontend):** Es una Single Page Application (SPA) desarrollada con Angular. Se encarga de toda la lógica de presentación, la interacción con el usuario y la comunicación con el backend a través de llamadas HTTP a la API REST.
- **Servidor (Backend):** Desarrollado con Symfony, expone una API RESTful que maneja toda la lógica de negocio, el acceso y la manipulación de datos, y la autenticación de usuarios.
- **Comunicación:** El frontend y el backend se comunican exclusivamente mediante peticiones HTTP. Los datos se intercambian en formato JSON.
- **Base de Datos:** El backend utiliza Doctrine ORM para interactuar con una base de datos relacional MariaDB donde se persiste toda la información de la aplicación.
- **Autenticación:** Se implementa un sistema de autenticación basado en JSON Web Tokens (JWT). El cliente envía las credenciales al backend, y si son válidas, el backend devuelve un JWT que el cliente almacenará y enviará en las cabeceras de las solicitudes subsiguientes para acceder a rutas protegidas.

Esta arquitectura permite que el frontend y el backend se desarrollen, desplieguen y escalen de forma independiente.



### 3.3.2. Frontend (Angular)

La parte pública y privada del cliente web se ha desarrollado utilizando el framework **Angular** (versión ~19.2.0), basado en TypeScript.

El diseño del frontend se ha enfocado en ofrecer una experiencia de usuario intuitiva, accesible, con una navegación clara y un rendimiento óptimo.

- **Tecnologías y Librerías Clave:**

- **Angular Core:** Para la estructura fundamental de la aplicación, componentes, directivas, servicios, inyección de dependencias, etc.
- **Angular Router:** Para gestionar la navegación entre las diferentes vistas (páginas) de la aplicación.
- **RxJS:** Para la programación reactiva, manejo de eventos asíncronos y flujos de datos, especialmente en la comunicación con la API.
- **Bootstrap (^5.3.3) y @ng-bootstrap/ng-bootstrap (^18.0.0):** Para la maquetación y los componentes de interfaz de usuario (modales, calendarios, paginación, etc.), asegurando un diseño moderno, responsive y consistente. Se utilizan también Bootstrap Icons para la iconografía.
- **Angular Forms:** Para la creación y validación de formularios, tanto en la parte pública (reservas) como en la privada (gestión de clientes, check-in).

- **Estructura:**
  - **Componentes:** La interfaz se construye a base de componentes reutilizables (ej. HeaderComponent, FooterComponent, LoaderComponent).
  - **Servicios:** La lógica de negocio y la comunicación con la API del backend se encapsulan en servicios (ej. AuthService, BookingService, RoomService).
  - **Modelos:** Se definen interfaces o clases TypeScript para representar las entidades de datos (ej. User, Room, Booking).
  - **Páginas/Vistas:** Componentes de nivel superior que representan las diferentes secciones de la aplicación (ej. HomePage, RoomsPage, AdminDashboardPage).
- **Funcionalidades Implementadas:**
  - **Parte Pública:**
    - Página de inicio atractiva y centrada en la UX.
    - Sección de información sobre el hotel.
    - Catálogo de tipos de habitaciones con detalles y precios.
    - Formulario para realizar reservas.
  - **Parte Privada (Dashboard):**
    - Login seguro para trabajadores y administradores.
    - Gestión de check-in y check-out de huéspedes.
    - Registro, consulta y administración de la información de los clientes.
    - Listado y gestión de reservas.
    - El diseño del frontend se ha enfocado en ofrecer una experiencia de usuario intuitiva, con una navegación clara y un rendimiento óptimo.

### 3.3.3. Mockups y Diseño de Interfaz

El diseño de la interfaz de usuario (UI) y la experiencia de usuario (UX) han sido consideraciones primordiales durante el desarrollo del frontend. El objetivo era crear una aplicación visualmente atractiva, fácil de usar y que transmitiera profesionalidad.

- **Principios de Diseño:**
  - **Modernidad y Simplicidad:** Se optó por un diseño limpio, con un uso adecuado del espacio en blanco para evitar la sobrecarga visual.
  - **Consistencia:** Se mantiene una coherencia visual en todos los elementos de la interfaz (botones, tipografía, colores, espaciado) tanto en la parte pública como en la privada. El header y footer son consistentes a lo largo de la navegación.
  - **Usabilidad:** Se priorizó la facilidad de navegación y la claridad en la presentación de la información. Los flujos de usuario, como el proceso de reserva o el check-in, se diseñaron para ser intuitivos.
  - **Responsividad:** Gracias a Bootstrap, la aplicación se adapta a diferentes tamaños de pantalla (escritorio, tablet, móvil).
- **Paleta de Colores:** Se utilizó una paleta de colores monocromática anaranjada. Este color principal se complementa con tonos neutros (blancos, grises, negros) para el texto y los fondos, creando un contraste adecuado y una identidad visual distintiva.

### 3.3.4. Backend (Symfony)

El backend de la aplicación se ha desarrollado utilizando el framework Symfony (versión 7.2.\*), sobre PHP (versión >=8.2). Proporciona una API RESTful que sirve como punto central para toda la lógica de negocio y la gestión de datos.

- **Tecnologías y Bundles Clave:**

- **Symfony Framework Bundle:** El núcleo del framework Symfony.
- **Doctrine ORM Bundle:** Para el mapeo objeto-relacional, permitiendo interactuar con la base de datos utilizando objetos PHP en lugar de escribir SQL directamente. Incluye doctrine/dbal (^3) para la capa de abstracción de base de datos y doctrine/doctrine-migrations-bundle (^3.4) para gestionar la evolución del esquema de la base de datos.
- **firebase/php-jwt:** Librería para la generación y validación de JSON Web Tokens.
- **NelmioCorsBundle:** Para gestionar las políticas de Cross-Origin Resource Sharing (CORS), esencial para que el frontend Angular (servido desde un origen diferente en desarrollo) pueda comunicarse con la API.
- **Symfony Serializer Component:** Para convertir objetos PHP a JSON (para las respuestas de la API) y viceversa (para los datos de las solicitudes).
- **Symfony Maker Bundle:** Herramienta de desarrollo para generar código boilerplate (controladores, entidades, formularios, etc.).

- **Estructura de la API:**

- **Controladores:** Clases PHP que reciben las peticiones HTTP, interactúan con los servicios y el ORM, y devuelven respuestas HTTP (generalmente JSON).
- **Entidades:** Clases PHP que representan las tablas de la base de datos, mapeadas por Doctrine.
- **Repositorios:** Clases que encapsulan la lógica para consultar las entidades de la base de datos.
- **Servicios:** Clases que contienen lógica de negocio reutilizable.

### Endpoints Principales (Resumen de api\_documentation.md):

- **Autenticación:**

- **POST /login:** Autentica al usuario y devuelve un JWT y datos del usuario.
- **POST /logout:** Invalida el JWT (requiere gestión de lista negra de tokens).
- **POST /register:** Crea un nuevo usuario (cliente).

- **Gestión de Usuarios (Empleados/Administradores):**

- **GET /users, GET /users/{id}, PUT /users/{id}, DELETE /users/{id}** (con roles protegidos).

- **Gestión de Habitaciones y Tipos de Habitación:**

- Endpoints CRUD para Room y RoomType.
- Endpoints para consultar disponibilidad.

- **Gestión de Reservas:**
  - POST /bookings (crear reserva, accesible por clientes autenticados o pública con validación).
  - GET /bookings, GET /bookings/{id}, PUT /bookings/{id} (para gestión interna, check-in/out, cancelación).
- **Gestión de Servicios Adicionales.**
  - (La api\_documentation.md proporciona una descripción exhaustiva de cada endpoint, los parámetros esperados, las respuestas y los códigos de estado HTTP).
- **Seguridad:**
  - Las rutas sensibles de la API están protegidas y requieren un JWT válido en la cabecera Authorization (Bearer token).
  - Se pueden definir roles (ROLE\_CLIENT, ROLE\_EMPLOYEE, ROLE\_ADMIN) para un control de acceso más granular a los diferentes endpoints, gestionado por Symfony Security.

El backend está diseñado para ser robusto, seguro y eficiente, proporcionando al frontend todos los datos y funcionalidades necesarias.

### 3.3.5. Base de Datos

La persistencia de los datos de la aplicación se gestiona mediante una base de datos relacional, con la que se interactúa a través del ORM Doctrine en el backend Symfony. El diseño del esquema de la base de datos es fundamental para el correcto funcionamiento de la aplicación.

- **Esquema Entidad-Relación (E-R):**

El esquema se ha diseñado para modelar las principales entidades y sus relaciones en el contexto de la gestión hotelera.

- Las entidades principales son:

- **user:** Almacena información de los usuarios, tanto clientes como personal del hotel. Incluye campos como id, name, surnames, email, password, role.
- **room\_type:** Define los diferentes tipos de habitaciones disponibles (ej. Individual, Doble, Suite).
- **room:** Representa una habitación física específica del hotel. Se relaciona con room\_type.
- **booking:** Registra las reservas realizadas por los clientes. Se relaciona con user y room.
- **service:** Define servicios adicionales que puede ofrecer el hotel (ej. Desayuno, Parking, Spa).
- **image:** Almacena información de las imágenes.
- **Tablas de Unión (Pivote):**
  - **room\_type\_image:** Relación muchos a muchos entre room\_type e image.

- **service\_image:** Relación muchos a muchos entre service e image.
- **booking\_service:** Relación muchos a muchos entre booking y service, para registrar los servicios contratados en una reserva.
- **token:** Almacena información sobre los JWTs, especialmente para gestionar su revocación (lista negra).
- **doctrine\_migration\_versions:** Tabla utilizada por Doctrine Migrations para rastrear las migraciones aplicadas.

(Referencia: Ver Anexo A para una representación visual del Esquema Entidad-Relación proporcionado en schema.png.)

- Consideraciones del Diseño:

- **Normalización:** Se ha buscado un grado adecuado de normalización para evitar la redundancia de datos y asegurar la integridad.
- **Relaciones:** Se han establecido claves foráneas para mantener la integridad referencial entre las tablas.
- **Indexación:** Aunque no se detalla aquí, en una implementación productiva se añadirían índices a las columnas frecuentemente usadas en búsquedas y uniones (ej. claves foráneas, campos de fecha en reservas) para optimizar el rendimiento de las consultas.
- **Tipos de Datos:** Se han seleccionado tipos de datos apropiados para cada campo.

Doctrine ORM abstrae la interacción directa con SQL, permitiendo definir estas entidades y sus relaciones como clases PHP.

Las migraciones de Doctrine facilitan la evolución controlada del esquema de la base de datos a medida que el proyecto avanza.

### 3.4. Problemas/dificultades encontradas:

Durante el desarrollo del proyecto, se presentaron algunos desafíos técnicos y de gestión:

#### 1. Problema: Integración de autenticación JWT entre partes pública y privada.

- **Descripción:** Asegurar un flujo de autenticación robusto y seguro con JSON Web Tokens, incluyendo la generación, almacenamiento seguro en el cliente (Angular), envío en cada solicitud protegida, validación en el servidor (Symfony), y gestión de la expiración y revocación de tokens.
- **Solución:**
  - En Symfony, se utilizó firebase/php-jwt para la codificación/decodificación de tokens y un sistema propio para integrar la lógica de autenticación.
  - En Angular, se crearon interceptores HTTP para adjuntar automáticamente el token JWT a las cabeceras de las peticiones salientes a la API. El token se almacenó en localStorage del navegador.
  - Se implementó una gestión centralizada de sesiones/tokens en el cliente para manejar el login, logout (invalidando el token en el cliente) y la redirección automática en caso de tokens expirados o inválidos.

## 2. Problema: Gestión del estado y la comunicación entre componentes en Angular.

- **Descripción:** A medida que la aplicación frontend crecía en complejidad, mantener un flujo de datos coherente y una comunicación eficiente entre componentes (especialmente entre componentes no relacionados jerárquicamente) se volvió un desafío.
- **Solución:** Se hizo un uso extensivo de Servicios de Angular para centralizar la lógica de negocio y el estado compartido. Para flujos de datos más complejos y eventos asíncronos, se utilizó RxJS con Observables y Subjects/BehaviorSubjects para permitir que los componentes se suscriban a cambios de estado o eventos.

### 3.5. Resultados obtenidos:

Al finalizar el desarrollo principal del proyecto, se han alcanzado los siguientes resultados:

- **Desarrollo funcional de ambas partes (pública y privada):**
  - La parte pública permite a los usuarios visualizar información del hotel, los tipos de habitación, sus precios y realizar reservas de manera efectiva.
  - La parte privada (dashboard) ofrece a los administradores y empleados del hotel las herramientas necesarias para gestionar reservas, clientes, y realizar operaciones de check-in y check-out, tras una autenticación segura.
- **Diseño atractivo y consistente con énfasis en la experiencia del usuario:**
  - Se ha logrado una interfaz de usuario moderna, intuitiva y fácil de navegar, siguiendo los principios de UX/UI definidos.
  - La aplicación es responsive, adaptándose a diferentes dispositivos (escritorio, tablet, móvil).
- **API RESTful funcional y segura:** El backend Symfony expone una API robusta que maneja la lógica de negocio y la persistencia de datos, con mecanismos de autenticación y autorización implementados.
- **Cumplimiento de Objetivos:** Se han cumplido la mayoría de los objetivos específicos planteados al inicio del proyecto, sentando una base sólida para futuras mejoras y expansiones.

## 4. Conclusiones

Este proyecto me ha permitido mejorar y consolidar habilidades en tecnologías web modernas como Angular para el desarrollo frontend y Symfony para el backend.

La implementación de una aplicación full-stack desde cero, abarcando desde el diseño de la base de datos hasta la interfaz de usuario y el despliegue, ha sido una experiencia de aprendizaje muy buena.

He profundizado en el diseño de aplicaciones, aplicando principios de arquitectura cliente-servidor, API RESTful, y patrones de diseño en ambos frameworks. La gestión de la autenticación con JWT y la configuración de un entorno de desarrollo y despliegue han sido aspectos particularmente complicados.

A nivel profesional, la experiencia en la aplicación (aunque simulada) de metodologías ágiles como SCRUM para la gestión del proyecto, la resolución de problemas técnicos complejos ha sido invaluable. Estos conocimientos y habilidades son directamente transferibles al entorno laboral actual, donde la agilidad, la eficiencia son cada vez más demandados.

El proyecto ha cumplido con los objetivos principales propuestos, entregando una solución funcional para la gestión hotelera. Si bien existen áreas de mejora y expansión, la base tecnológica y funcional establecida es sólida.

Personalmente, el proyecto ha reforzado la importancia de una buena planificación, la división de problemas complejos en tareas más manejables y la perseverancia en la resolución de obstáculos.

## 5. Líneas futuras de trabajo

Aunque el proyecto actual es funcional, existen diversas áreas donde podría expandirse y mejorarse en el futuro:

- **Soporte Multilingüe:**
  - Añadir soporte multilingüe para la interfaz pública, permitiendo a los usuarios seleccionar su idioma preferido. Esto implicaría el uso de librerías de internacionalización tanto en Angular como en Symfony.
- **Funcionalidad de Análisis de Datos y Reportes:**
  - Ampliar la funcionalidad del dashboard de administración con un módulo de análisis de datos y generación de reportes (ej. ocupación, ingresos por periodo, reservas por canal, etc.) para ayudar en la toma de decisiones estratégicas del hotel.
- **Integración con Pasarelas de Pago:**
  - Integrar la aplicación con pasarelas de pago online (ej. Stripe, PayPal) para permitir el pago de las reservas directamente a través de la web.
- **Gestión Avanzada de Habitaciones y Tarifas:**
  - Implementar una gestión más detallada del inventario de habitaciones (ej. bloqueo de habitaciones por mantenimiento).
  - Permitir la configuración de tarifas dinámicas (ej. por temporada, por día de la semana, ofertas especiales).
- **Módulo de Opiniones y Valoraciones:**
  - Permitir a los clientes dejar opiniones y valoraciones sobre su estancia, que podrían mostrarse (previa moderación) en la parte pública.

- **Mejoras en la Seguridad:**
  - Implementar autenticación de dos factores (2FA) para el acceso al panel de administración.
  - Realizar auditorías de seguridad periódicas.
- **Optimización del Rendimiento:**
  - Aunque se ha tenido en cuenta, realizar perfiles de rendimiento más exhaustivos tanto en el frontend (ej. lazy loading de módulos Angular) como en el backend (ej. optimización de consultas Doctrine, caching).
- **Pruebas Automatizadas:**
  - Ampliar la cobertura de pruebas unitarias y de integración en el backend (PHPUnit) y frontend.

Estas líneas futuras de trabajo podrían convertir la aplicación en una herramienta aún más completa y competitiva en el mercado de software de gestión hotelera.

## 6. Bibliografía

*Documentación de JWT.* (s.f.). Obtenido de <https://jwt.io/>

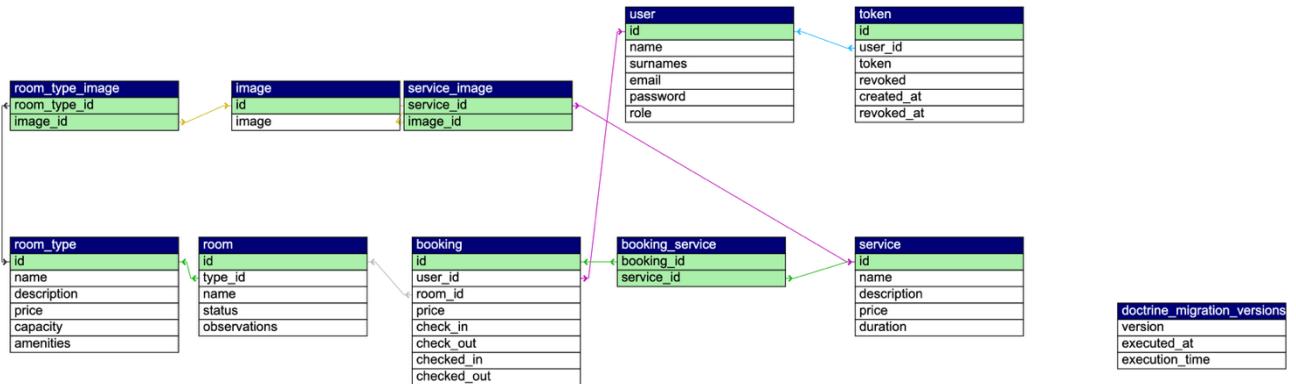
*Documentación de php-jwt.* (s.f.). Obtenido de  
<https://github.com/firebase/php-jwt>

*Documentación oficial de Angular.* (s.f.). Obtenido de  
<https://angular.dev>

*Documentación oficial de Bootstrap.* (s.f.). Obtenido de  
<https://getbootstrap.com/docs>

*Documentación oficial de Symfony.* (s.f.). Obtenido de  
<https://symfony.com/doc>

## ANEXO A: TABLA DE ENTIDADES Y RELACIONES BBDD



### Descripción de las Entidades Principales:

- **user**: Gestiona los datos de clientes y personal.
- **room\_type**: Define las categorías de habitaciones.
- **room**: Representa las habitaciones físicas.
- **booking**: Almacena los detalles de las reservas.
- **service**: Lista los servicios adicionales ofrecidos.
- **image**: Almacena referencias a imágenes.
- **token**: Utilizada para la gestión de tokens JWT (ej. revocación).
- Las tablas **room\_type\_image**, **service\_image**, **booking\_service** son tablas pivot para relaciones muchos-a-muchos.