UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería Informática

ElCapirote – App para la Gestión de Hermandades Cofrades

ElCapirote – App for Holy Week Brotherhoods Management

Realizado por
David Sarmiento Rincón

Tutorizado por
Francisco José Jaime Rodríguez

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Septiembre 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

ElCapirote – App para la Gestión de Hermandades Cofrades

ElCapirote – App for Holy Week Brotherhoods Management

Realizado por
David Sarmiento Rincón

Tutorizado por
Francisco José Jaime Rodríguez

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2025

Fecha defensa: septiembre de 2025

Abstract

This Project arises from the need to modernize the management of Holy Week brotherhoods, traditionally supported by outdated tools or inefficient manual processes. With this project, the aim has been to provide a digital solution in the form of a web application that centralizes and optimizes the administration of members, carriers, and payments, offering greater accessibility and reliability in the management of information.

To this end, the main objectives were: to digitalize the management of member data, automate economic operations, facilitate the control of processions and clothing loans, and provide configurable lists and templates adapted to the needs of each organization. The development has been carried out through an architecture based on *Spring Boot* and *Angular*, with an agile methodology (*SCRUM*) that allowed iterative validation of the system's progress.

The results achieved show that the project meets the initially defined requirements, providing a stable, usable application consistent with the identified needs. Furthermore, the experience has represented significant learning in the field of software engineering, strengthening both technical and methodological skills. Overall, this work demonstrates the feasibility and value of the proposed solution, while laying the foundations for future improvements that may extend its functionalities and respond to new demands of the sector.

Keywords: Holy Week Brotherhoods, Spring Boot, Angular, SCRUM, Web Application

Resumen

Este Trabajo Fin de Grado surge de la necesidad de modernizar la gestión de las cofradías y hermandades, tradicionalmente apoyada en herramientas obsoletas o en procesos manuales poco eficientes. Con este proyecto se ha buscado ofrecer una solución digital en forma de aplicación WEB que centralice y optimice la administración de miembros, horquilleros y pagos aportando mayor accesibilidad y fiabilidad en el tratamiento de la información.

Para ello se plantearon como objetivos principales: digitalizar la gestión de datos de miembros, automatizar operaciones económicas, facilitar el control de salidas procesionales y préstamos de vestuario, así como ofrecer listados y plantillas configurables adaptadas a las necesidades de cada organización. El desarrollo se ha abordado mediante una arquitectura basada en *Spring Boot* y *Angular*, con una metodología ágil (*SCRUM*) que ha permitido validar iterativamente los avances del sistema.

Los resultados alcanzados muestran que el proyecto cumple con los requisitos inicialmente definidos, proporcionando una aplicación estable, usable y coherente con las necesidades detectadas. Además, la experiencia ha supuesto un aprendizaje significativo en el ámbito de la ingeniería del software, afianzando competencias técnicas y metodológicas. En conjunto, este trabajo demuestra la viabilidad y el valor de la solución propuesta, a la vez que sienta las bases para futuras mejoras que permitan ampliar sus funcionalidades y responder a nuevas demandas del sector.

Palabras clave: Cofradías de Semana Santa, Spring Boot, Angular, SCRUM, Aplicación Web

Índice

1 Introducción

1.1. Motivación

La elección de este proyecto surge de una necesidad detectada en el ámbito de las cofradías y hermandades. En particular, en mi propia cofradía se utiliza actualmente un programa de gestión antiguo, poco usable y con un rendimiento limitado, que además supone un coste elevado para la organización. Esta situación genera constantes quejas entre los usuarios, ya que dificulta la eficiencia en la gestión administrativa y en la organización de actividades.

A ello se suma que otras cofradías presentan problemas similares o, en muchos casos, ni siquiera disponen de una aplicación informática, llevando a cabo gran parte de su gestión de manera manual y en papel. Esta falta de digitalización complica el acceso a la información, incrementa el riesgo de errores y limita la capacidad de coordinación.

Por todo ello, el desarrollo de una aplicación moderna, accesible y eficiente busca dar respuesta a esta problemática, proporcionando una herramienta que facilite y agilice la gestión integral de las cofradías. De este modo, el proyecto no solo responde a una necesidad personal y cercana, sino que también plantea una solución con potencial utilidad en un contexto más amplio en todas las cofradías y hermandades de España.

1.2. Objetivos

El presente proyecto tiene como finalidad dar respuesta a las necesidades identificadas en la gestión de las cofradías y hermandades, proporcionando una solución moderna y eficaz. Para ello, se plantean los siguientes objetivos principales:

- **Digitalizar la gestión de miembros y horquilleros**, permitiendo un control centralizado de sus datos personales, históricos y situación dentro de la cofradía.

- **Automatizar la gestión de pagos y cuotas**, facilitando el registro, control y consulta de operaciones económicas asociadas a los hermanos y a las actividades de la cofradía.
- **Gestionar tallajes, salidas procesionales y préstamos de vestuario**, reduciendo el uso de procesos manuales y minimizando los posibles errores derivados de la gestión en papel.
- **Proporcionar listados y plantillas configurables**, que permitan a los administradores disponer de información clara y estructurada en función de las necesidades de cada situación.
- **Garantizar la seguridad y la usabilidad del sistema**, mediante autenticación de usuarios, protección de datos sensibles y una interfaz accesible e intuitiva.
- **Seguir de forma rigurosa el proceso de ingeniería del software**, aplicando buenas prácticas de análisis, diseño, desarrollo y pruebas, de manera que el proyecto no solo resuelva una necesidad práctica, sino que también sirva como ejemplo completo del ciclo de vida de una aplicación software.

En conjunto, estos objetivos buscan dotar a las cofradías de una herramienta integral que optimice sus procesos administrativos, facilite el acceso a la información y mejore la organización global de la hermandad.

1.3. Estructura del documento

El presente documento se organiza en diferentes capítulos que siguen el flujo natural de un proyecto de ingeniería del software. Tras esta introducción, en el capítulo 2 se describen las tecnologías y herramientas seleccionadas, justificando su elección en función de las necesidades del proyecto. En el capítulo 3 se expone la metodología de desarrollo empleada, detallando la comparación entre distintos enfoques y la aplicación práctica de *Scrum*.

El capítulo 4 recoge de forma exhaustiva los requisitos del sistema, tanto funcionales como no funcionales y futuros, así como su tabla de trazabilidad. Posteriormente, en el capítulo 5 se presentan los casos de uso, incluyendo diagramas y especificaciones que reflejan las principales interacciones entre actores y sistema.

El capítulo 6 aborda el diseño de la aplicación mediante los distintos diagramas UML (clases y secuencia) y la realización de mockups, mientras que el capítulo 7 se centra en el desarrollo, diferenciando **backend** y **frontend**, y explicando la implementación de las funcionalidades clave junto con las pruebas realizadas.

Finalmente, el capítulo 8 reúne las conclusiones obtenidas y las posibles líneas futuras de evolución del sistema. Los apéndices complementan el contenido principal con el detalle completo de casos de uso, mockups y diagramas adicionales, así como el manual de usuario.

2

Tecnologías y Herramientas Utilizadas

En el desarrollo de este proyecto se ha empleado un conjunto de tecnologías y herramientas que cubren tanto la gestión de la memoria como el diseño, el desarrollo de la aplicación y la gestión de datos. A continuación, se describen las más relevantes, organizadas en dos bloques principales: **tecnologías**, que hacen referencia a los lenguajes, frameworks y motores empleados, y **herramientas**, que han apoyado la implementación, organización y documentación del proyecto.

2.1. Tecnologías

A continuación se enumerarán las tencologías, frameworks y motores de bases de datos usados para la realización de el proyecto, así como una justificación de porqué se han usado.

2.1.1. Angular

En el desarrollo del frontend se seleccionó el framework Angular, dado que ya había sido utilizado en asignaturas previas y en prácticas curriculares. Angular ofrece un entorno robusto para aplicaciones SPA (*Single Page Application*) y facilita el mantenimiento de código modular. Inicialmente se valoró el uso de Thymeleaf [1], aunque finalmente se optó por Angular por su mayor versatilidad y proyección profesional.

2.1.2. Java Spring Boot

Para el backend se empleó Spring Boot, un framework ampliamente utilizado en el desarrollo empresarial por su robustez, facilidad de uso y ecosistema integrado. Su elección se apoyó en la experiencia previa adquirida en asignaturas de la titulación. Aunque se consideraron alternativas como NestJS (utilizado en prácticas externas), se optó por Spring Boot por su integración nativa con Java y su madurez en proyectos de gran escala.

2.1.3. MySQL

Como motor de base de datos se optó por MySQL, principalmente por su facilidad de instalación y uso gracias a MySQL Workbench. Frente a otras alternativas como SQL Server u Oracle Database, MySQL destaca por su carácter abierto, documentación extensa y comunidad activa, lo que simplifica su adopción en proyectos académicos y profesionales.

2.1.4. Tailwind CSS y DaisyUI

En cuanto al diseño de la interfaz de usuario, se ha utilizado **Tailwind CSS**, un *framework* de utilidades que permite aplicar estilos de manera rápida y flexible mediante clases predefinidas, garantizando coherencia visual y facilidad de personalización. Sobre esta base se ha integrado **DaisyUI**, una librería de componentes que extiende Tailwind con elementos ya diseñados (botones, formularios, tarjetas, etc.), acelerando así el desarrollo de la interfaz y asegurando una experiencia de usuario moderna y consistente.

2.1.5. LaTeX

Se utilizó LaTeX como tecnología de composición tipográfica para la redacción de la memoria, garantizando una maquetación profesional, el uso de bibliografía integrada y una estructura modular que facilita el mantenimiento del documento.

2.1.6. Git

Para el control de versiones se ha empleado **Git**, herramienta que permite gestionar de manera distribuida el historial de cambios del proyecto, realizar ramificaciones (*branching*) y

combinar desarrollos (*merging*), ofreciendo una base sólida para el trabajo colaborativo y el versionado del código.

2.2. Herramientas

Así mismo, como en la subsección anterior, a continuación se expondrán todas las herramientas utilizadas en el proyecto para explotar al máximo las tecnologías seleccionadas.

2.2.1. Visual Studio Code

La edición y desarrollo del código se realizaron en Visual Studio Code, un entorno de desarrollo versátil y altamente personalizable, con soporte para múltiples lenguajes y extensiones. La elección se basó en su flexibilidad y la posibilidad de integrar fácilmente plugins para frameworks como Angular y Spring Boot.

2.2.2. Overleaf

Para la redacción de la memoria se utilizó Overleaf, un editor en línea de LaTeX que facilita la edición colaborativa y la gestión de proyectos académicos. Su uso permitió mantener una estructura ordenada del documento y agilizar la compilación en diferentes fases del trabajo.

2.2.3. Visual Paradigm

El modelado de los diagramas UML se realizó con Visual Paradigm, herramienta seleccionada principalmente porque la Universidad de Málaga ofrece licencias académicas gratuitas. Además, se trata de un software ampliamente utilizado en el ámbito docente y profesional, lo que facilitó la elaboración de los diagramas de casos de uso, clases y secuencia con un entorno conocido y eficaz.

2.2.4. Figma

Para el diseño del *mockup* de la aplicación se empleó Figma, una herramienta moderna y colaborativa que permite crear prototipos de alta fidelidad en línea [2]. La decisión de usar Figma se tomó frente a alternativas como Pencil Project [3], ya que ofrece una interfaz más

moderna, una mayor integración con flujos de trabajo actuales y facilidades de colaboración en tiempo real.

2.2.5. MySQL Workbench

La administración de la base de datos se realizó con MySQL Workbench, interfaz gráfica que simplifica la creación de esquemas, consultas y gestión de datos, complementando al motor MySQL.

2.2.6. GitHub

Como repositorio remoto se utilizó **GitHub**, complementando a Git con funcionalidades avanzadas como la gestión de ramas, *pull requests*, control de incidencias y soporte para integración continua, esenciales para coordinar de forma eficiente las distintas iteraciones del proyecto.

2.2.7. Trello

La gestión de tareas y planificación de los *sprints* se realizó con Trello, herramienta de gestión ágil basada en tableros Kanban. Su uso permitió organizar de manera visual el flujo de trabajo, asignar tareas y monitorizar el avance del proyecto.

3

Metodología de Desarrollo

Para el desarrollo de este proyecto se ha optado por una metodología ágil, concretamente **Scrum**, ya que se adapta mejor a las características y necesidades de un Trabajo Fin de Grado (TFG).

Según los apuntes de la asignatura de Ingeniería del Software, los procesos de desarrollo pueden clasificarse en metodologías tradicionales y metodologías ágiles [4]. Cada una presenta ventajas e inconvenientes dependiendo del contexto del proyecto, el tamaño del equipo y el grado de definición de los requisitos.

3.1. Comparación entre metodologías

El **modelo en cascada**, uno de los más antiguos y utilizados en entornos industriales, se caracteriza por dividir el desarrollo en fases secuenciales: análisis, diseño, implementación, pruebas y mantenimiento. Su principal ventaja es la claridad en la planificación y la documentación exhaustiva. Sin embargo, resulta rígido, ya que apenas permite volver a fases anteriores sin un alto coste. Para un TFG, donde los requisitos pueden evolucionar y algunas decisiones se van tomando de forma progresiva, esta falta de flexibilidad lo hace poco adecuado.

Por otra parte, modelos iterativos como el **modelo incremental** o el **modelo en espiral** ofrecen mayor adaptabilidad. El incremental permite entregar versiones parciales del sistema en cada iteración, mientras que el espiral se centra en la gestión del riesgo. Ambos son más versátiles que el cascada, pero suelen implicar una mayor complejidad de gestión y planificación, lo cual no es del todo eficiente en un proyecto académico individual.

En contraste, los **métodos ágiles** y, en particular, **Scrum**, ponen el énfasis en la *entrega continua de valor*, en la *colaboración* con el cliente y en la capacidad de adaptación al cambio.

A diferencia del cascada, Scrum no requiere que todos los requisitos estén perfectamente definidos desde el inicio, y frente al incremental o espiral, su marco de trabajo es más sencillo y ligero, lo que lo hace muy apropiado para un TFG desarrollado en solitario.

3.2. Justificación de la elección de Scrum

La elección de Scrum en este proyecto se fundamenta en varias razones:

- **Flexibilidad**: al tratarse de un TFG, es habitual que durante el desarrollo surjan nuevas ideas o se redefinan requisitos. Scrum, con sus iteraciones cortas, permite incorporar estos cambios sin afectar al proyecto completo.

- **Gestión del tiempo**: las iteraciones (sprints) de dos semanas permiten planificar objetivos concretos, facilitando un control claro del avance.

- **Simplicidad**: frente a otros modelos iterativos más complejos, Scrum ofrece un marco ligero que se adapta bien a un desarrollo individual.

- **Entrega incremental**: cada sprint proporciona un incremento funcional del sistema, lo que asegura disponer de un prototipo en funcionamiento desde las primeras fases.

- **Motivación y productividad**: la división en sprints con objetivos definidos ayuda a mantener la motivación y a enfocar los esfuerzos en tareas alcanzables.

3.3. Aplicación práctica de Scrum en el proyecto

La metodología se implementó mediante un **tablero Kanban en Trello**, que recoge tanto el *product backlog* como los distintos *sprint backlog*. Cada sprint tuvo una duración aproximada de dos semanas y se centró en un conjunto de tareas específicas (por ejemplo: especificación de requisitos, elaboración de diagramas UML, implementación del backend, pruebas, etc.).

En las Figuras 1, 2 y 3 se muestran capturas del tablero utilizado, donde se reflejan los diferentes sprints con sus entregables correspondientes.

Figura 1: Tablero de Trello con los primeros sprints del proyecto (Backlog, Sprint 1, Sprint 2, Sprint 3).

Figura 2: Tablero de Trello con los sprints intermedios (Sprint 4 a Sprint 7).

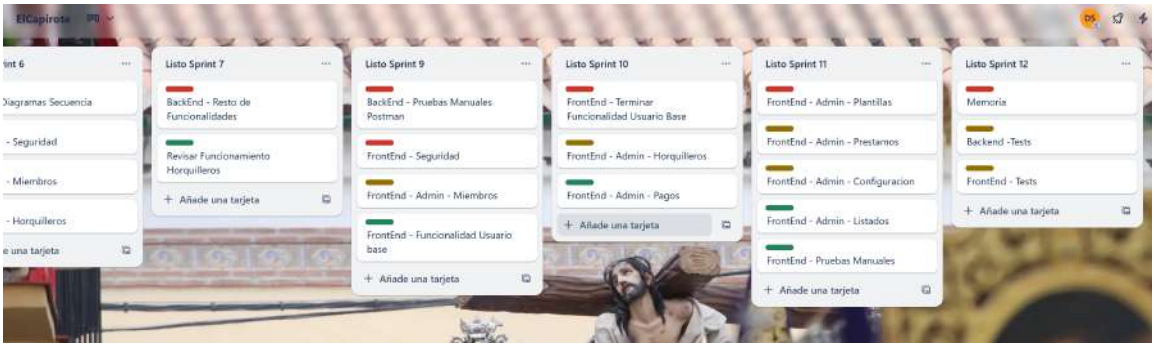


Figura 3: Tablero de Trello con los últimos sprints del proyecto (Sprint 9 a Sprint 12).

Teniendo en cuenta todo lo expuesto anteriormente, podemos decir que la aplicación de Scrum permitió:

- Mantener un control claro de las tareas en curso y las finalizadas.
- Priorizar funcionalidades esenciales antes de abordar requisitos secundarios.
- Adaptar progresivamente el desarrollo a nuevas ideas y cambios surgidos durante el proyecto.
- Garantizar la entrega de un producto funcional en todas las fases del desarrollo.

4

Requisitos

En ingeniería del software, un **requisito** se entiende como una función, característica, condición, capacidad o información que el sistema debe realizar, satisfacer o gestionar [5]. Los requisitos constituyen la base para el diseño, la implementación y la validación del sistema, al capturar de forma estructurada las necesidades de las partes interesadas y los objetivos del proyecto [5, 6].

Clásicamente se distinguen las siguientes categorías:

- **Requisitos funcionales (RF)**: describen los servicios, comportamientos y operaciones que el sistema debe proporcionar (qué hace el sistema).
- **Requisitos no funcionales (RNF)**: establecen criterios de calidad y restricciones sobre cómo deben cumplirse los RF (rendimiento, seguridad, usabilidad, tolerancia a fallos, compatibilidad, etc.).
- **Requisitos de información**: especifican la información que el sistema debe gestionar (tipos de datos, volúmenes, integridad y consistencia). En este trabajo se integran dentro de RF o RNF según corresponda [5].
- **Requisitos futuros (RFF)**: funcionalidades deseables para versiones posteriores; no forman parte del alcance inicial, pero orientan la evolución del sistema.

De acuerdo con las buenas prácticas de la ingeniería de requisitos, estos deben ser claros, completos, no ambiguos, verificables y, en la medida de lo posible, atómicos; además, conviene dotarlos de un identificador único y mantener su trazabilidad con casos de uso y pruebas [6].

A continuación enumerarán por orden ascendente la lista de requisitos de la aplicación a desarrollar. Para tener una mayor contextualización , dentro de la aplicación habrá tres roles:

Rol	Permisos principales
Usuario sin registrar	Acceso únicamente a la pantalla de inicio de sesión. No puede usar ninguna otra funcionalidad.
Usuario registrado	Acceso a su perfil y a la consulta/edición de sus datos personales (según permisos).
Administrador	Acceso completo: gestión de miembros, tronos, salidas procesionales, horquilleros, tallajes, pagos, préstamos, listados e interfaz de configuración.

Cuadro 1: Roles y alcance de permisos

4.1. Requisitos Funcionales

RF1: Iniciar Sesión

El sistema permitirá a todo usuario iniciar sesión en la aplicación. Para ello, se le requerirá la dirección de correo electrónico y la contraseña. Durante el proceso de inicio de sesión, el sistema llevará a cabo una comprobación de credenciales, en la cual se comprobarán los datos introducidos. Si son correctos, se permitirá el inicio, y si no lo son, se mandará un mensaje de error.

RF2: Recuperar Contraseña

Si un usuario no logra acceder a la plataforma, podrá acceder a la funcionalidad de cambiar contraseña. El sistema le pedirá su correo electrónico. Tras introducirlo y la aplicación comprueba que existe una cuenta con dicha dirección, se le enviará un correo con un código auto-generado por el sistema. Una vez el usuario introduce el código en la aplicación, se le permitirá cambiar la contraseña.

RF2.1: Comprobar Contraseña

Durante el proceso de registro, el sistema llevará a cabo una comprobación de contraseña, en la cual comprobará si es suficientemente segura. Para que sea segura, una contraseña deberá tener al menos 8 caracteres, un carácter minúscula, otro mayúscula, y un número. Si la contraseña es apta, se permitirá el registro.

RF3: Modificar Datos Personales

La aplicación permitirá al usuario registrado editar sus datos dentro de la aplicación, tales como dirección física, dirección de correo, teléfono, etc. Otros datos, como puede ser el número de la túnica de préstamo, no se permitirá su edición.

RF4: CRUD Miembros

La aplicación deberá permitir al usuario administrador hacer CRUD *(create, read, update, delete)* de los miembros de la cofradía. Con miembros se entienden hermanos, penitentes [1], mantillas, miembros de la junta, etc.

Se agrupan todos estos requisitos en uno sólo para facilitar la lectura y comprensión del documento.

RF4.1: Crear Miembro

La aplicación permitirá crear un nuevo miembro. Para ello, le requerirá los siguientes datos: *nombre, apellidos, DNI, dirección de correo, dirección física, número de teléfono y antigüedad.* Si la antigüedad se dejase vacía, se asignará el año actual. También se le pedirá al usuario confirmar si el nuevo miembro será hermano, o simplemente miembro de la cofradía. Si se confirma que es nuevo hermano, se generará automáticamente la cuota de hermano perteneciente a ese año. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF4.2: Leer Miembro

La aplicación permitirá al usuario administrador ver una lista de todos los miembros de la base de datos. Si se quisiera ver datos de un miembro en específico, presionando sobre el nombre, nos saldrán los datos del mismo.

RF4.3: Editar Miembro

La aplicación permitirá al usuario administrador modificar los datos de un miembro en específico. Una vez se presiona en su nombre dentro de la lista, y se abrirá su ficha de miembro. Para editarla, el usuario presionará el botón *editar* y permitirá la edición de los datos. Podrá editar tanto datos como el teléfono, o marcar

[1] El Salida Procesional es un participante en las procesiones de Semana Santa que acompaña a los tronos vistiendo túnica y capirote, como signo de recogimiento y penitencia.

si el Miembro pasa a ser Hermano o viceversa. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF4.4: Dar de Baja a un Miembro

La aplicación permitirá dar de baja a un miembro. Dentro de la ficha aparecerá un botón para dar de baja a dicho miembro. Una vez se pulse, se pedirá el motivo de la baja y se guardará la fecha exacta de la baja. Este estado de baja hará que la ficha siga en la base de datos, la ficha se pueda buscar en la vista de miembros, pero que el miembro no salga en los listados que el usuario administrador obtenga de la aplicación. Una vez se dé de baja al miembro, se actualizará la base de datos con los nuevos datos especificados.

RF5: Registrar Usuario

El usuario administrador podrá registrar como usuario a cualquier miembro de la cofradía. Para ello, existirá un botón de *Registrar*. Una vez pulsado el botón, se habilitará el correo electrónico del usuario para que este pueda acceder a la aplicación. Una vez se de de alta a un usuario, le llegará a su correo electrónico las instrucciones para usar la aplicación. Para poder entrar a la aplicación, los usuarios deberán crear una nueva contraseña a través de la funcionalidad de *Recuperar Contraseña*, (RF2).

RF6: CRUD Salida Procesional

La aplicación deberá permitir a un administrador hacer CRUD *(create, read, update, delete)* de los penitentes de cada sección de la cofradía (Cristo y Virgen).

RF6.1: Crear Salida Procesional

La aplicación permitirá crear una nueva Salida Procesional al usuario administrador. Tras buscar a el miembro en cuestión, se elegirá el trono y la sección del cortejo procesional en la que participará. Además se le podrá asignar un número de túnica, capirote y capa si la llevase, para llevar un control sobre el inventario. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF6.2: Leer Salidas Procesional

La aplicación permitirá al usuario obtener una lista con todas las salidas procesionales existentes en la base de datos del miembro seleccionado.

RF6.3: Editar Salida Procesional

La aplicación permitirá editar la ficha de un Salida Procesional en concreto. Pudiendo editar tanto los datos de miembro, como los datos de Salida Procesional, como el puesto o túnica Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF6.4: Eliminar Una Salida Procesional

La aplicación permitirá a el usuario administrador eliminar una Salida Procesional. Esto se hará seleccionando la línea en la que se refleja la salida a eliminar, y presionando sobre el botón color rojo *Eliminar Salida*

RF7: CRUD Horquillero [2]

La aplicación deberá permitir a un administrador hacer CRUD *(create, read, update, delete)* de los horquilleros de los tronos de la cofradía.

RF7.1: Crear Horquillero

La aplicación permitirá crear un nuevo horquillero al usuario administrador, siguiendo dos opciones. Pudiendo recuperar datos de los miembros de la cofradía, o simplemente creando un nuevo miembro. El proceso para crear un nuevo horquillero a partir de un miembro se basará en buscar un miembro a partir de su nombre y apellidos, seleccionarlo, y automáticamente se recuperarán los datos, lo que creará una relación entre el trono y el miembro (horquillero). Si se opta por crear un nuevo miembro, el formulario para crear la ficha será el mismo que en la vista de miembros. Esta opción también creará una relación entre el trono y el miembro (horquillero). Una vez tengamos la ficha de horquillero con los datos, podremos asignarle una talla al horquillero, su hombro de preferencia, y también habrá una sección de observaciones a tener en cuenta. Una vez

[2]En la Semana Santa de Vélez-Málaga, se denomina **Horquillero** al portador de un trono procesional. El término tiene su origen en la horquilla de madera utilizada tradicionalmente para apoyar el trono en los descansos de la procesión. Aunque en otros puntos de Málaga se usa Hombre de Trono, en Vélez-Málaga se sigue usando como parte de su historia cultural y cofrade.

introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF7.2: Leer Horquillero

La aplicación permitirá al usuario obtener una lista con todos los horquilleros del trono seleccionado. Para ver la ficha de un horquillero en concreto se presionará en su nombre, y se abrirá la ficha. Se podrá buscar a un horquillero en concreto con su nombre y apellidos, para una mayor fluidez en el uso de la aplicación.

RF7.3: Editar Horquillero

La aplicación permitirá editar la ficha de un horquillero en concreto. Pudiendo editar tanto los datos de miembro, como los datos de horquillero, como la talla, o el puesto auto-asignado con el algoritmo. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF7.4: Dar de Baja a un Horquillero

La aplicación permitirá a el usuario administrador dar de baja a un horquillero. Dentro de la ficha aparecerá un botón para dar de baja a dicho miembro. Una vez se pulse, se pedirá el motivo de la baja y se guardará la fecha exacta de la baja. Este estado de baja hará que la ficha siga en la base de datos, la ficha se pueda buscar en la vista de miembros, pero que el miembro no salga en los listados que el usuario administrador obtenga de la aplicación. Una vez se dé de baja al miembro, se actualizará la base de datos con los nuevos datos especificados.

RF8: CRUD Tallaje La aplicación deberá permitir a un administrador hacer CRUD *(create, read, update, delete)* de los tallajes de los horquilleros de la cofradía.

RF8.1: Crear Tallaje

La aplicación permitirá crear un nuevo tallaje al usuario administrador. Para ello tendrá que seleccionar el horquillero, y más tarde añadir los datos del tallaje, como puede ser la altura del horquillero. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF8.2: Leer Tallaje

La aplicación permitirá ver una lista con todos los tallajes, y además podrá ver el histórico de tallajes de cada miembro.

RF8.3: Editar Tallaje

La aplicación permitirá editar la ficha de un horquillero en concreto. Pudiendo editar tanto los datos de miembro, como los datos de horquillero, como la talla, o el puesto auto-asignado con el algoritmo. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF8.4: Eliminar Tallaje

La aplicación permitirá a el usuario administrador dar de baja a un horquillero. Dentro de la ficha aparecerá un botón para dar de baja a dicho miembro. Una vez se pulse, se pedirá el motivo de la baja y se guardará la fecha exacta de la baja. Este estado de baja hará que la ficha siga en la base de datos, la ficha se pueda buscar en la vista de miembros, pero que el miembro no salga en los listados que el usuario administrador obtenga de la aplicación. Una vez se dé de baja al miembro, se actualizará la base de datos con los nuevos datos especificados.

RF9: Plantilla Horquilleros.

La aplicación proporcionará al usuario con rol de administrador un botón denominado *Plantilla*, a través del cual podrá visualizar una representación gráfica del trono. Dicha representación mostrará un número de varales [3] acorde con la configuración establecida, así como una cantidad de cuadrículas en cada varal determinada por los parámetros configurados. Estas cuadrículas se completarán progresivamente a medida que se asignen los puestos a los horquilleros.

RF9.1: Auto-asignación de Horquilleros

La aplicación implementará un algoritmo para la asignación automática de puestos a los horquilleros. Cada horquillero será asignado, en primer lugar, al mismo varal que ocupó el año anterior y, posteriormente, se ordenará dentro de dicho varal según su talla. En el caso de los horquilleros de nueva incorporación, serán ubicados en el varal con menor cantidad de personal, procurando respetar, en

[3] En un trono de Semana Santa, el **Varal** es una barra metálica a que sobresale del trono y sirve para que los horquilleros lo carguen sobre sus hombros.

la medida de lo posible, su preferencia en cuanto al hombro de carga.

RF10: Plantilla Cortejo

La aplicación permitirá al usuario administrador, mediante un botón de *Plantilla* ver una plantilla con los participantes del cortejo procesional.

RF10.1: Auto-asignación

La aplicación permitirá, mediante un algoritmo, asignar en orden los puestos de penitentes dentro de cada sección. Se ordenarán por año de nacimiento, y más tarde por antigüedad en la cofradía del miembro.

RF11: CRUD Pagos

La aplicación deberá permitir a un administrador hacer CRUD *(create, read, update, delete)* de los pagos realizados.

RF11.1: Crear Pago

La aplicación permitirá al usuario crear un nuevo pago. A este nuevo pago se le podrá asociar un miembro de la cofradía, no siendo obligatorio. Además se requerirá la cantidad de donativo a abonar, y una breve descripción para identificar el pago. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF11.2: Leer Pago

La aplicación permitirá al usuario leer todos los pagos realizados en el historial del uso de la aplicación. Se podrá buscar un pago en concreto con el nombre y apellidos del miembro asociado, por la descripción, o por fecha, todo ello para una mayor fluidez en el uso de la aplicación.

RF11.3: Editar Pago

La aplicación permitirá editar un pago en concreto. Pudiendo editar tanto los datos de miembro, como los datos de cobro. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF11.4: Eliminar Pago

La aplicación permitirá al usuario administrador eliminar los datos de un pago

de la base de datos. Para ello, dentro de la ficha del pago habrá un botón de *Eliminar*. Para poder hacer efectiva la acción, pedirá una confirmación con la contraseña de acceso del usuario administrador. Una vez confirmado, se actualizará la base de datos.

RF12: Arqueo Diario de Caja

La aplicación permitirá al usuario administrador sacar automáticamente una suma detallada con todos los pagos en la fecha seleccionada, que por defecto se mantendrá en el día activo. Así se facilitaría el cierre de las cuentas de cada día.

RF13: CRUD Préstamo Ropa

La aplicación deberá permitir a un administrador hacer CRUD *(create, read, update, delete)* de prestamos de ropa.

RF13.1: Crear Préstamo

La aplicación permitirá al usuario crear un nuevo préstamo. A este nuevo pago se le tendrá asociar un miembro de la cofradía. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF13.2: Leer Préstamo

La aplicación permitirá al usuario leer todos los préstamos realizados en el historial del uso de la aplicación. Se podrá buscar un pago en concreto con el nombre y apellidos del miembro asociado o por fecha, todo ello para una mayor fluidez en el uso de la aplicación.

RF13.3: Editar Préstamo

La aplicación permitirá editar un préstamo en concreto. Pudiendo editar los datos de la ropa prestada. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF13.4: Devolución Préstamos

La aplicación permitirá al usuario administrador entrar en la vista de inventario. En ella podrá ver una lista de todas las túnicas prestadas y a qué miembro se le fue prestada. Si se presiona sobre una túnica, aparecerán los datos del miembro y un botón de *Devolución*, que eliminará este préstamo del sistema.

RF13.5: Devolución Préstamos Masiva

En la vista de Inventario habrá un botón de devolución masiva que eliminará todos los préstamos activos del sistema. Esto se hace ya que algunas cofradías se quedan con las túnicas prestadas en el mismo encierro, y otros miembros las devuelven en las siguientes semanas. Al existir las dos opciones, se facilita el uso de la aplicación.

RF14: Listados

La aplicación permitirá al usuario administrador sacar listados según una serie de parámetros configurables mediante listas desplegables. Podrá elegir si sacar el listado de entre miembros, hermanos, horquilleros, etc. Podrá elegir qué datos quiere que aparezcan por cada miembro, por ejemplo, sólo nombre y apellidos y número de teléfono, o si es Salida Procesional, que también salga el puesto asignado.

RF15: Configuración

La aplicación permitirá al usuario administrador acceder a la configuración de la aplicación.

RF15.1: Configurar Cuotas

El usuario podrá configurar las cuotas predeterminadas de hermano, de Salida Procesional o de horquillero.

RF15.2: Limpieza de Bajas

La aplicación permitirá a el usuario administrador generar una petición al sistema que hará que los miembros dados de baja durante los años definidos por el usuario administrador se eliminarán definitivamente del sistema, independientemente de que sean hermanos, penitentes, horquilleros, etc. También quedarán eliminados los prestamos que fueran devueltos anteriormente a los años indicados por el usuario administrador.

RF15.3: Hacer Visible Puestos

El usuario administrador podrá hacer visible a cada uno de los usuarios el puesto que tienen asignado dentro del trono, o de la sección de la procesión. Estos datos pasarán a estar visibles para los usuarios en su perfil.

RF15.4: Hacer Invisible Puestos

El usuario administrador podrá hacer invisible a cada uno de los usuarios el puesto que tienen asignado dentro del trono, o de la sección de la procesión. Estos datos pasarán a estar invisibles para los usuarios en su perfil.

RF16: CRUD Sección Penitentes

La aplicación deberá permitir a un administrador hacer CRUD *(create, read, update, delete)* de los sección de los penitentes de cada sección de la cofradía (Cristo y Virgen).

RF16.1: Crear Sección Penitentes

El usuario podrá añadir nuevos sección a la sección que sea necesario. Para ello habrá un botón de *Crear Cargo*. Se abrirá un formulario en el que le pedirá que sección pertenece el puesto (Cristo o Virgen), el nombre del puesto, y el número máximo de penitentes asignables. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF16.2: Leer Sección Penitentes

El usuario administrador, dentro de la configuración, tendrá la opción de leer todos los sección existentes para los penitentes.

RF16.3: Editar Sección Penitentes

El usuario administrador tendrá la opción de modificar un cargo. Pudiendo modificar nombre o el número máximo. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF16.4: Eliminar Sección Penitentes

El usuario administrador podrá eliminar un cargo en concreto de los existentes. Para ello tendrá que entrar en la ficha del cargo, en la que habrá un botón de *Eliminar*. Para poder hacer efectiva la acción, pedirá una confirmación con la contraseña de acceso del usuario administrador. Una vez confirmado, se actualizará la base de datos.

RF17: CRUD Tronos

La aplicación deberá permitir a un administrador hacer CRUD *(create, read, update, delete)* de los tronos de la cofradía, que por defecto serán Cristo y Virgen.

RF17.1: Crear Trono

El usuario administrador podrá añadir un nuevo trono si es necesario. Para ello habrá un botón de *Crear Cargo*. Se abrirá un formulario en el que le pedirá que sección pertenece el puesto (Cristo o Virgen), el nombre del puesto, y el número máximo de penitentes asignables. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF17.2: Leer Tronos

El usuario administrador, dentro de la configuración, tendrá la opción de leer todos los tronos existentes en la base de datos.

RF17.3: Editar Tronos

El usuario administrador tendrá la opción de modificar trono. Pudiendo tanto configurar el número de varales del trono, cómo las personas que entran en cada varal. Una vez introducidos los datos, se presionará en *guardar*, lo que actualizará la base de datos.

RF17.4: Dar de Baja un Trono

El usuario administrador podrá dar de baja un trono en concreto de los existentes. Para ello tendrá que entrar en la ficha del trono, en la que habrá un botón de *Baja*. Para poder hacer efectiva la acción, pedirá una confirmación con la contraseña de acceso del usuario administrador. Una vez confirmado, se actualizará la base de datos.

4.2. Requisitos No Funcionales

RNF1: Usabilidad

Para poder realizar cualquier acción dentro de la aplicación, como podría ser crear un horquillero, o gestionar cobros, sólo se necesitarán 5 clics de ratón.

RNF2: Seguridad

La aplicación seguirá unos estándares de seguridad altos. Para ello se implementarán funciones como la ya mencionada en el **RF2.2**, y otras como las siguientes. *(Los siguientes requisitos son funcionales, pero se sigue la estructura a la que pertenecen, es decir RNF2.* para que tengan identificador único.*

RNF2.1: Trato de contraseñas

Las contraseñas no serán guardadas como texto en claro (*RAW*), sino que se guardará un Hash de la contraseña, lo que aumenta la seguridad en caso de filtración de datos.

RNF2.2: Trato de datos personales

La aplicación guardará los datos personales en la base de datos de forma cifrada, para así tener una mayor seguridad. Otros datos no personales no serán cifrados, para así optimizar el uso de recursos.

RNF3: Escalabilidad

El sistema debe permitir el crecimiento en el número de usuarios, datos y cofradías sin necesidad de rediseñar su arquitectura, manteniendo su estabilidad y rendimiento a medida que evolucione.

RNF4: Documentación

El sistema debe contar con documentación técnica (código comentado, API Swagger/OpenAPI, diagramas de arquitectura y diseño) y funcional (manual de usuario), de forma que se facilite su mantenimiento y evolución futura.

4.3. Requisitos Opcionales (RFO)

RFO1: Listados avanzados (constructor tipo SQL)

El sistema ofrecerá un *constructor de consultas* que permita seleccionar tabla(s), campos, condiciones, ordenaciones y agrupaciones con una interfaz visual, emulando la potencia de una sentencia SQL pero sin necesidad de escribir código. *Este requisito opcional ha sido implementado*, ampliando la visión inicial de la aplicación y posibilitando exportar los resultados en varios formatos (CSV, JSON y PDF).

RFO2: Arqueo ampliado por rangos de fechas

El arqueo diario se extenderá para permitir cierres y resúmenes filtrados por un intervalo de fechas, con subtotales por tipo de pago y estado (pendiente/pagado). *Este requisito opcional también ha sido implementado*, reforzando la utilidad contable del sistema frente a la idea original centrada solo en el día en curso.

RFO3: Notificaciones y exportaciones programadas

El sistema permitirá programar la generación y el envío automático de determinados listados o arqueos (por ejemplo, un resumen semanal/mensual) a correos designados o a un repositorio interno de documentos, con historial de ejecuciones y registros de entrega.

Es importante destacar que los dos primeros requisitos opcionales (RFO1 y RFO2) han sido finalmente implementados durante el desarrollo. Esto ha permitido ampliar la visión inicial de la aplicación, proporcionando un sistema mucho más potente y flexible tanto en la generación de listados como en la gestión económica. Gracias a estas mejoras, la aplicación no solo cumple con los objetivos planteados en un principio, sino que también ofrece un valor añadido que facilita la labor de administración y refuerza la utilidad práctica del sistema en escenarios reales de uso.

4.4. Requisitos Futuros (RFF)

RFF1: Uso de Fotos de Perfil

La aplicación permitirá a el usuario subir fotos para usarlas de perfil, y así queda reflejada tanto para él mismo, como para los usuarios administradores.

RFF2: Gráficas de Cuentas

La aplicación permitirá a el usuario administrador obtener gráficas detalladas con los datos del dinero recaudado registrado en el sistema.

RFF3: Listados por Defecto

La aplicación traerá a petición de cada cofradía unos listados por defecto, los que más necesiten, para ahorrar tiempo a la hora de realizar consultas.

RFF4: ChatBot LLM

La aplicación permitirá a el usuario administrador poder establecer una conversación con un ChatBot LLM que le podrá ayudar en todo lo necesario.

RFF5: Pago desde la aplicación

La aplicación permitirá, en el futuro, realizar determinados pagos de forma telemática,

como la **cuota de hermano** o la **cuota de salida procesional**, como por ejemplo de las mantillas. De este modo, se evitará la necesidad de acudir físicamente a la sede de la cofradía, ya que dichos pagos podrán efectuarse directamente desde la aplicación mediante tarjeta bancaria.

RFF6: Interfaz SuperAdministrador

La aplicación tendrá, en el futuro, una interfaz de **superadministrador** que permitirá gestionar múltiples cofradías desde un único panel. A través de esta interfaz se podrán crear nuevas cofradías, dar de alta administradores asociados a cada una de ellas y supervisar la configuración general del sistema. De este modo, la aplicación podrá evolucionar hacia una solución multi–cofradía, escalable y adaptable a distintas hermandades.

4.5. Tabla de trazabilidad de requisitos

La tabla de trazabilidad de requisitos tiene como finalidad ofrecer una visión resumida y organizada de todos los requisitos identificados en el proyecto. De esta manera, se facilita su consulta rápida, clasificación y verificación, asegurando que no se omita ninguno en el proceso de desarrollo. Asimismo, permite disponer de un control claro entre los requisitos funcionales, no funcionales y futuros, lo que contribuye a mantener la coherencia del sistema y a planificar de forma más eficiente las posibles ampliaciones de la aplicación.

ID	Descripción breve
RF1	Iniciar sesión
RF2	Recuperar contraseña (incluye política de fortaleza)
RF3	Modificar datos personales
RF4	CRUD Miembros (crear, leer, editar, baja)
RF5	Registrar usuario (habilitar acceso)
RF6	CRUD Salida procesional
RF7	CRUD Horquilleros
RF8	CRUD Tallajes
RF9	Plantilla de horquilleros (autoasignación)
RF10	Plantilla de cortejo (autoasignación)

RF11	CRUD Pagos
RF12	Arqueo diario de caja
RF13	CRUD Préstamo de ropa (incluye devoluciones y masiva)
RF14	Listados configurables
RF15	Configuración (cuotas, limpieza de bajas, visibilidad de puestos)
RF16	CRUD Secciones de penitentes
RF17	CRUD Tronos
RNF1	Usabilidad: ≤ 5 clics por operación típica
RNF2	Seguridad (hash de contraseñas, cifrado de datos personales)
RNF3	Escalabilidad: nuevas cofradías, más usuarios y datos
RNF4	Documentación: documentación técnica y manual de usuario
RFO1	Listados Avanzados
RFO2	Arqueo Ampliado
RFO3	Notificaciones y exportaciones
RFF1	Fotos de perfil
RFF2	Gráficas de cuentas
RFF3	Listados por defecto
RFF4	ChatBot LLM
RFF5	Pago Telemático
RFF6	Interfaz de superadministrador para gestionar múltiples cofradías y crear nuevos administradores.

Cuadro 2: Resumen y trazabilidad de requisitos

5

Casos de Uso

Los *casos de uso* capturan los requisitos funcionales del sistema, guían el desarrollo, facilitan la comunicación entre usuarios, analistas y diseñadores, y presentan una visión del sistema como *caja negra*. Además, sirven para validar la arquitectura y son punto de partida para la generación de casos de prueba [7].

Un diagrama de casos de uso está compuesto por **actores**, **casos de uso** y **relaciones** entre ellos. Un caso de uso representa un requisito funcional, describe secuencias de acciones (con variaciones) que producen resultados observables para un actor, y se modela como una elipse con su nombre [7]. Los **actores** son entidades externas (personas u otros sistemas) que interactúan con el sistema y para los que el caso de uso produce un efecto tangible (p. ej., cálculo de un resultado, generación o cambio de estado de un objeto) [7].

Las **relaciones** habituales incluyen: (i) comunicación *actor–caso de uso*; (ii) *inclusión* (*include*) para factorizar comportamiento común reutilizable; (iii) *extensión* (*extend*) para comportamiento opcional condicionado en puntos de extensión; y (iv) *generalización*, tanto de actores como de casos de uso [7].

La **especificación textual** de un caso de uso suele estructurarse con *precondiciones*, *postcondiciones* (mínimas y de éxito), *escenario principal* (flujo básico) y *escenarios alternativos o de excepción*; opcionalmente se añaden diagramas de actividad o de estados. En cada paso se indica si actúa el actor o el sistema; no se detallan aspectos de interfaz de usuario; y se define claramente cómo se inicia el caso de uso [7].

Como técnica de modelado, se recomienda: identificar y organizar actores; considerar interacciones normales y excepcionales; nombrar los comportamientos como casos de uso; factorizar con *include*/*extend*; y finalmente modelar actores, casos y relaciones en los diagramas [7].

5.1. Diagramas de casos de uso

Tras la definición teórica de los casos de uso, se presentan a continuación los diagramas correspondientes. Estos diagramas ofrecen una representación visual de los actores, los casos de uso y las relaciones descritas previamente, permitiendo obtener de un vistazo la estructura general del sistema y las interacciones fundamentales entre usuarios y aplicación.

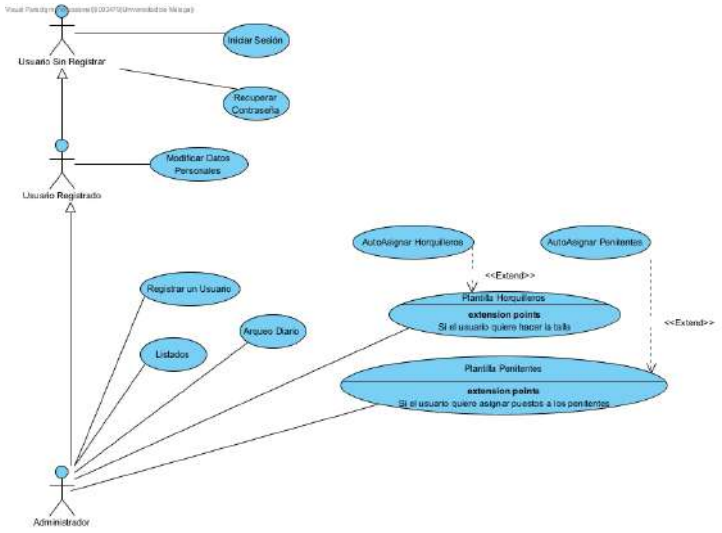


Figura 4: Diagrama de casos de uso - 1

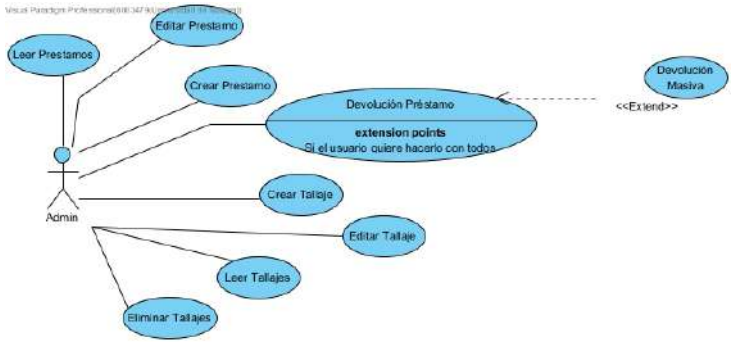


Figura 5: Diagrama de casos de uso - 2

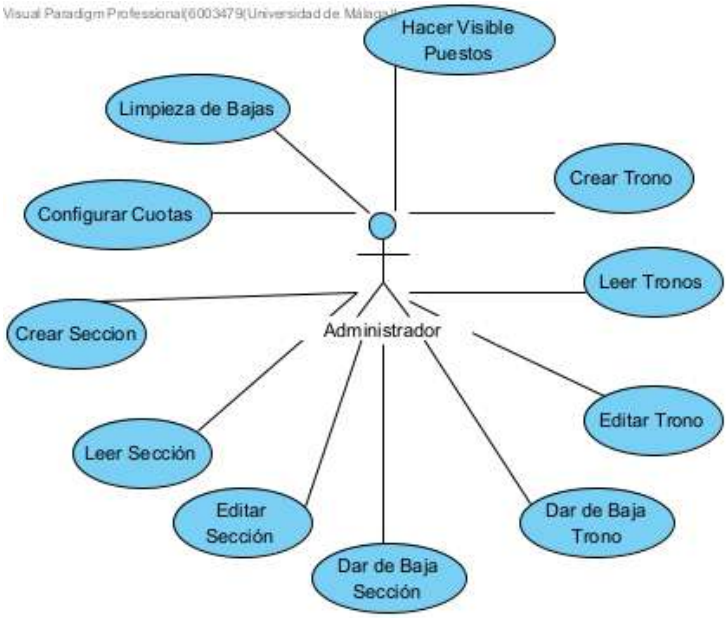


Figura 6: Diagrama de casos de uso - 3

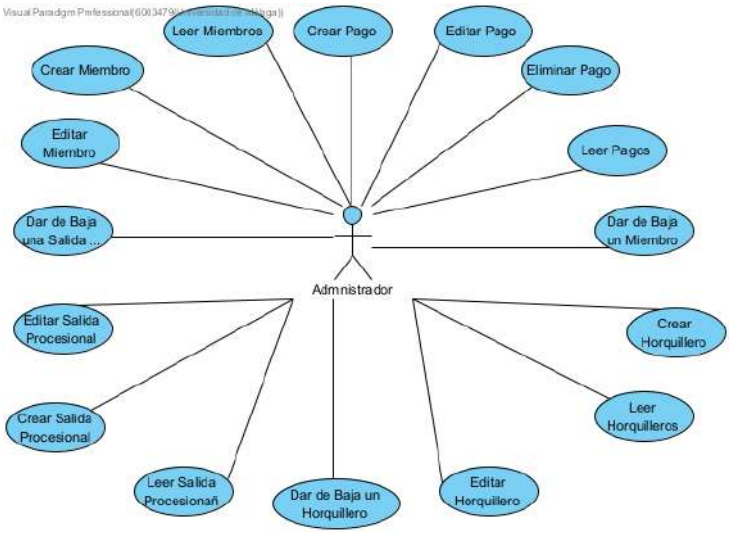


Figura 7: Diagrama de casos de uso - 4

5.2. Especificación de Casos de Uso

En este apartado se especifican de manera detallada dos casos de uso representativos del sistema: **Iniciar Sesión** y **Crear Miembro**. Ambos ilustran los elementos clave (actores, precondiciones, escenarios de éxito y alternativos, y postcondiciones) y reflejan la lógica central de la aplicación. El resto de casos de uso definidos durante el análisis, que abarcan la totalidad de funcionalidades del sistema, se encuentran recopilados en el **Apéndice A.1**.

- **CU01:** Iniciar Sesión

  Actores Usuario no registrado.

  Precondición El usuario no ha iniciado sesión en la aplicación.

  Escenario de Éxito

  1. Pulsar en *Inicio de sesión*.
  2. Introducir correo electrónico y contraseña.
  3. El sistema valida que las credenciales son correctas.
  4. El sistema permite el acceso y redirige al área correspondiente según el rol.

  Escenario Alternativo A1

  3b El sistema detecta credenciales incorrectas.

  4b El sistema muestra un mensaje de error: "Credenciales inválidas".

  5b Pulsar en *Aceptar*.

  6b Volver a introducir credenciales.

  Postcondición El usuario queda autenticado como *Usuario Registrado* o *Administrador* según sus credenciales.

- **CU04:** Crear Miembro

  Actores Administrador.

  Precondición Sesión iniciada como Administrador.

  Escenario de Éxito

1. Acceder a *Miembros*.
2. Pulsar en *Crear*.
3. Introducir nombre, apellidos, DNI, correo, dirección, teléfono y antigüedad (si se deja vacío, se asigna el año actual).
4. (Opcional) Marcar “Es hermano”.
5. El sistema valida obligatorios, formato y unicidad (DNI/correo).
6. Pulsar en *Guardar*.

Escenario Alternativo A1

5b Algún dato es incorrecto o falta.

6b El sistema muestra un mensaje de error indicando el campo.

7b Pulsar en *Aceptar*.

8b Corregir datos.

Escenario Alternativo A2

5c Duplicidad de DNI o correo.

6c El sistema informa y bloquea el alta.

Postcondición El miembro queda creado correctamente en la base de datos, quedando reflejados todos los datos introducidos; si se marcó “Es hermano”, se genera la cuota del año actual.

6

Diseño de la aplicación

En esta sección se presentan los principales artefactos de diseño elaborados durante el desarrollo del proyecto. Se incluyen los **mockups de la interfaz**, que permitieron definir la estética y la experiencia de usuario; los **diagramas de clases**, que muestran la estructura lógica del sistema y sus relaciones; y los **diagramas de secuencia**, que detallan la dinámica de ejecución de algunos de los casos de uso más relevantes. Estos elementos proporcionan una visión clara y estructurada del sistema antes de su implementación, sirviendo como puente entre los requisitos, casos de uso y la fase de desarrollo.

6.1. MockUp de la Aplicación

Para definir la estética inicial de la aplicación y explorar la disposición de los elementos principales de la interfaz, se diseñó un *mockup* con la herramienta **Figma**. Este prototipo no funcional permitió visualizar la primera idea de interfaz y sirvió de guía durante las fases posteriores de diseño y desarrollo.

A continuación se muestran dos pantallas representativas: la **página de inicio (landing)** y la **página principal del usuario registrado (home)**. Estas capturas reflejan la línea estética general planteada para la aplicación.

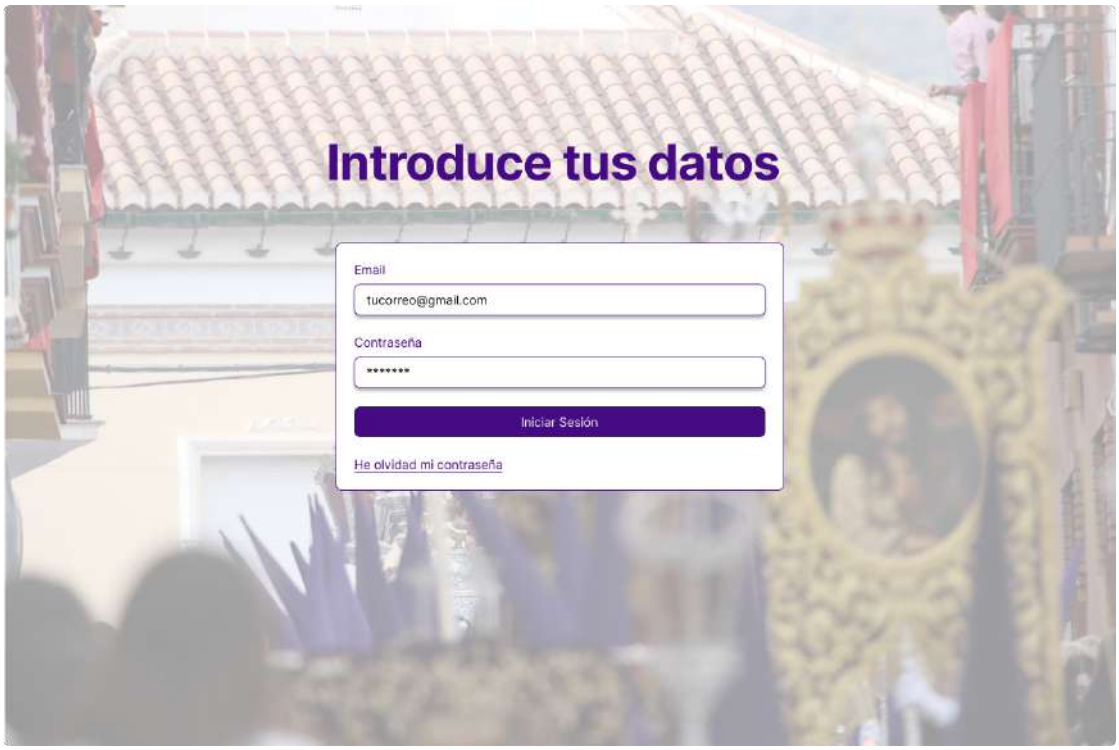


Figura 8: Pantalla de inicio de sesión con datos de usuario

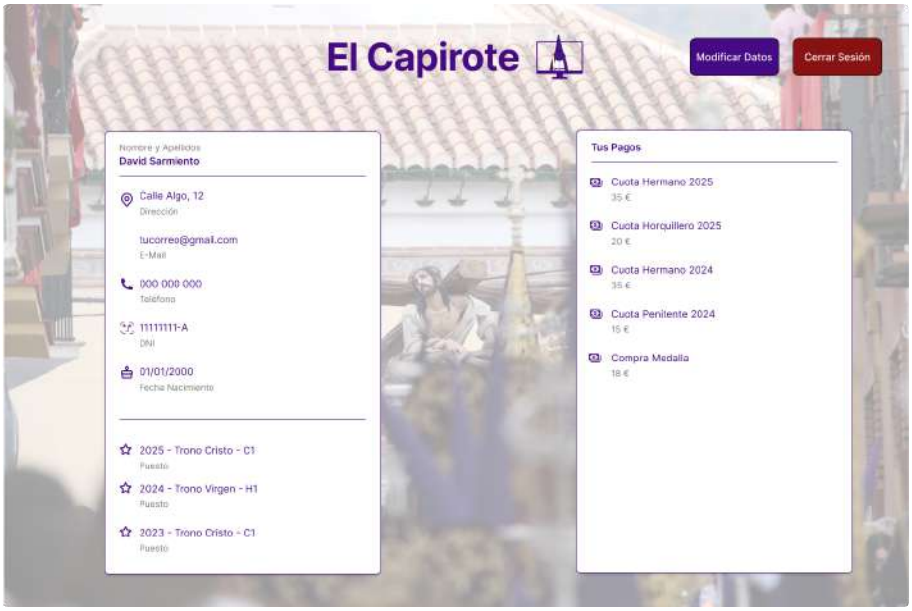


Figura 9: Mockup de la página principal del usuario registrado (*home*)

El resto de pantallas diseñadas en Figma, que incluyen flujos de navegación adicionales y diferentes vistas del sistema, se encuentran recogidas en el **Apéndice A.2**.

6.2. Diagramas de Clases

Los diagramas de clases constituyen uno de los elementos fundamentales del modelado orientado a objetos en UML, ya que permiten representar la estructura estática del sistema. Su objetivo principal es describir los aspectos estáticos del dominio mediante mecanismos de abstracción como la clasificación, la generalización y la agregación, lo que los convierte en un modelo más estable y sencillo de interpretar que los diagramas dinámicos [8].

Este tipo de diagramas recoge las clases del sistema junto con sus asociaciones, atributos y operaciones, mostrando cómo se relacionan entre sí y definiendo la estructura de los objetos que intervienen en la solución. Es importante señalar que los diagramas de clases no reflejan comportamientos temporales, sino que se centran en la organización estructural [8].

En un diagrama de clases, las **clases** representan abstracciones de objetos que comparten atributos, relaciones y operaciones comunes. Cada clase se representa mediante una caja dividida en tres secciones: nombre de la clase (que debe ser un sustantivo o expresión nominal), atributos y operaciones. Los **atributos** son propiedades compartidas por los objetos instanciados de la clase, mientras que las **operaciones** describen los servicios que dichos objetos pueden ofrecer [8].

Por otro lado, las **relaciones** entre clases permiten modelar dependencias estructurales o de comportamiento. Entre ellas destacan:

- *Asociación*: relación estructural que conecta instancias de dos clases, pudiendo incluir roles y multiplicidades.
- *Agregación y composición*: asociaciones todo–parte, donde la composición implica una dependencia más fuerte entre los objetos.
- *Dependencia*: indica que una clase utiliza los servicios de otra.
- *Generalización o herencia*: relación *es-un* entre superclases y subclases, permitiendo la reutilización y extensión de comportamiento.

Finalmente, los diagramas de clases se construyen de forma iterativa. El proceso suele comenzar identificando las clases y asociaciones candidatas a partir de la definición del problema y el conocimiento del dominio, para posteriormente refinar atributos, operaciones y relaciones, introduciendo herencia o agrupándolas en paquetes según sea necesario [8].

El diseño de clases ha seguido un proceso iterativo, refinándose a medida que se avan-

zaba en el desarrollo de la aplicación. A continuación se presentan las distintas iteraciones realizadas y, finalmente, la descripción detallada del diagrama definitivo.

6.2.1. Iteración 1

En la primera versión del diagrama de clases (Figura 10) se propuso un único esquema global que incluía tanto las entidades como la lógica de negocio. El diagrama mostraba las principales clases del sistema: *Usuario*, *Miembro*, *Horquillero*, *Pago*, *Trono*, *Sección*, *Configuración*, *Préstamo*, *Tallaje* y *SalidaProcesional*.

Este primer enfoque permitía tener una visión global del sistema, pero presentaba ciertas limitaciones:

- La inclusión de métodos de servicio dentro de las clases de entidad sobrecargaba el diseño.
- La separación entre datos persistentes y lógica de negocio no estaba claramente definida.
- Algunas relaciones resultaban ambiguas de cara a su implementación posterior.

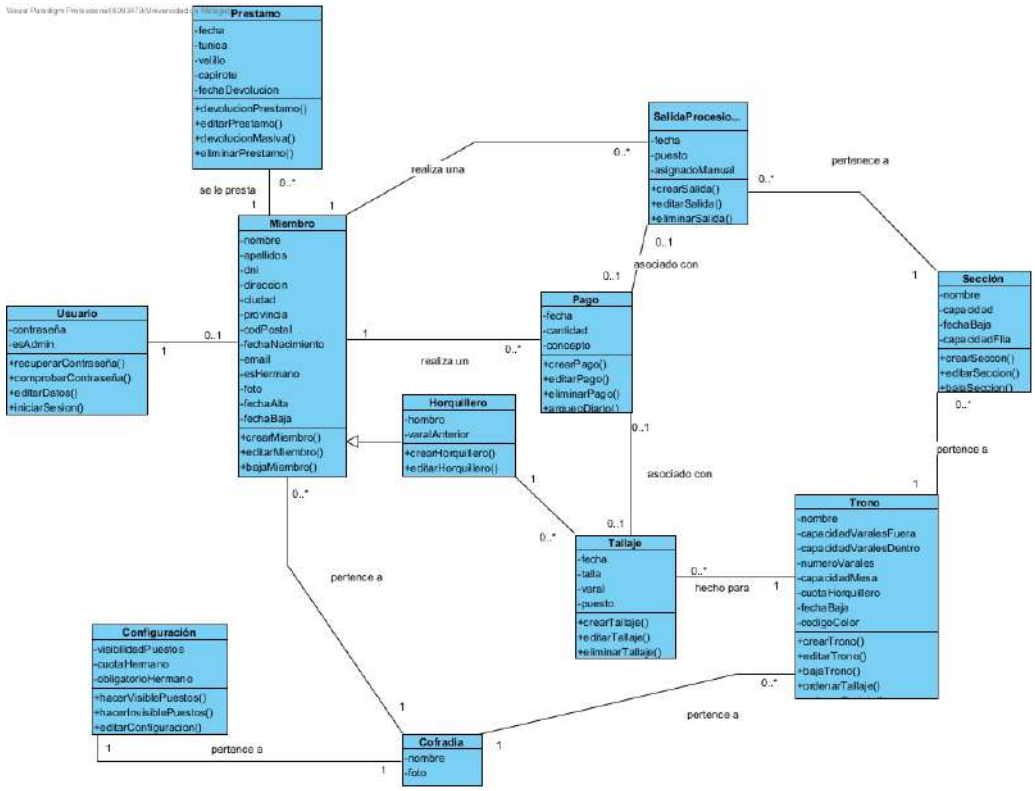


Figura 10: Primera iteración del diagrama de clases (visión global inicial).

6.2.2. Iteración 2

Tras elaborar los diagramas de secuencia, se detectó la necesidad de distinguir entre las entidades persistentes y los servicios encargados de la lógica de negocio. De este modo, en la segunda iteración el modelo se dividió en dos diagramas diferenciados:

- **Diagrama de entidades** (Figura 11): centrado en las clases que representan datos persistentes, como *Miembro*, *Usuario*, *Préstamo*, *Pago*, *Trono*, etc.
- **Diagrama de servicios** (Figura 12): recogía la lógica de negocio en clases como *MiembroService*, *UsuarioService*, *PagoService*, *OrdenarTallajeService*, etc. Cada una de ellas contenía los métodos que permiten gestionar las operaciones CRUD y otros procesos asociados.

Esta separación mejoró notablemente la claridad y mantenibilidad del sistema, permitiendo diferenciar claramente qué parte corresponde al almacenamiento de datos y qué parte a la lógica funcional.

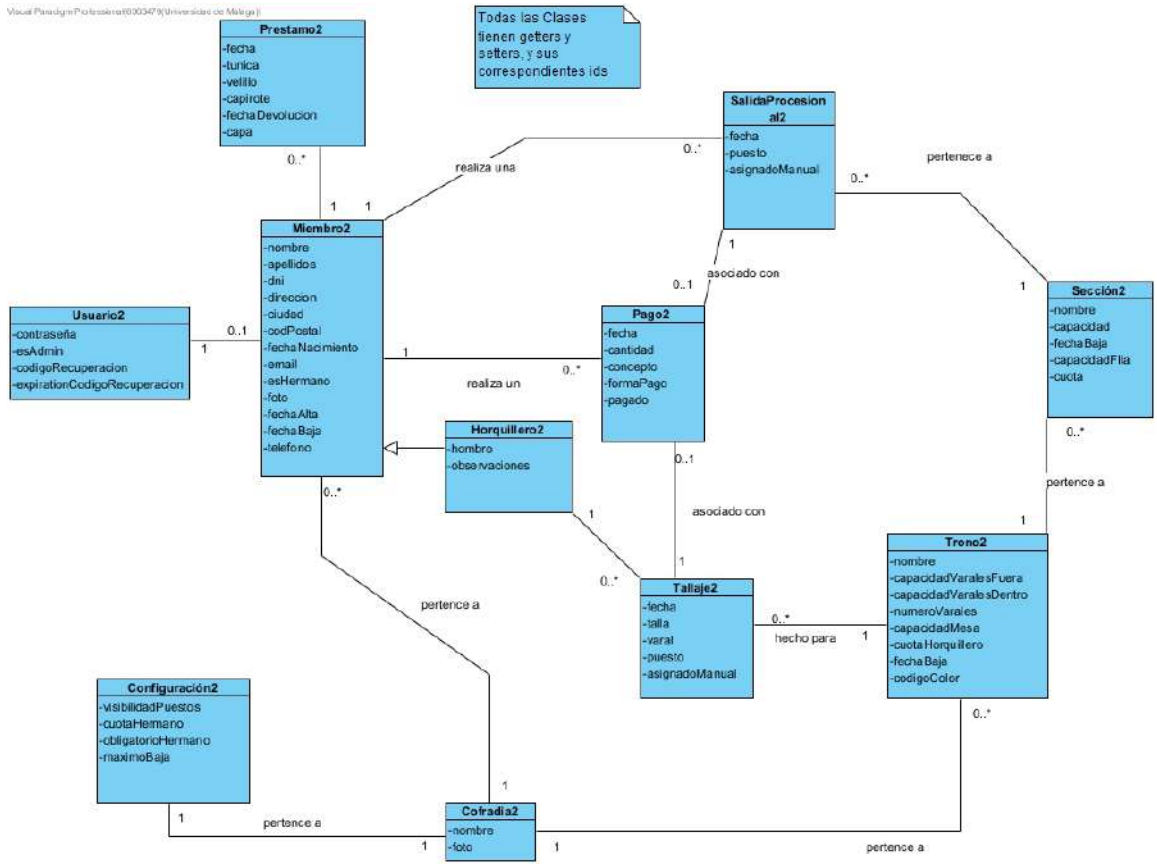


Figura 11: Segunda iteración: diagrama de clases de entidades.

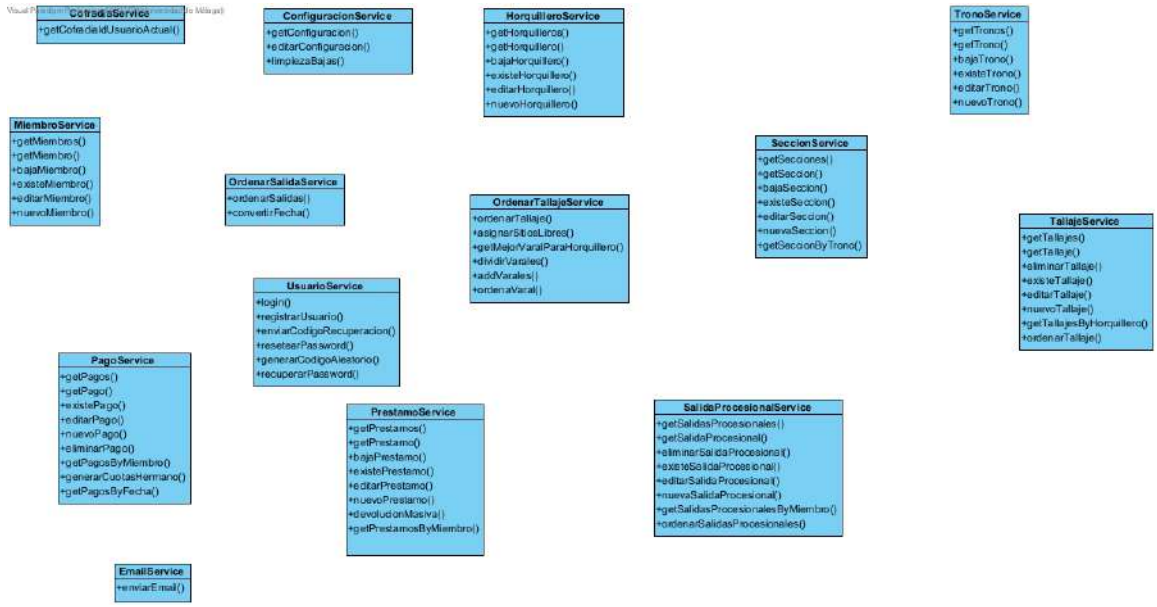


Figura 12: Segunda iteración: diagrama de clases de servicios.

6.2.3. Iteración 3 (Definitiva)

La tercera iteración (Figuras 13 y 14) corresponde al diseño final, obtenido tras el desarrollo real de la aplicación y las adaptaciones derivadas de la implementación. En esta versión se consolidó la separación entre entidades y servicios, y se incorporaron mejoras adicionales como la introducción de la clase **AuditoríaLog**, destinada al registro de operaciones realizadas por los usuarios.

Clases principales del diagrama de entidades:

- **Usuario**: almacena las credenciales y roles de acceso, con atributos para recuperación de contraseñas y control de administrador.
- **Miembro**: representa al núcleo de la aplicación, con información personal y estado dentro de la cofradía.
- **Horquillero**, **Tallaje** y **SalidaProcesional**: definen la participación de los miembros en los tronos y cortejos, incluyendo asignación de puestos y tallajes.
- **Pago** y **Préstamo**: gestionan la tesorería y el inventario de túnicas y enseres.
- **Trono** y **Sección**: estructuran la organización procesional, vinculando miembros y salidas.
- **Configuración**: centraliza parámetros globales como cuotas o visibilidad de puestos.
- **AuditoríaLog**: registra cada operación realizada en el sistema, incluyendo fecha, usuario, entidad afectada y tipo de acción.

Relaciones más relevantes:

- Un *Miembro* puede estar asociado a múltiples *Pagos*, *Préstamos*, *Tallajes* y *SalidasProcesionales*.
- Un *Trono* se organiza en *Secciones*, cada una con su capacidad y lista de penitentes (salidas procesionales). Aparte de las Secciones, cada Trono es llevado por un número de Horquilleros, y esta lista queda reflejada por la tabla Tallaje, que une los Tronos con los propios Horquilleros.

- Las clases de configuración y auditoría mantienen relaciones de asociación que refuerzan la seguridad y trazabilidad del sistema.

Clases de servicios: En el diagrama de servicios se incluyen clases específicas para cada entidad (*UsuarioService*, *MiembroService*, *PagoService*, etc.), responsables de implementar las operaciones CRUD, validaciones y lógica adicional (como el cálculo de tallajes o la auto-asignación de puestos). Destacan:

- **OrdenarTallajeService**: implementa la lógica de asignación automática de horquilleros en los varales.
- **OrdenarSalidaService**: automatiza la distribución de penitentes en el cortejo.
- **AuditoríaService**: registra cada acción en la base de datos para garantizar trazabilidad.

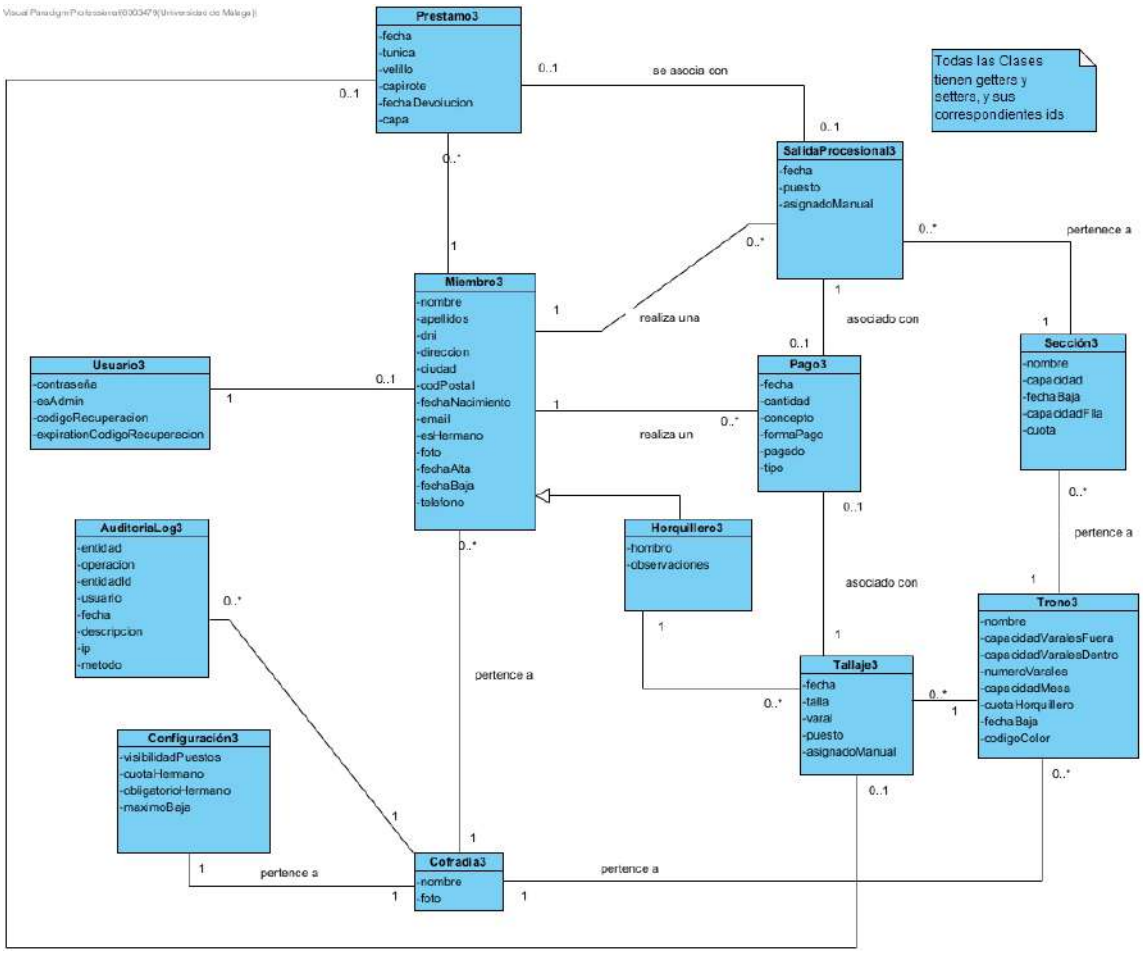


Figura 13: Tercera iteración (definitiva): diagrama de clases de entidades.

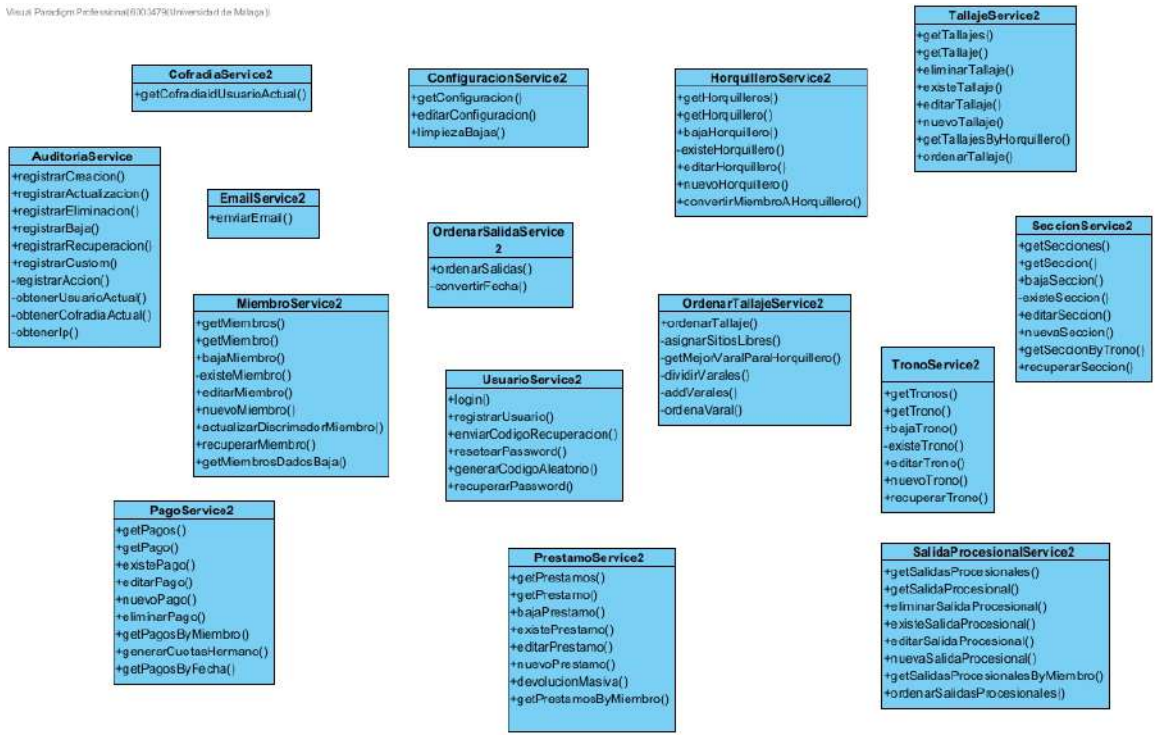


Figura 14: Tercera iteración (definitiva): diagrama de clases de servicios.

Conclusión: La evolución del diagrama de clases refleja el proceso de refinamiento propio de un desarrollo iterativo. Se partió de un diseño único y general, se pasó a una separación clara entre entidades y servicios, y finalmente se consolidó un modelo más completo, con nuevas clases de soporte (*AuditoríaLog*) y una mayor precisión en las relaciones. El resultado es un esquema estructurado, mantenible y alineado con los requisitos funcionales y no funcionales definidos en fases previas.

6.3. Diagramas de Secuencia

Los diagramas de secuencia son un tipo de diagrama de interacción en UML que muestran la comunicación entre un conjunto de objetos, poniendo el énfasis en el orden temporal de los mensajes intercambiados. De esta forma, permiten visualizar de manera clara cómo se desarrolla un escenario particular, complementando a los casos de uso al detallar su dinámica de ejecución [9].

Estos diagramas se caracterizan porque el tiempo progresa de arriba hacia abajo, mientras que los objetos participantes se distribuyen horizontalmente de izquierda a derecha, en función

de su intervención en la interacción. Cada participante se representa con una **línea de vida**, que indica su existencia en el tiempo, y con rectángulos de **activación**, que reflejan cuándo está activo en la secuencia [9].

Los mensajes intercambiados pueden adoptar diferentes formas:

- **Síncronos**: el emisor queda bloqueado hasta recibir respuesta (flechas de cabeza rellena).
- **Asíncronos**: generan un nuevo hilo de ejecución y no esperan respuesta (flechas de cabeza abierta).
- **Auto-mensajes**: cuando un objeto se envía un mensaje a sí mismo.
- **Mensajes encontrados y perdidos**: permiten representar mensajes cuyo origen o destino no está claramente definido.

Asimismo, los diagramas de secuencia pueden incorporar **fragmentos combinados**, que permiten modelar comportamientos más complejos mediante estructuras como alternativas, bucles, condiciones de opción, interrupciones (*break*) o ejecuciones paralelas. No obstante, se recomienda su uso moderado para evitar diagramas excesivamente enmarañados y difíciles de interpretar [9].

En el proceso de desarrollo, los diagramas de secuencia son especialmente útiles en la fase de análisis y diseño, ya que sirven para refinar los casos de uso e identificar los objetos y métodos que intervendrán en la implementación. De esta manera, proporcionan un puente claro entre la especificación de requisitos y la codificación del sistema.

A continuación se muestran y describen dos de los distintos diagramas de secuencia elaborados, los cuales detallan la dinámica de algunos de los casos de uso principales del sistema.

Además de los diagramas aquí presentados, se elaboraron otros diagramas de secuencia que completan la representación de distintos flujos relevantes de la aplicación. Para no sobrecargar esta sección, dichos diagramas se incluyen en el **Apéndice A.3**, donde pueden consultarse en detalle.

6.3.1. Creación de Tallaje

En este caso, el administrador crea un nuevo tallaje para un horquillero. Primero, se verifica que el horquillero y el trono asociados existan en el sistema. Si alguno de ellos no existe, se devuelve un error. En caso afirmativo, se ejecuta el método $\mathrm{crearTallaje}()$, lo que genera un

nuevo registro. El diagrama contempla los caminos alternativos de error mediante fragmentos alt, diferenciando entre horquillero inexistente, trono inexistente y tallaje creado exitosamente.

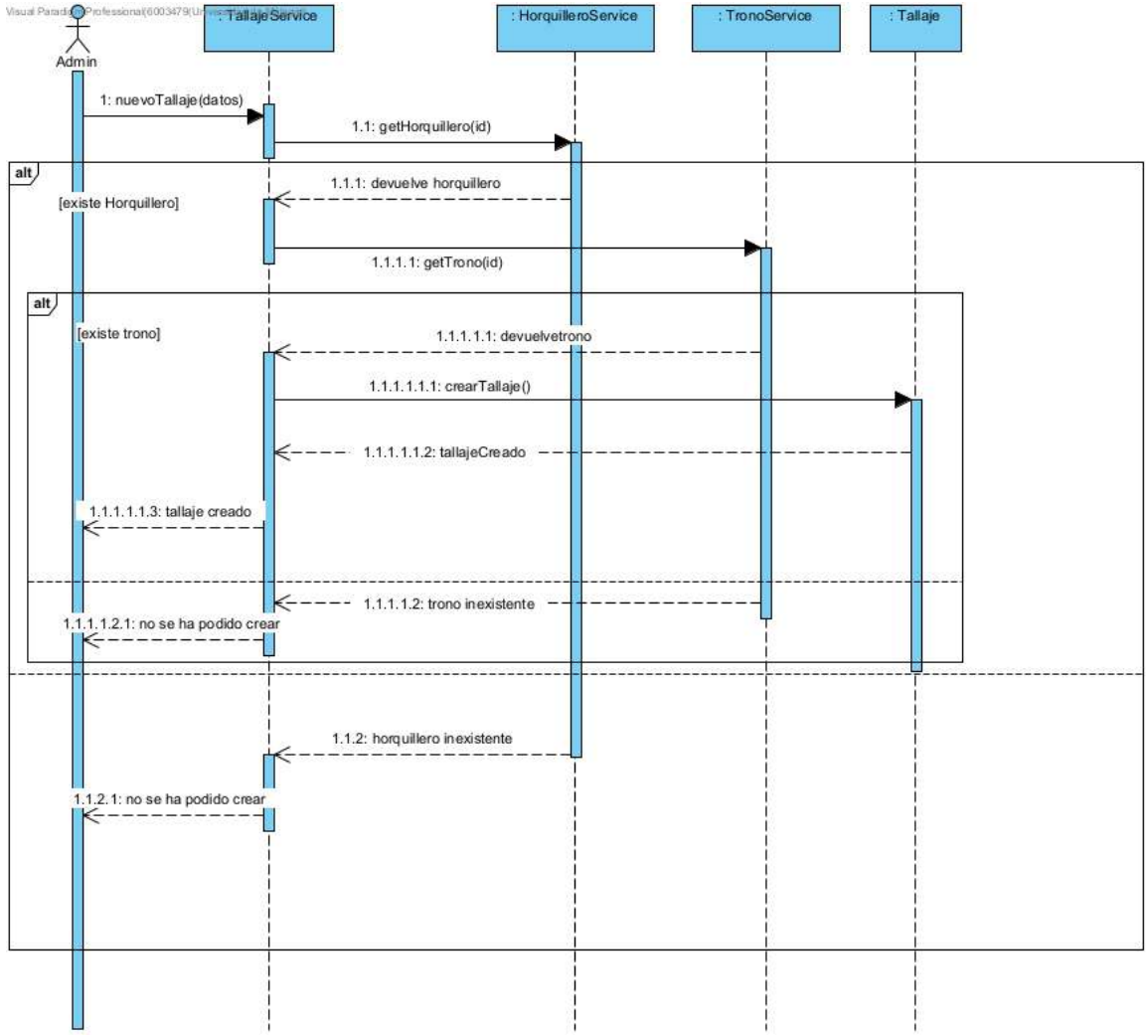


Figura 15: Diagrama de secuencia para la creación de un tallaje.

6.3.2. Ordenación de Tallaje

El diagrama muestra cómo el administrador ordena los tallajes de un trono en una determinada fecha. El servicio TallajeService consulta la existencia del trono y, si este existe, delega la lógica en OrdenarTallajeService. Este último se encarga de dividir tallajes en varales, rellenar los puestos libres y ordenarlos. El uso del fragmento alternativo refleja el caso de trono inexistente, en cuyo caso el proceso se detiene.

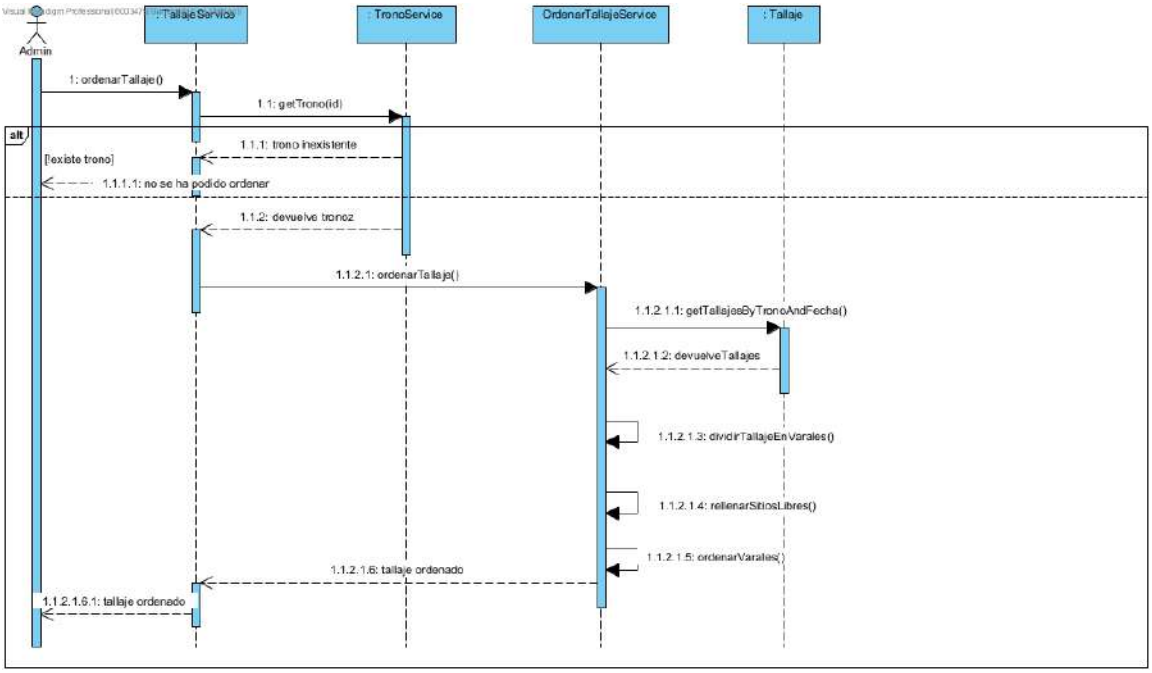


Figura 16: Diagrama de secuencia para la ordenación de tallajes.

7

Desarrollo del Proyecto

Una vez definidos los requisitos, seleccionadas las tecnologías y realizado el diseño mediante los diferentes diagramas UML, se procedió al desarrollo de la aplicación. Esta fase constituye la materialización práctica de las decisiones previas, transformando los modelos conceptuales en una solución funcional y operativa.

El proceso de desarrollo siguió un enfoque iterativo e incremental, lo que permitió implementar y validar las distintas funcionalidades en ciclos cortos, reduciendo riesgos y facilitando la detección temprana de errores. Cada iteración estuvo orientada a cubrir un conjunto de casos de uso previamente priorizados, asegurando así que las funcionalidades críticas estuvieran disponibles en las primeras fases del desarrollo.

La aplicación se estructuró en dos capas principales: un **frontend**, desarrollado con Angular, encargado de la interacción con el usuario y la presentación de la información; y un **backend**, implementado en Spring Boot, responsable de la lógica de negocio y la gestión de datos. Ambos módulos se conectan con una base de datos MySQL, que garantiza la persistencia de la información y el soporte a las operaciones definidas en los requisitos.

Durante esta fase también se aplicaron principios de modularidad y reutilización, siguiendo las directrices establecidas en los diagramas de clases y de secuencia. Además, se emplearon herramientas de control de versiones y entornos de desarrollo integrados que facilitaron la organización del código, la trazabilidad de cambios y la integración progresiva de componentes.

En las siguientes secciones se detallan las principales decisiones de implementación, la estructura del código, las funcionalidades desarrolladas y las iteraciones realizadas hasta obtener la versión final de la aplicación.

7.1. Desarrollo Backend

El backend se ha implementado con *Spring Boot* 3.x siguiendo una arquitectura por capas y paquetes bien diferenciados:

- **controller**: expone la API REST (controladores), maneja *DTOs*, códigos de estado HTTP y documentación con OpenAPI/Swagger.
- **service**: encapsula la lógica de negocio (servicios de dominio) y coordina transacciones.
- **repository**: acceso a datos mediante *Spring Data JPA* (interfaces JpaRepository).
- **model**: entidades JPA que representan el modelo de dominio.
- **Crypto**: Contiene toda la lógica de encriptación y desencriptación de datos personales.
- **security**: autenticación y autorización con *Spring Security* y JWT (filtros de autenticación/validación).
- **aspectos** y **anotaciones**: auditoría transversal con AOP y la anotación @Auditable para registrar operaciones.
- **configs**: configuración general, constantes y metadatos de OpenAPI.
- **dtos**: objetos de transferencia para peticiones y respuestas de la API.

Esta separación favorece la cohesión interna de cada capa y reduce el acoplamiento, permitiendo iterar y probar cada componente de forma aislada.

En este proyecto no se ha elaborado un diagrama Entidad/Relación (E/R) de manera independiente. La razón principal es que, al emplear **Java Spring Boot** junto con **JPA** (*Java Persistence API*), las entidades del modelo de dominio ya se encuentran mapeadas directamente a las tablas de la base de datos. De este modo, el **diagrama de clases** (Figura 13) representa de forma fiel y suficiente la estructura de datos, las relaciones y las cardinalidades que normalmente se expresarían en un E/R. Por ello, se consideró redundante realizar un diagrama adicional, dado que la trazabilidad entre clases y tablas es prácticamente equivalente.

7.1.1. Modelo de datos (entidades JPA)

El modelo implementa la tercera iteración del diseño de clases. Las entidades principales son: Miembro, Usuario, Horquillero, Pago, Prestamo, Tallaje, SalidaProcesional, Seccion, Trono, Configuracion, Cofradia y AuditoriaLog.

A destacar:

- **Herencia en miembros**: Miembro se anota con @Inheritance(strategy = InheritanceType.JOINED) y usa @DiscriminatorColumn, separando datos comunes y específicos.

Listing 1: Clase Miembro

```
@Inheritance(strategy = InheritanceType.JOINED)
// datos comunes en esta tabla, y los específicos en las
   tablas hijas
@DiscriminatorColumn(name = "dtype", discriminatorType =
   DiscriminatorType.STRING)
public class Miembro {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Convert(converter = CryptoStringConverter.class)
    private String nombre;

    @Convert(converter = CryptoStringConverter.class)
    private String apellidos;

    @Convert(converter = CryptoStringConverter.class)
    private String dni;

    @Convert(converter = CryptoStringConverter.class)
```

```
    private String direccion;

    @Convert(converter = CryptoStringConverter.class)
    private String ciudad;

    @Convert(converter = CryptoStringConverter.class)
    private String telefono; // Cambiado a String para
cifrar

    @Convert(converter = CryptoStringConverter.class)
    private String codPostal;

    private Integer antiguedad;

    @Convert(converter = CryptoStringConverter.class)
    private String fechaNacimiento;

    // Temporalmente sin cifrar el email para la
transición
    @Convert(converter = CryptoStringConverter.class)
    private String email;

    // columna hash (determinista) para UNIQUE y búsquedas
    @Column(name = "email_hash", unique = true, length =
64)
    private String emailHash;

    @Builder.Default
    private Boolean esHermano = false;

    private String foto;
```

```
    private LocalDate fechaAlta;
    private LocalDate fechaBaja;

    // Atributos de usuario
    @OneToOne(fetch = FetchType.EAGER, cascade =
   CascadeType.ALL, orphanRemoval = true)
    private Usuario usuario;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(nullable = false)
    private Cofradia cofradia;

}
```

Listing 1: Clase Miembro

- **Relaciones clave**:
  - Miembro → Usuario (1:1) para credenciales y estado.
  - Miembro → Cofradia (N:1).
  - Tallaje, SalidaProcesional y Pago se asocian a Miembro y/o Trono/Seccion según el caso.
- **Auditoría**: AuditoriaLog registra entidad, id, usuario, IP, método y tipo de operación.

7.1.2. Servicios y lógica de negocio

La lógica de negocio reside en **service**. Se dispone de servicios especializados (UsuarioService, PagoService, PrestamoService, TallajeService, SalidaProcesionalService, SeccionService, TronoService, ConfiguracionService, CofradiaService), además de:

- **MiembroService**: al igual que los servicios anteriores, encapsula la lógica de negocio relacionada con su modelo (en este caso, los **miembros**). Algunos ejemplos de esta lógica de negocio (métodos de la clase) son:

Listing 2: Método para obtener los miembros activos

```
/**
 * Obtiene todos los miembros activos (que no tienen fecha
    de baja).
 * @return Lista de miembros activos
 */
public List<Miembro> getMiembros() {
    // Obtiene el ID de la cofradía del usuario actual
    Long cofradiaId = cofradiaService.
   getCofradiaIdUsuarioActual();
    return miembroRepository.findByCofradiaId(cofradiaId).
   stream()
            .filter(miembro -> miembro.getFechaBaja() ==
   null)
            .collect(Collectors.toList());
}
```

Listing 2: Método para obtener los miembros activos

Listing 3: Método para crear un nuevo miembro

```
/**
 * Crea un nuevo miembro en la cofradía.
 * @param nuevo Datos del nuevo miembro
 * @return Miembro creado y guardado
 * @throws CofradiaNoEncontradaException si la cofradía
   especificada no existe
 * @throws MiembroExistenteException si ya existe un
   miembro con el mismo email y DNI
 */
@Auditable(operacion = Auditable.TipoOperacion.CREATE,
   entidad = "MIEMBRO")
public Miembro nuevoMiembro(Miembro nuevo) {
```

```
        if (nuevo.getDni() == "") nuevo.setDni(null);

        // ID de la cofradía del usuario actual
        Long cofradiaId = cofradiaService.
getCofradiaIdUsuarioActual();

        Cofradia cofradia = cofradiaRepository.findById(
cofradiaId)
                .orElseThrow(CofradiaNoEncontradaException::
new);

        if (this.existeMiembro(nuevo)) throw new
MiembroExistenteException();

        nuevo.setCofradia(cofradia);
        nuevo.setFechaBaja(null);

        nuevo = miembroRepository.save(nuevo);

        if (nuevo.getEsHermano()) {
            // Si el nuevo miembro es hermano, se le asigna un
 pago inicial
            pagoService.nuevoPago(Pago.builder()
                    .fecha(LocalDate.now())
                    .concepto(nuevo.getNombre() + " " + nuevo.
getApellidos() + " " + LocalDate.now().getYear())
                    .tipo("Cuota de Hermano")
                    .cantidad(configuracionService.
getConfiguracion().getCuotaHermano().doubleValue())
                    .formaPago("Efectivo")
                    .pagado(false)
```

```
                    .miembro(nuevo)
                    .build());
        }
        return nuevo;
    }
```

Listing 3: Método para crear un nuevo miembro

- **AuditoriaService**: registra automáticamente *CREATE, UPDATE, DELETE, BAJA, RECUPERACION, CONSULTA, CUSTOM* mediante AOP y la anotación @Auditable.

7.1.3. Algoritmos de ordenación

Estos algoritmos forman parte de la lógica de negocio de tallajes y salidas procesionales. Para mejorar escalabilidad y legibilidad, se implementan en servicios paralelos.

Ordenar tallajes

Para ordenar tallajes se siguen una serie de pasos preestablecidos. Para definirlos, se mantuvieron reuniones con distintas cofradías de Vélez-Málaga, afinando el proceso y el algoritmo resultante.

1. Obtener todos los tallajes de un trono específico (buscarTallajes()).
2. Dividir los tallajes en listas por varal (dividirVarales()).
3. Calcular los huecos libres en cada varal (sitiosLibres()).
4. Asignar los huecos libres a tallajes sin varal (varal **N**) (asignarSitiosLibres()).
5. Ordenar cada varal con la gente ya asignada (ordenaVaral()).

Se excluyen horquilleros dados de baja y se separan asignaciones *manuales* de *automáticas* para ordenar únicamente estas últimas. Se respetan las capacidades de los varales, se **prioriza** la preferencia de hombro y, cuando aplica, se mantiene el varal del año anterior.

Listing 4: Método principal de ordenar tallaje

```
public void ordenarTallaje(Long idTrono, TallajeRepository
    tallajeRepository) {

    Optional<Trono> optTrono = tronoRepository.findById(idTrono
    );
    if (optTrono.isEmpty()) return;

    Trono trono = optTrono.get();

    List<Tallaje> tallajes = buscarTallajes(idTrono,
    tallajeRepository); // tallajes con varales
    Map<Character, List<Tallaje>> tallajesMap = dividirVarales(
    tallajes, trono.getNumeroVarales());
    Map<Character, Integer> sitiosLibresMap = sitiosLibres(
    tallajesMap, trono);

    // Asignar los sitios libres a los tallajes nuevos
    asignarSitiosLibres(tallajesMap, sitiosLibresMap);

    // Orden por talla y antigüedad
    ordenaVaral(tallajesMap, trono);
}
```

Listing 4: Método principal de ordenar tallaje

Ordenar salidas

Para ordenar las salidas procesionales de una sección concreta, se siguió un procedimiento similar de entrevistas con cofradías de Vélez-Málaga y se definió el siguiente algoritmo:

Listing 5: Método principal de ordenar salidas

```
public void ordenarSalidas(Long seccionId,
```

```
    SalidaProcesionalRepository salidaRepository) {

        // Obtener las salidas procesionales de la sección para el
    año actual
        List<SalidaProcesional> salidas = buscarSalidas(seccionId,
    salidaRepository);

        // Separar en manuales y automáticas
        Map<Integer, SalidaProcesional> salidasManuales = new
    HashMap<>();
        List<SalidaProcesional> salidasAutomaticas = new ArrayList
    <>();
        separarSalidasManualesYAutomaticas(salidas, salidasManuales
    , salidasAutomaticas);

        // Filtrar miembros dados de baja
        filtrarMiembrosDadosDeBaja(salidasManuales,
    salidasAutomaticas);

        // Ordenar las automáticas por criterios de precedencia
        ordenarSalidasAutomaticas(salidasAutomaticas);

        // Asignar posiciones finales
        asignarPosiciones(salidasManuales, salidasAutomaticas,
    salidaRepository);
}
```

Listing 5: Método principal de ordenar salidas

7.1.4. Capa REST (controladores) y manejo de errores

Los controladores REST (paquete controller) exponen endpoints organizados por recurso. El manejo de errores se centraliza con RestExceptionHandler, devolviendo *payloads* coherentes (ErrorResponseDTO) y códigos HTTP apropiados. La API se documenta con anotaciones OpenAPI y un OpenApiConfig que define metadatos y el servidor local. Como ejemplo, se muestran algunos endpoints de MiembroController:

Listing 6: Endpoint: creación de un nuevo miembro

```
@PostMapping
public ResponseEntity<?> addMiembro(@RequestBody
    NuevoMiembroDTO nuevo, UriComponentsBuilder builder) {
     try{
          Miembro miembro  = miembroService.nuevoMiembro(Mapper.
    toMiembro(nuevo));
          URI uri = builder
               .path("/miembro")
               .path(String.format("/%d", miembro.getId() ))
               .build()
               .toUri();
          return ResponseEntity.created(uri).body(miembro);
     }catch(MiembroExistenteException e) {
          ErrorResponseDTO error = new ErrorResponseDTO("Miembro
    existente",
          "Ya existe un miembro con el email: " + nuevo.getEmail
    ());
          return ResponseEntity.status(HttpStatus.CONFLICT).body(
    error);
     }catch(CofradiaNoEncontradaException e) {
          ErrorResponseDTO error = new ErrorResponseDTO("Cofradía
     no encontrada", "No se encontró la cofradía especificada");
          return ResponseEntity.badRequest().body(error);
```

```
    } catch (PagoExistenteException e) {
        ErrorResponseDTO error = new ErrorResponseDTO("
Conflicto con pago", "Ya existe un pago asociado que impide
esta operación");
        return ResponseEntity.status(HttpStatus.CONFLICT).body(
error);
    } catch (Exception e) {
        ErrorResponseDTO error = new ErrorResponseDTO("Error
interno",
                "Error interno del servidor: " + e.getMessage());
        return ResponseEntity.status(HttpStatus.
INTERNAL_SERVER_ERROR).body(error);
        }
```

Listing 6: Endpoint: creación de un nuevo miembro

7.1.5. Seguridad: Spring Security + JWT

La seguridad se apoya en **Spring Security**:

- SpringSecurityConfiguration: define el SecurityFilterChain, la política *stateless*, CORS y el PasswordEncoder (BCrypt).
- JwtAuthenticationFilter y JwtValidationFilter: gestionan el inicio de sesión y la validación de tokens JWT.
- JpaUserDetailsService: integra usuarios de base de datos con el proveedor de autenticación.

Los flujos de autenticación se exponen bajo /auth: /login, /checkStatus, /registrar/{id}, /recuperar, /reset.

7.1.6. Crypto: Cifrado de datos personales

Con el fin de garantizar la **seguridad y confidencialidad** de los datos personales gestionados en la aplicación, se ha desarrollado un componente específico de Crypto encargado

de realizar el cifrado y descifrado transparente de la información sensible almacenada en la base de datos. Este mecanismo se ha implementado mediante **AES en modo GCM** con **llave de 256 bits**, inicialización aleatoria (*IV*) y etiqueta de autenticación (*tag*) de 128 bits, lo que asegura tanto la confidencialidad como la integridad de los datos.

El núcleo de la lógica reside en el servicio CryptoService, un @Component de Spring que inicializa la clave a partir de una propiedad externa (app.crypto.key) y expone métodos estáticos para cifrar (enc), descifrar (dec) y comprobar si un dato ya está cifrado (isEncrypted). De este modo, se garantiza un uso sencillo y homogéneo en toda la aplicación.

El cifrado se integra en las entidades JPA mediante el convertidor CryptoStringConverter, implementando la interfaz AttributeConverter. Este convertidor intercepta automáticamente las operaciones de lectura y escritura en la base de datos, aplicando CryptoService.enc() antes de persistir y CryptoService.dec() al recuperar. Con ello, la lógica de cifrado queda completamente desacoplada del resto de la aplicación, facilitando su mantenimiento y evolución.

Gracias a este enfoque:

- Se asegura que los datos personales sensibles (nombre, apellidos, DNI, dirección, etc.) permanezcan cifrados en reposo.
- Se mantiene la transparencia en la lógica de negocio, ya que el desarrollador interactúa con los atributos en texto plano.
- Se permite compatibilidad hacia atrás, al devolver sin error datos no cifrados durante migraciones o pruebas.

En conjunto, la inclusión del módulo Crypto refuerza los requisitos de seguridad (RNF2) del sistema, cumpliendo con las buenas prácticas en protección de datos personales y normativa vigente en materia de privacidad.

7.1.7. Tabla de endpoints principales

A continuación se resume la API por controlador (método y ruta). Los parámetros entre llaves indican *path variables*. La API se documenta con **OpenAPI** (@Operation, @ApiResponses) y se explora mediante **Swagger UI**.

Controlador	**Método**	**Ruta**
UsuarioController	GET	/auth/checkStatus
	POST	/auth/login, /auth/registrar/{id}, /auth/recuperar, /auth/reset
Miembro Controller	GET	/miembro, /miembro/bajas, /miembro/{id}, /miembro/email/{email}
	POST	/miembro
	PUT	/miembro/{id}, /miembro/recuperar/{id}
	DELETE	/miembro/{id}
Horquillero Controller	GET	/horquillero, /horquillero/{id}, /horquillero/email/{email}
	POST	/horquillero, /horquillero/convertir/{miembroId}
	PUT	/horquillero/{id}
	DELETE	/horquillero/{id}
PagoController	GET	/pago, /pago/{id}
	POST	/pago, /pago/generarCuotasHermano
	PUT	/pago/{id}
	DELETE	/pago/{id}
PrestamoController	GET	/prestamo, /prestamo/{id}
	POST	/prestamo
	PUT	/prestamo/{id}
	DELETE	/prestamo/{id}, /prestamo/devolucionMasiva
SalidaProcesional Controller	GET	/salidaProcesional, /salidaProcesional/{id}
	POST	/salidaProcesional, /salidaProcesional/ordenar/{idTrono}
	PUT	/salidaProcesional/{id}
	DELETE	/salidaProcesional/{id}
TallajeController	GET	/tallaje, /tallaje/{id}, /tallaje/horquillero/{id}, /tallaje/trono/{id}

Controlador	**Método**	**Ruta**
	POST	/tallaje, /tallaje/ordenar/{idTrono}
	PUT	/tallaje/{id}
	DELETE	/tallaje/{id}
SeccionController	GET	/seccion, /seccion/bajas, /seccion/{id}
	POST	/seccion
	PUT	/seccion/{id}, /seccion/recuperar/{id}
	DELETE	/seccion/{id}
TronoController	GET	/trono, /trono/bajas, /trono/{id}
	POST	/trono
	PUT	/trono/{id}, /trono/recuperar/{id}
	DELETE	/trono/{id}
Configuracion Controller	GET	/configuracion
	PUT	/configuracion
	DELETE	/configuracion/limpiarBajas

Cuadro 3: Resumen de la API REST publicada por el backend

La documentación de la API se puede consultar en **Swagger UI** (generada automáticamente a partir de anotaciones **OpenAPI**), tal y como se muestra en la Figura 17.

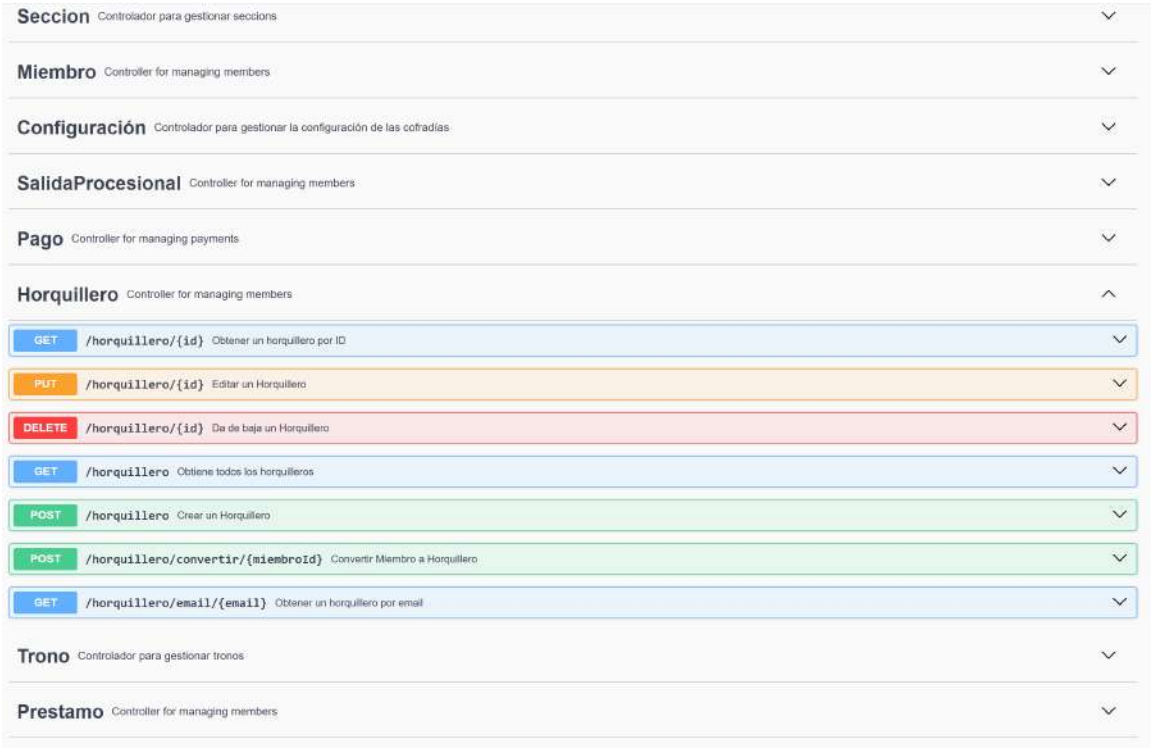


Figura 17: Captura de la documentación *Swagger UI*.

7.1.8. Pruebas

El proceso de pruebas ha sido fundamental para garantizar la calidad y correcto funcionamiento de la aplicación desarrollada. Se han aplicado diferentes enfoques en función de la capa de la aplicación (backend y frontend), adaptando las herramientas y metodologías más adecuadas en cada caso.

En el backend se han implementado **pruebas unitarias** utilizando el framework *JUnit*. Estas pruebas han estado centradas en la funcionalidad de gestión de miembros, validando casos clave como la creación de nuevos registros, la detección de duplicados, la baja de miembros y la correcta asociación con otros elementos del dominio (pagos, tronos, salidas procesionales). Los resultados obtenidos han sido positivos, confirmando que la lógica de negocio y las operaciones básicas del modelo funcionan según lo esperado. En la Figura 18 se muestra un ejemplo de la ejecución satisfactoria de las pruebas realizadas.

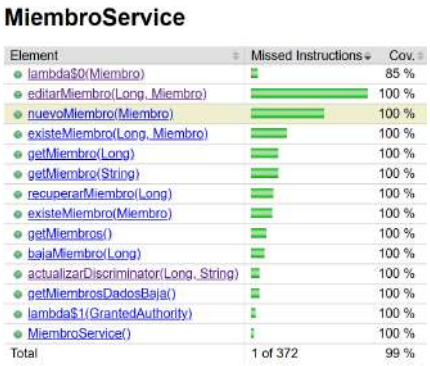


Figura 18: Resultados de las pruebas unitarias en el backend (JUnit).

Dado que la aplicación desarrollada es muy extensa y cuenta con un elevado número de funcionalidades, realizar pruebas unitarias exhaustivas de todos los módulos supondría un esfuerzo temporal prácticamente equivalente al de repetir todo el proceso completo de ingeniería de software llevado a cabo en el proyecto. Por este motivo, se ha optado por centrar las pruebas unitarias en la funcionalidad de **miembros**, que resulta representativa al abarcar operaciones de creación, validación, actualización y baja. De este modo, se demuestra que el proceso de *ingeniería del software* se ha seguido y que se posee la capacidad de diseñar e implementar correctamente pruebas unitarias, siendo extrapolable al resto de funcionalidades de la aplicación.

7.2. Desarrollo FrontEnd

El desarrollo del frontend se ha realizado utilizando **Angular 19**, el *framework* de Google basado en TypeScript más extendido para la construcción de aplicaciones web de una sola página (SPA). Esta elección responde tanto a la experiencia previa adquirida en asignaturas y prácticas curriculares, como a la robustez, modularidad y soporte a largo plazo que ofrece Angular en sus últimas versiones. El objetivo principal del frontend es proporcionar una interfaz de usuario clara, intuitiva y bien estructurada, que permita a los distintos roles de la aplicación acceder de manera eficiente a las funcionalidades definidas en los requisitos.

El frontend se ha organizado por áreas funcionales independientes, todas en `src/app/`: **admin**, **home**, **landing** y **auth**, junto con recursos transversales en `shared` (componentes, servicios, interceptores, utilidades). La navegación se declara en `app.routes.ts` con **guards**

basados en funciones (CanMatch/CanActivate) para proteger rutas.

Listing 7: Archivo que define las rutas principales de la aplicación

```
export const routes: Routes = [
  // Landing page route
  {
    path: 'landing',
    component: LandingPageComponent,
    canMatch: [NotAuthenticatedGuard],
  },
  {
    path: 'home',
    loadComponent: () => import('./home/pages/home-page/home-
   page.component'),
    canMatch: [AuthenticatedGuard, NotAdminGuard],
  },

  {
    path: 'home/editar',
    loadComponent: () => import('./home/pages/edit-page/edit-
   page.component'),
  },

  // auth routes
  {
    path: 'admin',
    canMatch: [IsAdminGuard],
    loadChildren: () => import('./admin/admin.routes'),
  },

  {
```

```
    path: 'auth',
    canMatch: [NotAuthenticatedGuard],
    loadChildren: () => import('./auth/auth.routes'),
  },

  {
    path: '**',
    redirectTo: 'landing',
  },
];
```

Listing 7: Archivo que define las rutas principales de la aplicación

7.2.1. **Área** admin

Ubicación: src/app/admin/

Propósito. Panel de administración con páginas y componentes por funcionalidad: *miembros*, *horquilleros*, *tallajes*, *salidas*, *pagos*, *préstamos*, *secciones*, *tronos*, *configuración* y *plantillas*. Incluye un **layout**, para que todas las páginas de este panel de administración tengan la misma apariencia.

Listing 8: Layout de la Aplicación

```
<style>
  .background-container {
    min-height: 100vh;
    width: 100%;
    background-image: url("/assets/fondo.JPG");
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    background-attachment: fixed;
```

```
    position: relative;
  }
</style>


<div class="background-container">
  <router-outlet></router-outlet>
  <app-notification-container></app-notification-container>
</div>
```

Listing 8: Layout de la Aplicación

De esta forma definimos el estilo de la aplicación aplicándole una foto de fondo en toda ella, y seguimos mostrando el contenido de la ruta con el Router-Outlet

En cuanto al área de administración, podremos encontrar las siguientes rutas principales, que dará paso a otras subrutas definidas por cada hijo (loadChildren):

Listing 9: Rutas principales dentro de Admin

```
export const adminRoutes: Routes = [
  {
    path: '',
    component: AdminLayoutComponent,
    children: [
      //Ruta para el login
      {
        path: 'home',
        component: AdminHomePageComponent,
      },

      {
        path: 'miembros',
        loadChildren: () => import('./miembros/miembros.routes'
  )
```

```
    },
    {
      path: 'horquilleros',
      loadChildren: () => import('./horquilleros/horquilleros
.routes')
    },
    {
      path: 'pagos',
      loadChildren: () => import('./pagos/pagos.routes')
    },
    {
      path: 'prestamos',
      loadChildren: () => import('./prestamos/prestamos.
routes')
    },
    {
      path: 'listados',
      loadChildren: () => import('./listados/listados.routes'
)
    },
    {
      path: 'configuracion',
      loadChildren: () => import('./configuracion/
configuracion.routes')
    },
    {
      path: 'plantillas',
      loadChildren: () => import('./plantillas/plantillas.
routes')
    },
    {
```

```
        path: '**',
        redirectTo: 'home',
      }
    ],
  },
];
```

Listing 9: Rutas principales dentro de Admin

Para cada sección de la aplicación, dada por las rutas definidas anteriormente, tenemos siempre la misma estructura: Servicios, Interfaces, Componentes, Páginas y Rutas .

- **Rutas**: Dentro de cada sección, tenemos una serie de rutas que corresponden con las funcionalidades que queremos que tenga la aplicación (requisitos), y vienen definidas en las posibles rutas.

Listing 10: Rutas dentro de Miembros

```
export const miembrosRoutes: Routes = [
  {
    path: 'crear',
    loadComponent: () => import('./pages/editar-miembro-
    page/editar-miembro-page.component')
  },
  {
    path: 'editar/:id',
    loadComponent: () => import('./pages/editar-miembro-
    page/editar-miembro-page.component')
  },
  {
    path: 'salida/:idMiembro',
    loadComponent: () => import('./pages/nueva-salida-page
    /nueva-salida-page.component')
  },
```

```
{
  path: 'salida/:idMiembro/:idSalida',
  loadComponent: () => import('./pages/nueva-salida-page
/nueva-salida-page.component')
},
{
  path:'pago/:idMiembro',
  loadComponent: () => import('../shared/pages/nuevo-
pago-page/nuevo-pago-page.component')
},
{
  path:'pago/:idMiembro/:idPago',
  loadComponent: () => import('../shared/pages/nuevo-
pago-page/nuevo-pago-page.component')
},
{
  path:'tallaje/:idHorquillero',
  loadComponent: () => import('../horquilleros/pages/
nuevo-tallaje-page/nuevo-tallaje-page.component')
},
{
  path:'tallaje/:idHorquillero/:idTallaje',
  loadComponent: () => import('../horquilleros/pages/
nuevo-tallaje-page/nuevo-tallaje-page.component')
},
{
  path: ':id',
  component: MiembrosPageComponent
},
{
  path: '',
```

```
    component: MiembrosPageComponent
  },
  {
    path: '**',
    redirectTo: ''
  }
];
```

Listing 10: Rutas dentro de Miembros

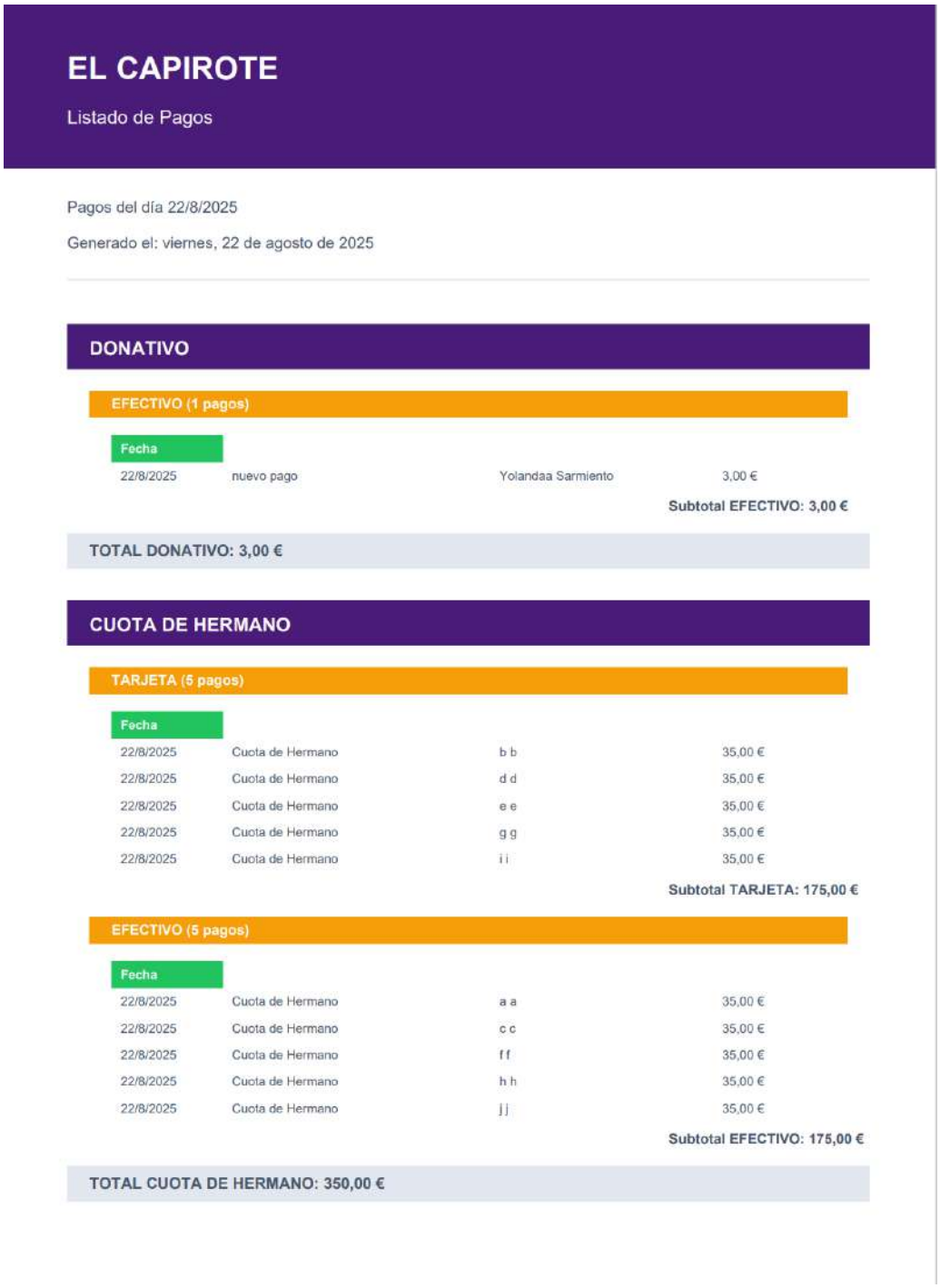
- **Servicios**: Los servicios sirven generalmente para realizar peticiones http a la api del backend, pudiendo tener caches para ahorrar peticiones, y otras lógicas.

Listing 11: Método que gestiona la petición http para crear un Miembro

```
public createMiembro(miembro: Partial<Miembro>):
    Observable<Miembro> {
     return this.http.post<Miembro>(`${URL}/miembro`,
    miembro).pipe(
        map((newMiembro) => MiembroMapper.dates(newMiembro))
    ,
        tap((newMiembro) => {
          // Agregar al caché
          this.cacheMiembros.set(newMiembro.id, newMiembro);
        }),
        catchError((error) => {
          console.error('Error al crear miembro:', error);
          throw error;
        })
      );
  }
```

Listing 11: Método que gestiona la petición http para crear un Miembro

Pero también puede haber otro tipo de servicios, como por ejemplo para generar exportaciones a diferentes estilos de archivos. Esto lo podemos encontrar entre otros en la sección de Pagos, que tenemos un servicio que nos imprime en **PDF** el resumen de pagos dentro de la fecha filtrada.



EL CAPIROTE

Listado de Pagos

Pagos del día 22/8/2025

Generado el: viernes, 22 de agosto de 2025

DONATIVO

EFECTIVO (1 pagos)

Fecha			
22/8/2025	nuevo pago	Yolandaa Sarmiento	3,00 €

Subtotal EFECTIVO: 3,00 €

TOTAL DONATIVO: 3,00 €

CUOTA DE HERMANO

TARJETA (5 pagos)

Fecha			
22/8/2025	Cuota de Hermano	b b	35,00 €
22/8/2025	Cuota de Hermano	d d	35,00 €
22/8/2025	Cuota de Hermano	e e	35,00 €
22/8/2025	Cuota de Hermano	g g	35,00 €
22/8/2025	Cuota de Hermano	i i	35,00 €

Subtotal TARJETA: 175,00 €

EFECTIVO (5 pagos)

Fecha			
22/8/2025	Cuota de Hermano	a a	35,00 €
22/8/2025	Cuota de Hermano	c c	35,00 €
22/8/2025	Cuota de Hermano	f f	35,00 €
22/8/2025	Cuota de Hermano	h h	35,00 €
22/8/2025	Cuota de Hermano	j j	35,00 €

Subtotal EFECTIVO: 175,00 €

TOTAL CUOTA DE HERMANO: 350,00 €

Figura 19: Documento generado por el servicio con el resumen de pagos - 1.

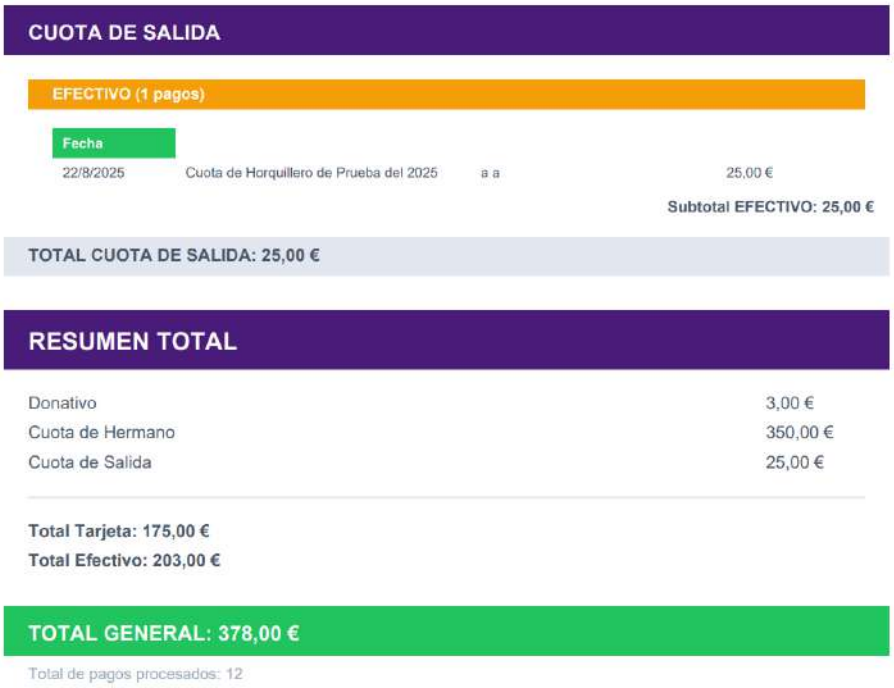


Figura 20: Documento generado por el servicio con el resumen de pagos - 2.

- **Páginas y Componentes**: Cada página y componente tiene su template html, y su componente de TypeScript, que compone toda la lógica como podemos ver en la siguiente imagen:

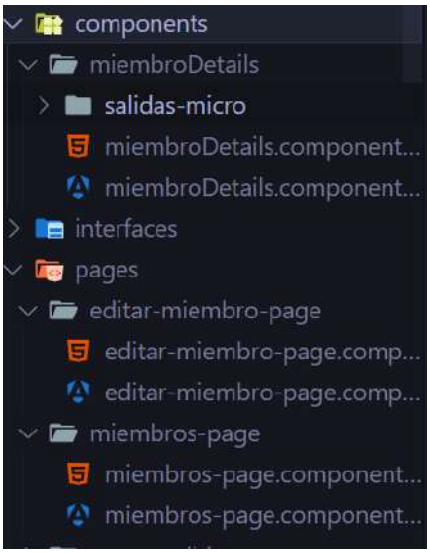


Figura 21: Captura de la estructura de archivos de componentes y páginas.

La diferencia principal es que los componentes se usan dentro de las páginas, y las páginas son lo que se expone a el usuario a través de las rutas.

- **Interfaces**: Las interfaces ayudan al tipado de datos, y correcto funcionamiento de la aplicación, y corresponden a los modelos del backend.

Listing 12: Interfaz de Miembro

```
export interface Miembro {
  id:                number;
  nombre:            string;
  apellidos:         string;
  dni:               string;
  direccion:         string;
  ciudad:            string;
  telefono:          number;
  codPostal:         number;
  antiguedad:        number;
  fechaNacimiento:   string;
  email:             string;
  esHermano:         boolean;
  foto:              string;
  fechaAlta:         Date;
  fechaBaja:         Date |null;
  usuario:           Usuario | null;
  cofradia:          Cofradia;
  hombro?:           string;
  observaciones?:    string;
}
export interface Cofradia {
  id:      number;
  nombre:  string;
  foto:    string;
```

```
}
export interface Usuario {
  id:       number;
  esAdmin:  boolean;
}
```

Listing 12: Interfaz de Miembro

7.2.2. **Área** home

Ubicación: src/app/home/

Propósito: Ofrece al usuario registrado su funcionalidad, por la cual pueden ver toda la información suya que hay en la aplicación, sus salidas procesionales y/o tallajes, y todos sus pagos. Ademas estos usuarios podrán editar sus propios datos personales.

7.2.3. **Área** landing

Ubicación: src/app/landing/

Propósito: Ofrece una pagina de llegada a la aplicación vistosa, por la cual los usuarios no registrados podrán llegar a la funcionalidad de iniciar sesión

7.2.4. **Área** auth **(Autenticación)**

Ubicación: src/app/auth/

Propósito: Implementar la seguridad en la aplicación, mediante las páginas de inicio de sesión y recuperación de contraseña, y el uso de servicios para las validaciones de estos datos y de los token JWT

Este área tiene un tipo de componente no explicado hasta ahora, los **Guardas**. Estos componentes comprueban si un usuario cumple unas condiciones, y lo dejan seguir (return true), y si no se cumplen las condiciones, se realiza una lógica concreta.

Listing 13: Guarda de Usuario Autenticado

```
export const AuthenticatedGuard: CanMatchFn = async (
  route: Route,
```

```
  segments: UrlSegment[]
) => {
  const authService = inject(AuthService);

  const router = inject(Router);

  const isAuthenticated = await firstValueFrom(authService.
   checkStatus());

  if (!isAuthenticated) {
    router.navigateByUrl('/');
    return false;
  }

  return true;
};
```

Listing 13: Guarda de Usuario Autenticado

Otro componente no explicado hasta ahora son los interceptores, que captan cualquier petición realizada por la aplicación, y realizan una lógica concreta.

Listing 14: Interceptor para anexar el token JWT

```
export function authInterceptor(
  req: HttpRequest<unknown>,
  next: HttpHandlerFn
) {
  const token = inject(AuthService).token();
  const newReq = req.clone({
    headers: req.headers.append('Authorization', `Bearer ${
   token}`),
  });
```

```
  return next(newReq);
}
```

Listing 14: Interceptor para anexar el token JWT

7.2.5. Comparativa entre Mockup y Producto Final

El desarrollo del frontend partió de un *mockup* realizado en Figma, pero a lo largo del proceso se fueron introduciendo mejoras en varios apartados. Algunas pantallas evolucionaron hacia un diseño más moderno y funcional, mientras que otras se mantuvieron prácticamente idénticas al diseño inicial, lo que confirma la solidez de la propuesta original.

Panel de administración

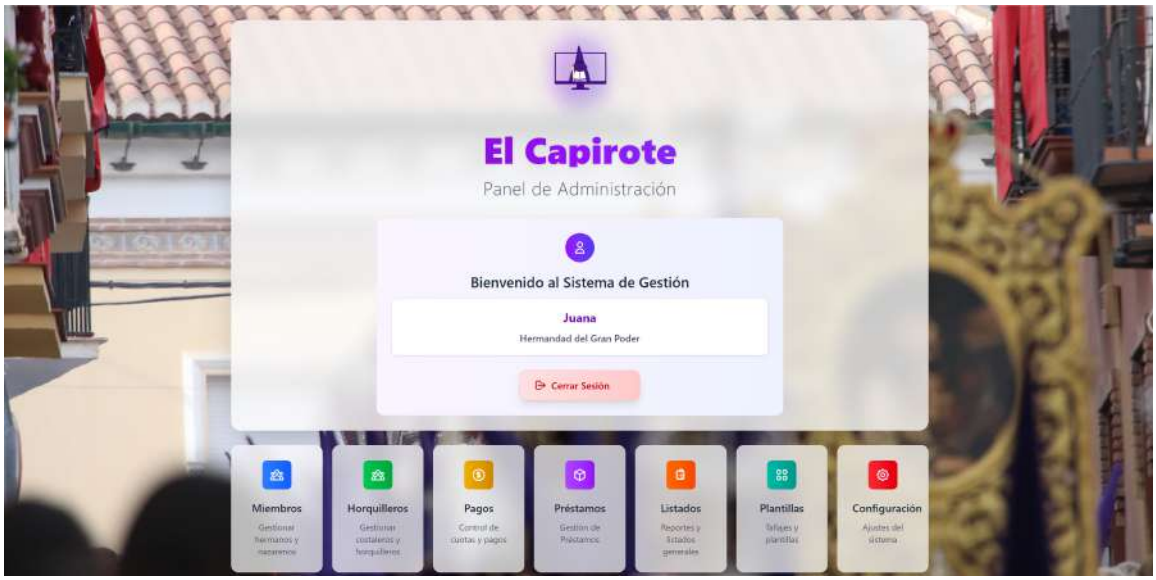


Figura 22: Panel de administración en el producto final

Figura 23: Panel de administración en el mockup inicial

En este caso, el producto final (Figura 22) presenta una interfaz más clara y moderna respecto al *mockup* (Figura 23). Se mejoró la disposición de las tarjetas y se aumentó el contraste de los iconos, lo que facilita la navegación.

Gestión de pagos

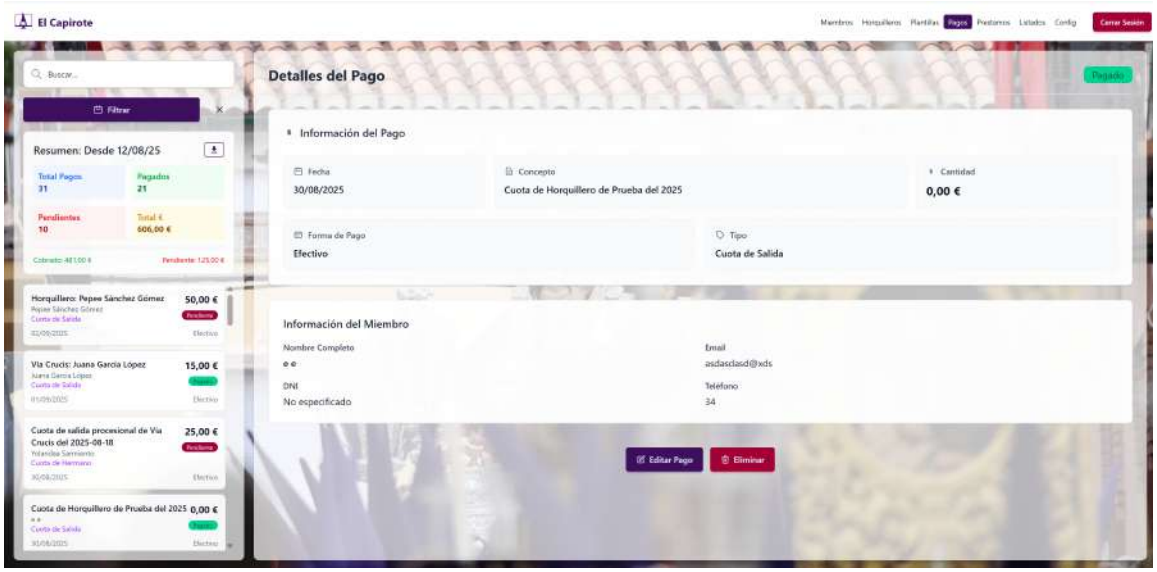


Figura 24: Pantalla de gestión de pagos en el producto final

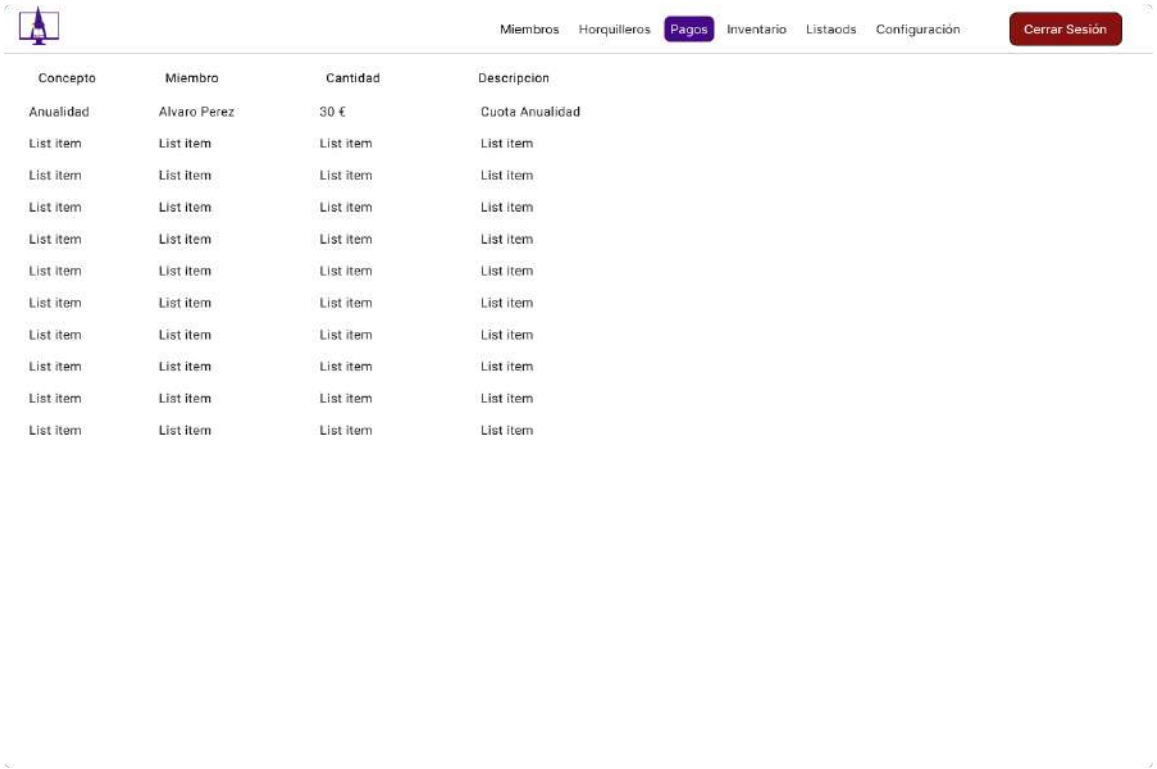


Figura 25: Pantalla de gestión de pagos en el mockup inicial

La sección de pagos también se refinó notablemente. El producto final (Figura 24) añade tarjetas con información detallada, uso de colores para estados (pagado/pendiente) y una disposición más intuitiva, frente a la versión inicial del mockup Figura 25 que mostraba un diseño

más básico.

Landing page

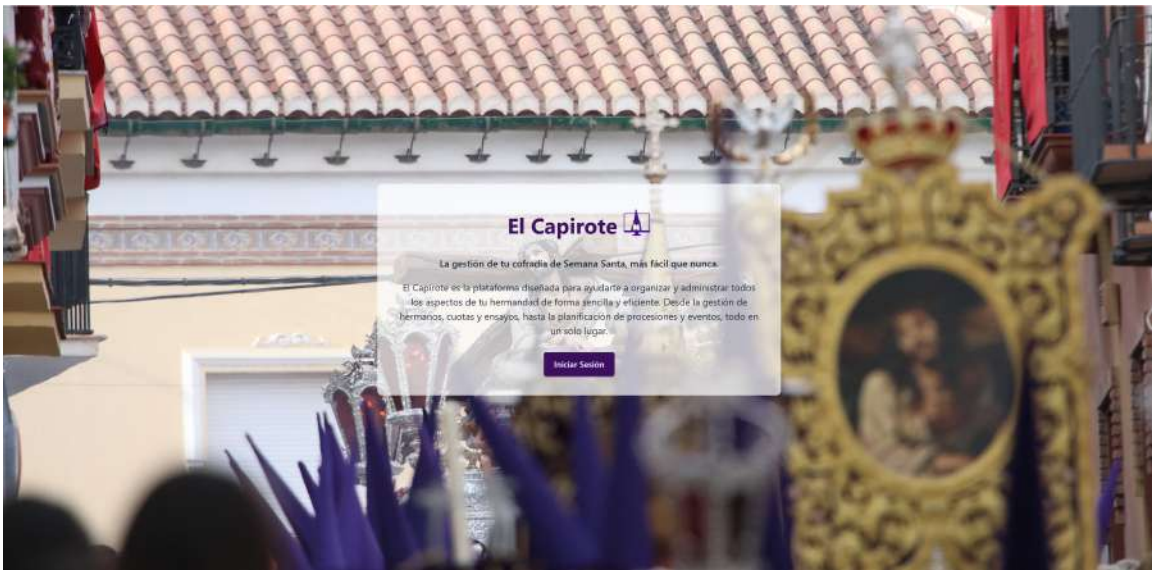


Figura 26: Landing page en el producto final

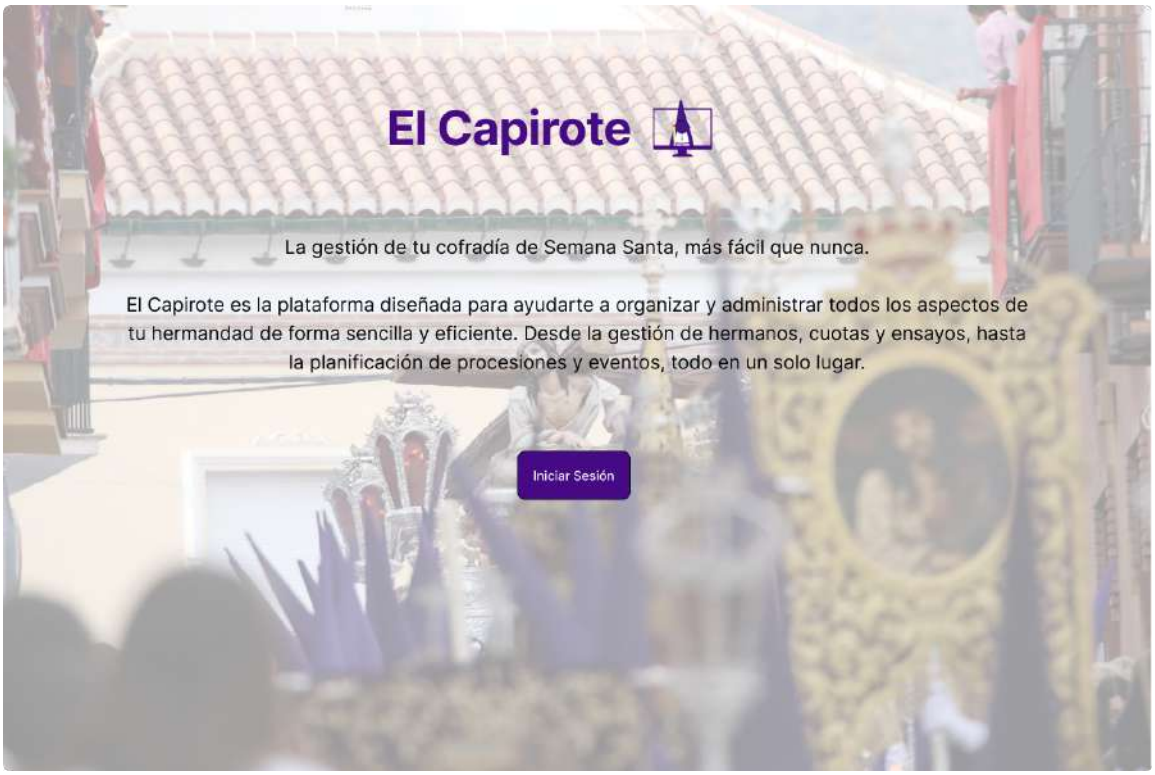


Figura 27: Landing page en el mockup inicial

En este caso, tanto el mockup como el producto final presentan prácticamente la misma

estética. Esto se debe a que el diseño original ya era adecuado: un fondo representativo, un bloque central con información clara y un botón de inicio de sesión destacado. Por ello, se decidió mantener la propuesta inicial sin cambios significativos.

En conclusión, el proceso de diseño del frontend ha seguido un enfoque selectivo: se mantuvieron aquellas pantallas del mockup que ya cumplían los objetivos de usabilidad y estética, y se mejoraron otras para ofrecer un producto final más intuitivo y robusto.

7.2.6. Resumen

La separación por áreas (**admin**, **home**, **landing**, **auth**) permite:

- **Mantenibilidad**: cada carpeta agrupa páginas, componentes, servicios e interfaces de su propia responsabilidad.
- **Escalabilidad**: añadir una nueva funcionalidad en `admin` implica crear una nueva subsección aislada.
- **Rendimiento**: carga diferida por áreas y `standalone components` reducen el *bundle* inicial.
- **Seguridad**: guards por estado/rol y un interceptor centralizado garantizan acceso controlado y sesiones robustas.

Además, para una mayor comprensión de todo lo explicado anteriormente, en el siguiente cuadro se podrá ver un resumen con todas las rutas de la aplicación.

Área	**Ruta**	**Descripción**
Landing	/landing	Página pública inicial con acceso a autenticación.
Auth	/auth/login /auth/lost-password /auth/reset-password	Formulario de inicio de sesión. Página para solicitar recuperación de contraseña. Formulario para restablecer contraseña con token.
Home	/home /home/edit	Página principal del usuario autenticado . Edición de datos personales del usuario.
Admin	/admin/home	Vista principal del panel de administración.

Área	Ruta	Descripción
	/admin/miembros	Gestión de miembros (listado, detalle, edición) y tabmién de las salidas procesionales asociadas a los mismos.
	/admin/horquilleros	Gestión de horquilleros y tallajes asociados.
	/admin/pagos	Gestión de pagos y cuotas de hermanos.
	/admin/prestamos	Gestión de préstamos de túnicas y enseres.
	/admin/plantillas	Gestión de plantillas de cortejo procesional.
	/admin/configuracion	Configuración general de la cofradía, control de recursos dados de baja, y gestión de tronos y secciones.

Cuadro 4: Resumen de rutas principales del frontend Angular

7.2.7. Pruebas Realizadas

En el frontend se optó por la realización de **pruebas manuales**, ejecutadas desde distintos dispositivos (ordenadores de escritorio, portátiles y móviles) con el fin de comprobar la correcta adaptación de la interfaz y la consistencia en la experiencia de usuario. Estas pruebas han permitido validar la navegación entre pantallas, la interacción con formularios y componentes, así como la comunicación con el backend mediante la API REST. El resultado fue satisfactorio, confirmando la estabilidad de la aplicación y el cumplimiento de los requisitos funcionales en todos los escenarios probados.

8

Conclusiones y Líneas Futuras

Antes de finalizar el presente trabajo, resulta pertinente recoger una síntesis de los resultados alcanzados y reflexionar sobre el proceso seguido. En las siguientes secciones se presentan, por un lado, las **conclusiones** más relevantes extraídas del desarrollo del proyecto y, por otro, las **líneas futuras** que permitirían ampliar y mejorar el sistema en versiones posteriores.

8.1. Conclusiones

El proyecto desarrollado ha logrado satisfacer los requisitos iniciales planteados, resolviendo de manera efectiva la problemática detectada en la gestión de las cofradías. La aplicación implementa las funcionalidades principales descritas en el análisis, y ha permitido afinar la implementación conforme avanzaba el desarrollo, garantizando un sistema estable, usable y coherente con las necesidades reales de los usuarios.

Aunque no se han podido cubrir todas las pruebas unitarias debido a la magnitud del sistema, la validación de la funcionalidad de miembros ha demostrado que el proceso de ingeniería de software se ha seguido con rigor y que el resto de módulos mantienen la misma consistencia en su diseño e implementación.

En cuanto a las tecnologías empleadas, cabe destacar el aprendizaje con **Angular**, que inicialmente presentó una curva de aprendizaje pronunciada, pero que finalmente se ha convertido en una de las herramientas favoritas de desarrollo por su potencia y versatilidad. Por otro lado, **Spring Boot** ha reafirmado la impresión positiva previa, consolidándose como un framework robusto y eficiente para el desarrollo de aplicaciones empresariales. En lo referente a herramientas de diseño, la experiencia con **Figma** ha sido menos satisfactoria, ya que al ser la primera vez que se utilizaba resultó algo lenta en determinados procesos, dificultando

parcialmente la elaboración de los *mockups*.

Desde un punto de vista académico, este Trabajo Fin de Grado ha sido una experiencia muy enriquecedora para comprender en profundidad la problemática del desarrollo de software. El proyecto ha permitido aplicar de manera práctica los conocimientos adquiridos en distintas asignaturas del grado, enfrentarse a decisiones de arquitectura y diseño en un caso real, y poner en práctica metodologías ágiles como Scrum para la organización en sprints.

Además, el desarrollo completo —desde el análisis de requisitos hasta la implementación y validación— ha puesto de manifiesto la importancia de seguir de forma disciplinada el proceso de ingeniería de software, mostrando cómo cada fase contribuye al éxito del producto final. En conjunto, este trabajo ha servido para reforzar competencias técnicas, mejorar la capacidad de planificación y gestión, y sentar unas bases sólidas para abordar futuros proyectos profesionales con mayor solvencia.

8.2. Líneas Futuras

Aunque el sistema desarrollado cubre satisfactoriamente los requisitos iniciales planteados, existen varias mejoras y funcionalidades que se han identificado como posibles líneas de evolución del proyecto. Estas se corresponden con los **requisitos futuros** definidos en el análisis, y su implementación permitiría ampliar el alcance y la utilidad de la aplicación:

- **Gestión avanzada de perfiles**: incorporación de fotos de perfil para los miembros y usuarios, mejorando la identificación visual y la usabilidad del sistema.
- **Visualización de información**: generación de gráficas y paneles dinámicos que faciliten el análisis de datos, como cuotas pagadas, número de miembros activos o distribución de salidas procesionales.
- **Listados predefinidos**: creación de listados por defecto que complementen los configurables ya existentes, permitiendo a los administradores obtener información recurrente de manera inmediata.
- **Integración de ChatBot con LLM**: desarrollo de un asistente conversacional basado en modelos de lenguaje que facilite la interacción con los usuarios, agilizando consultas frecuentes y ofreciendo soporte automatizado.

- **Pago telemático**: implementación de un sistema seguro de pago desde la aplicación, especialmente para cuotas específicas como la de hermano o la de salidas procesionales de mantillas, evitando la necesidad de acudir presencialmente a la sede de la cofradía.
- **Interfaz de superadministrador**: creación de un panel avanzado que permita gestionar múltiples cofradías, incluyendo la posibilidad de dar de alta nuevas hermandades y asignar o crear administradores específicos para cada una.

La incorporación de estas mejoras no solo incrementaría el valor añadido de la aplicación, sino que también respondería a nuevas demandas de los usuarios y cofradías, contribuyendo a una gestión más moderna, ágil y completamente digitalizada.

Referencias

[1] ThymeLeaf, "Thymeleaf official website." https://www.thymeleaf.org/, 2024. Accedido en agosto de 2025.

[2] Figma, "Qué es figma." https://help.figma.com/hc/es-419/articles/14563969806359--Qu%C3%A9-es-Figma, 2024. Accedido en agosto de 2025.

[3] P. Project, "Pencil project." https://pencil.evolus.vn/, 2024. Accedido en agosto de 2025.

[4] U. de Málaga, "Introducción a la ingeniería del software, tema 3: Procesos de software." Apuntes de clase, 2022. Apuntes de la asignatura, Grado en Ingeniería Informática.

[5] E. Medina and C. Rossi, "Requisitos de software." Universidad de Málaga, Material docente de Análisis y Diseño de Sistemas de Información (Curso 20/21), 2021.

[6] J. Dick, E. Hull, and K. Jackson, *Requirements Engineering*. Springer, 3rd ed., 2013.

[7] C. Rossi and E. Medina, "Casos de uso." Universidad de Málaga, Material docente de Análisis y Diseño de Sistemas de Información (Curso 20/21), 2021.

[8] C. Rossi and E. Medina, "Clases. curso 23/24." Apuntes de Ingeniería del Software, Universidad de Málaga, 2023. Tema: Diagramas de Clases.

[9] C. Rossi and E. Medina, "Diagramas de secuencia. curso 2022/23." Apuntes de Ingeniería del Software, Universidad de Málaga, 2023. Tema 5.3 del curso: Diagramas de Secuencia en UML.

Apéndice A
Documentación de Diseño Adicional

A.1. Especificación Completa de Casos de Uso

En este apartado del apéndice se recopilan de manera íntegra todas las especificaciones de los casos de uso definidos para el sistema que no hayan sido expuestas con anterioridad. Estos incluyen tanto las operaciones básicas de gestión (miembros, horquilleros, tallajes, pagos, salidas procesionales, préstamos, secciones y tronos) como las funcionalidades de configuración y utilidades adicionales. Su inclusión en este apartado tiene como objetivo mantener la claridad y concisión del cuerpo principal de la memoria, a la vez que se ofrece al lector una visión completa y detallada de todos los casos de uso que sustentan el sistema desarrollado.

CU1: Iniciar Sesión

CU2: Recuperar Contraseña

Actores Usuario no registrado.

Precondición El usuario no ha iniciado sesión en la aplicación.

Escenario de Éxito

1: En *Inicio de sesión*, pulsar en *Recuperar contraseña.*

2: Introducir correo electrónico.

3: El sistema comprueba si el correo existe y está habilitado.

4: El sistema envía un código temporal al correo.

5: Introducir el código recibido.

6: El sistema valida el código.

7: Introducir la nueva contraseña cumpliendo la política.

8: Pulsar en *Confirmar*.

Escenario Alternativo A1

5b Correo inexistente o inhabilitado.

6b El sistema muestra mensaje: “Correo no registrado o inhabilitado”.

7b Pulsar en *Aceptar*.

8b Volver a introducir correo.

Escenario Alternativo A2

7c Código incorrecto o caducado.

8c El sistema muestra mensaje: “Código inválido o expirado”.

9c Pulsar en *Aceptar*.

10c Volver a introducir código.

Postcondición La contraseña queda actualizada correctamente en la base de datos.

CU3: Modificar Datos Personales

Actores Usuario registrado.

Precondición Sesión iniciada como usuario registrado.

Escenario de Éxito

1: Pulsar en *Modificar datos*.

2: Introducir los datos a modificar.

3: El sistema valida formatos y campos obligatorios.

4: Pulsar en *Guardar*.

Escenario Alternativo A1

3b Algún dato es inválido.

4b El sistema muestra un mensaje de error indicando el campo afectado.

5b Pulsar en *Aceptar*.

6b Corregir datos e intentar de nuevo.

Postcondición Los datos válidos se actualizan correctamente en la base de datos.

CU4: Crear Miembro

CU5: Leer Miembros

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Pulsar sobre un miembro para abrir su ficha.

Postcondición Queda abierta la ficha del miembro buscado.

CU6: Editar Miembro

Actores Administrador.

Precondición Existencia del miembro a editar en la base de datos.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Buscar el miembro a editar.

3: Abrir su ficha.

4: Pulsar en *Editar*.

5: Introducir los nuevos datos.

6: El sistema valida formatos y reglas.

7: Pulsar en *Guardar*.

Escenario Alternativo A1

6b Algún dato es incorrecto.

7b El sistema muestra mensaje de error.

8b Pulsar en *Aceptar*.

9b Corregir datos.

Escenario Alternativo A2

3c Miembro no encontrado.

4c Pulsar en *Aceptar*.

5c Ajustar criterios de búsqueda y reintentar.

Postcondición Los datos se actualizan correctamente. Si se marca "Es hermano", se genera la cuota del año en curso.

CU7: Dar de Baja a un Miembro

Actores Administrador.

Precondición Existencia del miembro a dar de baja en la base de datos.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Buscar el miembro a dar de baja.

3: Abrir su ficha.

4: Pulsar en *Dar de baja*.

5: Introducir motivo.

6: Confirmar.

Escenario Alternativo A1

3b Miembro no encontrado.

4b Pulsar en *Aceptar*.

5b Revisar búsqueda y reintentar.

Postcondición El miembro queda en estado de baja, registrando motivo y fecha; no aparece en listados operativos.

CU8: Registrar Usuario

Actores Administrador.

Precondición Existencia del miembro; no debe estar ya habilitado como usuario.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Buscar el miembro.

3: Abrir su ficha.

4: Pulsar en *Dar de alta* (usuario).

5: Confirmar.

Escenario Alternativo A1

3b Miembro no encontrado.

4b Ajustar criterios de búsqueda y reintentar.

Escenario Alternativo A2

4c Miembro ya habilitado como usuario.

5c El sistema informa y cancela la operación.

Postcondición El correo queda habilitado para *Recuperar Contraseña* y posterior acceso.

CU9: Crear Horquillero

Actores Administrador.

Precondición No existe un horquillero con el mismo nombre y apellidos.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Pulsar en *Crear*.

3: Recuperar datos desde un miembro existente (búsqueda por nombre y apellidos).

4: Introducir datos de horquillería (talla, hombro preferente, observaciones).

5: El sistema valida.

6: Pulsar en *Guardar*.

Escenario Alternativo A1

3b Pulsar en *Crear de cero*.

4b Introducir datos del nuevo miembro y datos de horquillería.

5b El sistema valida.

Escenario Alternativo A2

4c Algún dato es incorrecto.

5c El sistema muestra error.

6c Pulsar en *Aceptar* y corregir.

Postcondición El horquillero queda creado y vinculado correctamente.

CU10: Leer Horquilleros

Actores Administrador.

Precondición Sesión como Administrador.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Pulsar sobre un horquillero para abrir su ficha.

Postcondición Queda abierta la ficha del horquillero.

CU11: Editar Horquillero

Actores Administrador.

Precondición Existencia del horquillero en la base de datos.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Buscar el horquillero.

3: Abrir su ficha.

4: Pulsar en *Editar*.

5: Introducir datos nuevos (talla, hombro preferente, etc.).

6: El sistema valida.

7: Pulsar en *Guardar*.

Escenario Alternativo A1

6b Algún dato es incorrecto.

7b El sistema muestra error.

8b Pulsar en *Aceptar* y corregir.

Escenario Alternativo A2

3c Horquillero no encontrado.

4c Pulsar en *Aceptar* y volver a buscar.

Postcondición Los datos quedan actualizados correctamente.

CU12: Dar de Baja a un Horquillero

Actores Administrador.

Precondición Existencia del horquillero a dar de baja.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Buscar el horquillero.

3: Abrir su ficha.

4: Pulsar en *Dar de baja*.

5: Introducir motivo y confirmar.

Escenario Alternativo A1

3b Horquillero no encontrado.

4b Pulsar en *Aceptar* y reintentar búsqueda.

Postcondición El horquillero queda en estado de baja, registrando motivo y fecha.

CU13: Crear Tallaje

Actores Administrador.

Precondición Existencia del horquillero.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Buscar la ficha del horquillero.

3: Pulsar en *Nuevo tallaje*.

4: Introducir datos del tallaje.

5: El sistema valida.

6: Pulsar en *Guardar*.

Escenario Alternativo A1

4b Algún dato es incorrecto.

5b El sistema muestra error.

6b Pulsar en *Aceptar* y corregir.

Postcondición El nuevo tallaje queda registrado correctamente.

CU14: Leer Tallaje

Actores Administrador.

Precondición Existencia del horquillero.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Pulsar sobre un horquillero para abrir su ficha y ver el histórico de tallajes.

Postcondición Se visualiza el histórico de tallajes del horquillero.

CU15: Editar Tallaje

Actores Administrador.

Precondición Existencia del tallaje a editar.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Buscar el horquillero.

3: Abrir su ficha.

4: Pulsar sobre el tallaje.

5: Pulsar en *Editar*.

6: Introducir los nuevos datos.

7: El sistema valida.

8: Pulsar en *Guardar*.

Escenario Alternativo A1

6b Algún dato es incorrecto.

7b El sistema muestra error.

8b Pulsar en *Aceptar* y corregir.

Escenario Alternativo A2

3c Horquillero no encontrado.

4c Pulsar en *Aceptar*.

5c Ajustar búsqueda y reintentar.

Postcondición El tallaje queda actualizado correctamente.

CU16: Eliminar Tallaje

Actores Administrador.

Precondición Existencia del tallaje a eliminar.

Escenario de Éxito

1: Acceder a *Horquilleros*.

2: Buscar el horquillero.

3: Abrir su ficha.

4: Seleccionar el tallaje a eliminar.

5: Pulsar en *Eliminar*.

6: El sistema solicita confirmación.

7: Confirmar.

Escenario Alternativo A1

3b Horquillero o tallaje no encontrado.

4b Pulsar en *Aceptar* y reintentar búsqueda.

Postcondición El tallaje queda eliminado de la base de datos.

CU17: Auto-asignación de Horquilleros

Actores Administrador.

Precondición Configuración de trono/varales y horquilleros disponibles.

Escenario de Éxito

1: Acceder a *Plantilla de horquilleros*.

2: Pulsar en *Auto-asignación*.

3: El sistema calcula una propuesta (mismo varal del año anterior y orden por talla; nuevas altas al varal con menor dotación respetando hombro preferente cuando sea posible).

4: El sistema muestra una previsualización.

5: Confirmar auto-asignación.

Postcondición Las asignaciones quedan guardadas correctamente.

CU18: Crear Salida Procesional

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Buscar la ficha del miembro.

3: Pulsar en *Nueva salida procesional*.

4: Introducir datos de la salida (trono, sección, puesto y, si aplica, túnica/capirote/capa).

5: El sistema valida.

6: Pulsar en *Guardar*.

Escenario Alternativo A1

4b Algún dato es incorrecto.

5b El sistema muestra error.

6b Pulsar en *Aceptar* y corregir.

Postcondición La salida queda creada y asociada al miembro.

CU19: Leer Salida Procesional

Actores Administrador.

Precondición Existencia del miembro.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Abrir la ficha del miembro y visualizar el histórico de salidas.

Postcondición Queda visible el histórico de salidas del miembro.

CU20: Editar Salida Procesional

Actores Administrador.

Precondición Existencia de la salida procesional a editar.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Buscar el miembro.

3: Abrir su ficha.

4: Seleccionar la salida procesional.

5: Pulsar en *Editar*.

6: Introducir los nuevos datos.

7: El sistema valida.

8: Pulsar en *Guardar*.

Escenario Alternativo A1

6b Algún dato es incorrecto.

7b El sistema muestra error.

8b Pulsar en *Aceptar* y corregir.

Escenario Alternativo A2

3c Miembro no encontrado.

4c Pulsar en *Aceptar*.

5c Ajustar búsqueda y reintentar.

Postcondición La salida queda actualizada correctamente.

CU21: Eliminar una Salida Procesional

Actores Administrador.

Precondición Existencia de la salida a eliminar.

Escenario de Éxito

1: Acceder a *Miembros*.

2: Buscar el miembro y abrir su ficha.

3: Seleccionar la salida a eliminar.

4: Pulsar en *Eliminar*.

5: El sistema solicita confirmación y, si procede, la acción sobre la cuota (devolución o convertir en donativo).

6: Confirmar.

Escenario Alternativo A1

3b Miembro o salida no encontrada.

4b Pulsar en *Aceptar* y reintentar.

Postcondición La salida se elimina y se aplica la acción indicada sobre la cuota, si corresponde.

CU22: Auto-asignación Cortejo Procesional

Actores Administrador.

Precondición Participantes y secciones configuradas.

Escenario de Éxito

1: Acceder a *Plantilla* desde *Miembros*.

2: Pulsar en *Auto-asignación*.

3: El sistema ordena por año de nacimiento y, en empate, por antigüedad.

4: El sistema muestra una previsualización.

5: Confirmar.

Postcondición Las asignaciones quedan guardadas correctamente.

CU23: Crear Pago

Actores Administrador (Tesorería).

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Tesorería*.

2: Pulsar en *Crear*.

3: (Opcional) Buscar y seleccionar miembro.

4: Introducir importe y descripción.

5: El sistema valida importe y formato.

6: Pulsar en *Pagado*.

Escenario Alternativo A1

3b Pulsar en *Saltar* (no asignar miembro).

4b Introducir datos de cobro.

Escenario Alternativo A2

4c Algún dato es incorrecto.

5c El sistema muestra error.

6c Pulsar en *Aceptar* y corregir.

Postcondición El pago queda registrado correctamente en la base de datos.

CU24: Leer Pago

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Tesorería*.

2: Pulsar sobre un pago para abrir su ficha.

Postcondición Queda abierta la ficha del pago seleccionado.

CU25: Editar Pago

Actores Administrador.

Precondición Existencia del pago a editar.

Escenario de Éxito

1: Acceder a *Tesorería*.

2: Buscar el pago a editar y abrir su ficha.

3: Pulsar en *Editar*.

4: Introducir los nuevos datos.

5: El sistema valida.

6: Pulsar en *Pagado*.

Escenario Alternativo A1

6b Algún dato es incorrecto.

7b El sistema muestra error.

8b Pulsar en *Aceptar* y corregir.

Escenario Alternativo A2

3c Pago no encontrado.

4c Pulsar en *Aceptar* y reintentar búsqueda.

Postcondición El pago queda actualizado correctamente.

CU26: Eliminar un Pago

Actores Administrador.

Precondición Existencia del pago a eliminar.

Escenario de Éxito

1: Acceder a *Tesorería*.

2: Buscar el pago y abrir su ficha.

3: Pulsar en *Eliminar*.

4: El sistema solicita confirmación.

5: Confirmar.

Escenario Alternativo A1

3b Pago no encontrado.

4b Pulsar en *Aceptar* y reintentar búsqueda.

Postcondición El pago queda eliminado correctamente.

CU27: Arqueo Diario de Caja

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Tesorería*.

2: Pulsar en *Arqueo diario*.

3: Seleccionar la fecha.

4: Pulsar en *Confirmar*.

Escenario Alternativo A1

3b La fecha seleccionada no tiene pagos.

4b El sistema informa: "No hay pagos en la fecha seleccionada".

5b Pulsar en *Aceptar*.

6b Seleccionar otra fecha.

Postcondición Se genera un documento con el detalle de pagos y el total del día.

CU28: Crear Préstamo

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Tras crear una *Salida procesional*, pulsar en *Crear préstamo*.

2: Introducir prendas (túnica, velillo, etc.).

3: El sistema valida.

4: Pulsar en *Guardar*.

Escenario Alternativo A1

3b Algún dato es incorrecto.

4b El sistema muestra error.

5b Pulsar en *Aceptar* y corregir.

Postcondición El préstamo queda registrado correctamente.

CU29: Leer Préstamos

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Inventario*.

2: Pulsar sobre un préstamo para abrir su ficha.

Postcondición Queda abierta la ficha del préstamo seleccionado.

CU30: Editar Préstamo

Actores Administrador.

Precondición Existencia del préstamo a editar.

Escenario de Éxito

1: Acceder a *Inventario*.

2: Buscar el préstamo y abrir su ficha.

3: Pulsar en *Editar*.

4: Introducir nuevos datos.

5: El sistema valida.

6: Pulsar en *Guardar*.

Escenario Alternativo A1

6b Algún dato es incorrecto.

7b El sistema muestra error.

8b Pulsar en *Aceptar* y corregir.

Escenario Alternativo A2

3c Préstamo no encontrado.

4c Pulsar en *Aceptar* y reintentar búsqueda.

Postcondición El préstamo queda actualizado correctamente.

CU31: Devolución Préstamo

Actores Administrador.

Precondición Existencia del préstamo a devolver.

Escenario de Éxito

1: Acceder a *Inventario*.

2: Buscar el préstamo y abrir su ficha.

3: Pulsar en *Devolución*.

4: El sistema solicita confirmación.

5: Confirmar.

Escenario Alternativo A1

3b Préstamo inexistente.

4b Pulsar en *Aceptar* y reintentar búsqueda.

Postcondición El préstamo queda marcado como devuelto (completado).

CU32: Devolución Préstamo Masiva

Actores Administrador.

Precondición Existencia de al menos un préstamo activo.

Escenario de Éxito

1: Acceder a *Inventario*.

2: Pulsar en *Devolución masiva*.

3: El sistema solicita confirmación.

4: Confirmar.

Postcondición Todos los préstamos activos quedan marcados como devueltos.

CU33: Listados

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Listados*.

2: Seleccionar campos a mostrar (p. ej., nombre, teléfono).

3: Seleccionar filtros (p. ej., horquilleros de un trono, penitentes de una sección).

4: Pulsar en *Continuar*.

Postcondición Se genera un documento con los datos solicitados aplicando los filtros seleccionados.

CU34: Configurar Cuotas

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Seleccionar *Configurar cuotas*.

3: Introducir los nuevos importes.

4: El sistema valida.

5: Pulsar en *Guardar*.

Escenario Alternativo A1

4b Algún dato es incorrecto.

5b El sistema muestra error.

6b Pulsar en *Aceptar* y corregir.

Postcondición Las cuantías de las cuotas quedan actualizadas correctamente.

CU35: Limpieza de Bajas

Actores Administrador.

Precondición Sesión iniciada como Administrador; umbral de años definido en configuración.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Limpieza de bajas*.

3: Pulsar en *Confirmar*.

Postcondición Se eliminan definitivamente las fichas en baja con antigüedad superior al umbral y los préstamos devueltos anteriores a dicho umbral.

CU36: Crear Sección Penitentes

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Sección*.

3: Pulsar en *Crear*.

4: Introducir datos del nuevo puesto (cortejo Cristo/Virgen, nombre, cupo máximo).

5: El sistema valida.

6: Pulsar en *Guardar*.

Escenario Alternativo A1

5b Algún dato es incorrecto.

6b El sistema muestra error.

7b Pulsar en *Aceptar* y corregir.

Postcondición La nueva sección/puesto queda creada correctamente.

CU37: Leer Sección

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Sección*.

3: Pulsar sobre una sección para abrir su ficha.

Postcondición Queda abierta la ficha de la sección buscada.

CU38: Editar Sección

Actores Administrador.

Precondición Existencia de la sección a editar.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Sección*.

3: Buscar la sección.

4: Abrir la ficha.

5: Pulsar en *Editar*.

6: Introducir los nuevos datos (nombre, cupo).

7: El sistema valida.

8: Pulsar en *Guardar*.

Escenario Alternativo A1

7b Algún dato es incorrecto.

8b El sistema muestra error.

9b Pulsar en *Aceptar* y corregir.

Escenario Alternativo A2

3c Sección no encontrada.

4c Ajustar búsqueda y reintentar.

Postcondición La sección queda actualizada correctamente.

CU39: Eliminar Sección Penitentes

Actores Administrador.

Precondición Existencia de la sección a eliminar.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Sección*.

3: Buscar la sección y abrir su ficha.

4: Pulsar en *Eliminar*.

5: El sistema solicita confirmación.

6: Confirmar.

Escenario Alternativo A1

4b Sección no encontrada.

5b Ajustar búsqueda y reintentar.

Postcondición La sección queda eliminada correctamente.

CU40: Crear Trono

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Tronos*.

3: Pulsar en *Crear*.

4: Introducir datos del nuevo trono (cortejo, nombre, parámetros iniciales).

5: El sistema valida.

6: Pulsar en *Guardar*.

Escenario Alternativo A1

5b Algún dato es incorrecto.

6b El sistema muestra error.

7b Pulsar en *Aceptar* y corregir.

Postcondición El nuevo trono queda creado correctamente.

CU41: Leer Tronos

Actores Administrador.

Precondición Sesión iniciada como Administrador.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Tronos*.

3: Pulsar sobre un trono para abrir su ficha.

Postcondición Queda abierta la ficha del trono.

CU42: Editar Trono

Actores Administrador.

Precondición Existencia del trono a editar.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Tronos*.

3: Buscar y abrir el trono.

4: Pulsar en *Editar*.

5: Modificar parámetros (número de varales, plazas por varal, etc.).

6: El sistema valida.

7: Pulsar en *Guardar*.

Escenario Alternativo A1

7b Algún dato es incorrecto.

8b El sistema muestra error.

9b Pulsar en *Aceptar* y corregir.

Escenario Alternativo A2

3c Trono no encontrado.

4c Ajustar búsqueda y reintentar.

Postcondición El trono queda actualizado correctamente.

CU43: Dar de Baja un Trono

Actores Administrador.

Precondición Existencia del trono a dar de baja.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Tronos*.

3: Buscar el trono y abrir su ficha.

4: Pulsar en *Dar de baja*.

5: El sistema solicita confirmación.

6: Confirmar.

Escenario Alternativo A1

4b Trono no encontrado.

5b Ajustar búsqueda y reintentar.

Postcondición El trono queda en estado de baja y se excluye de listados operativos.

CU44: Hacer Visible Puestos

Actores Administrador.

Precondición Los puestos no están visibles.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Hacer visibles los puestos*.

3: Confirmar.

Postcondición Los usuarios ven su puesto asignado en su perfil.

CU45: Hacer Invisible Puestos

Actores Administrador.

Precondición Los puestos están visibles.

Escenario de Éxito

1: Acceder a *Configuración*.

2: Pulsar en *Hacer invisibles los puestos*.

3: Confirmar.

Postcondición Los puestos dejan de ser visibles para los usuarios.

La tabla de trazabilidad tiene como objetivo establecer una correspondencia clara entre los requisitos definidos y los casos de uso o funcionalidades implementadas en la aplicación. De este modo, se garantiza que cada requisito ha sido tenido en cuenta durante el desarrollo, facilitando tanto la validación como la verificación del sistema. Además, permite identificar de manera sencilla qué partes del software están vinculadas a cada requisito, lo que resulta útil para el mantenimiento y la evolución futura del proyecto.

Requisito	**Casos de Uso relacionados**
RF1	CU01
RF2	CU02
RF3	CU03
RF4	(flujo inicial)
RF5	CU04–CU07
RF6	CU08
RF7	CU18–CU21
RF8	CU09–CU12
RF9	CU13–CU16
RF10	CU17
RF11	CU22
RF12	CU23–CU26
RF13	CU27
RF14	CU28–CU32
RF15	CU33
RF16	CU34–CU35, CU44–CU45
RF17	CU36–CU39
RF18	CU40–CU43
RNF1	(criterio transversal)
RNF2	CU01–CU02 (seguridad credenciales)
RNF3	(código hecho para diferentes cofradías)
RNF4	(swagger y manual de usuario)

RFF1	(futuro)
RFF2	(futuro)
RFF3	(futuro)
RFF4	(futuro)
RFF5	(futuro)
RFF6	(futuro)

Cuadro 5: Matriz de trazabilidad entre requisitos y casos de uso

A.2. MockUp Completo

Este apartado del apéndice recopila las pantallas del *mockup* realizado con Figma que ilustran la estética y los flujos principales de la aplicación (p. ej., *landing*, inicio de sesión, panel de administración y formularios de gestión). Las capturas sirven como referencia visual del diseño planteado antes del desarrollo.

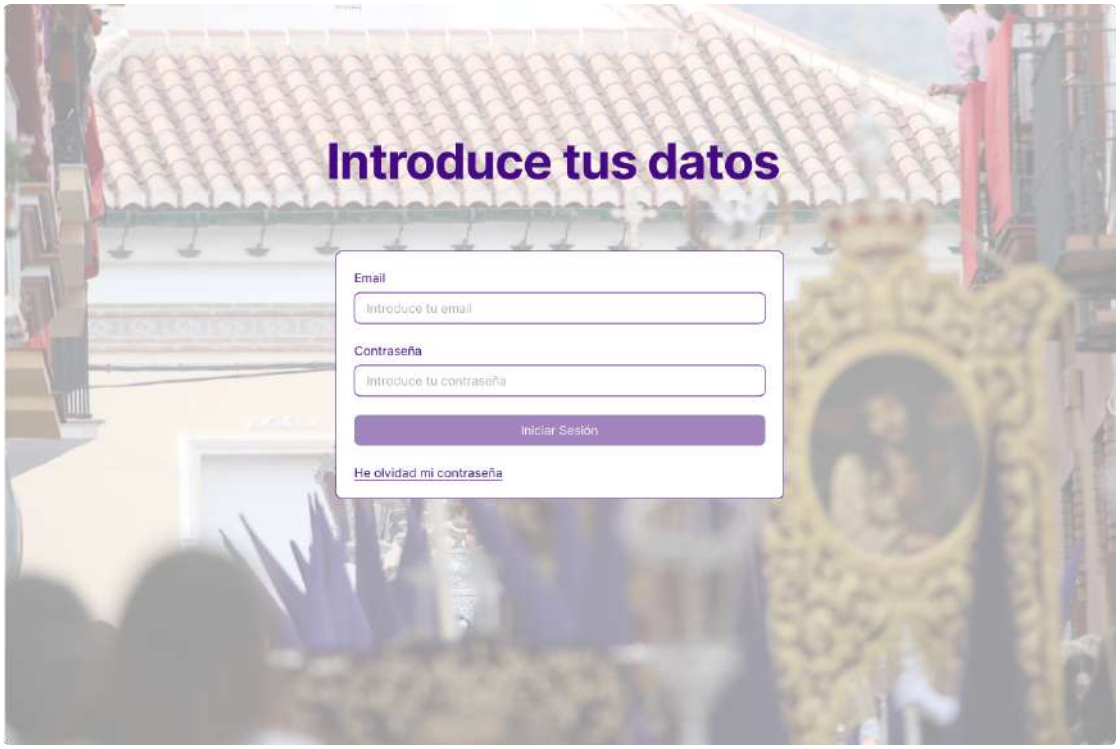


Figura 28: Pantalla de inicio de sesión sin datos

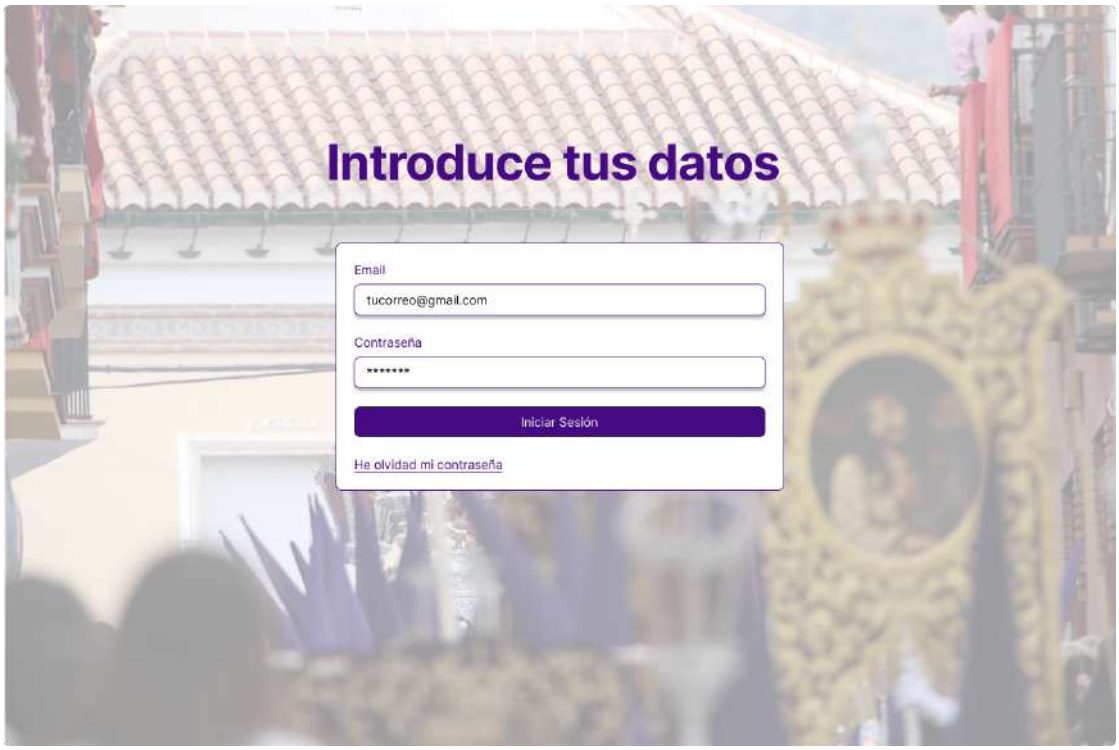


Figura 29: Pantalla de inicio de sesión con datos de usuario

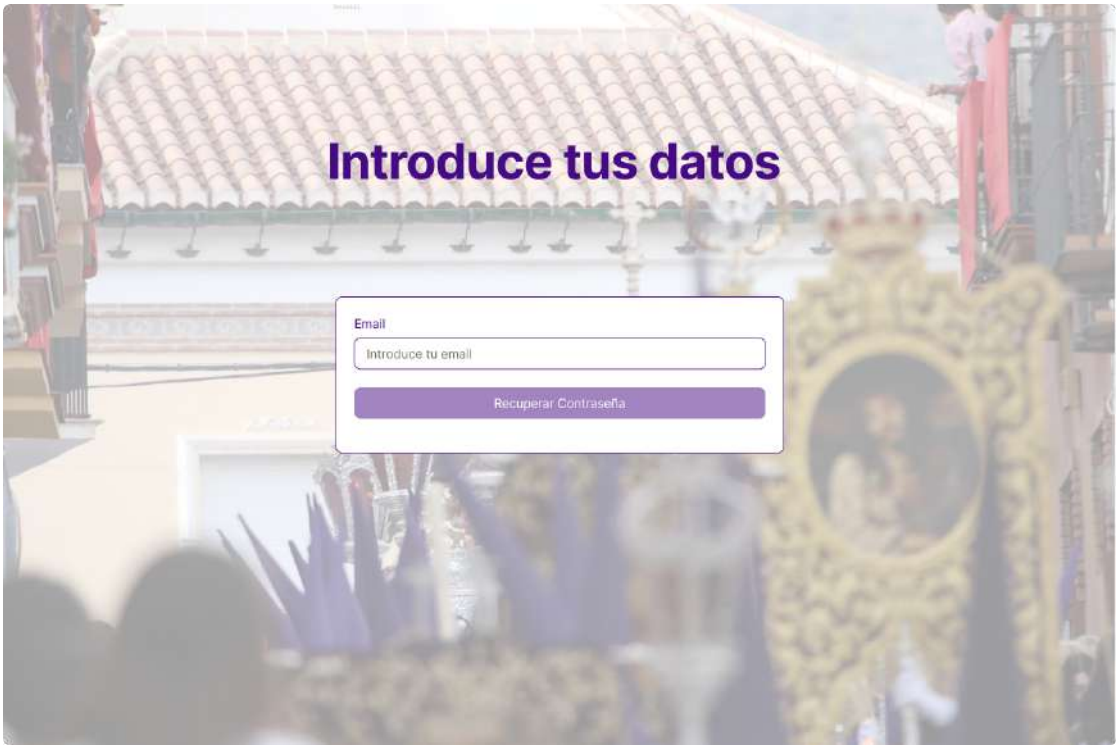


Figura 30: Pantalla inicial de recuperación de contraseña

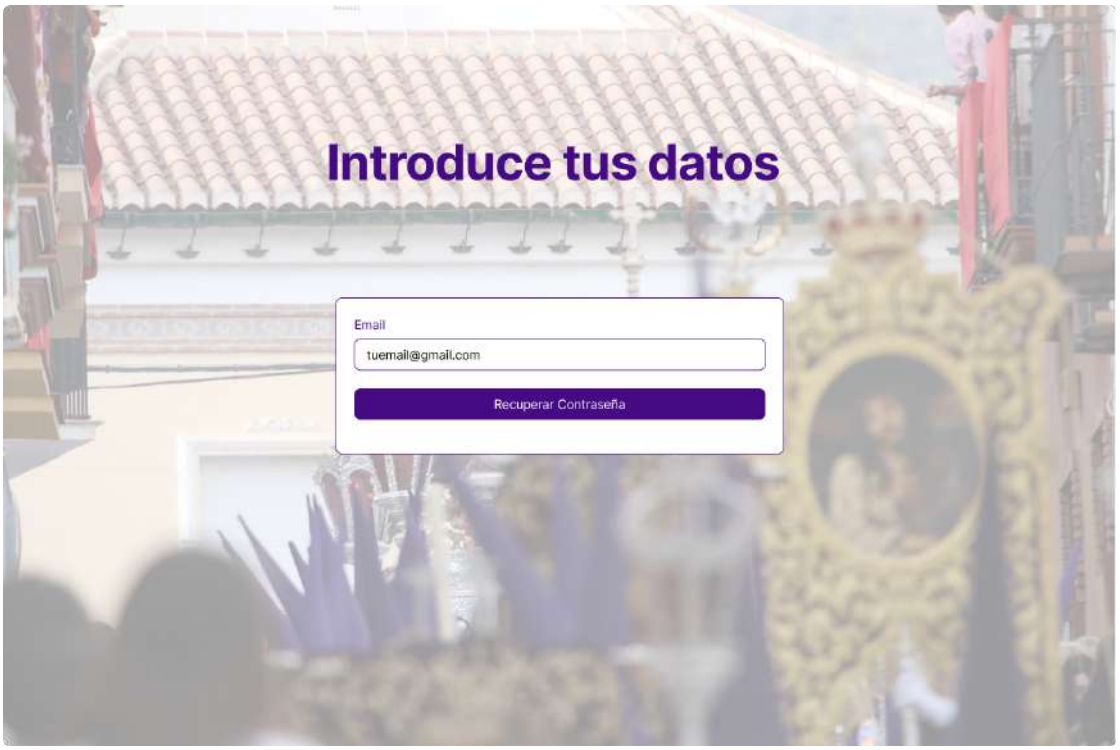


Figura 31: Recuperación de contraseña – introducción de datos

Figura 32: Recuperación de contraseña – segunda pantalla

Figura 33: Recuperación de contraseña – paso 2 con datos

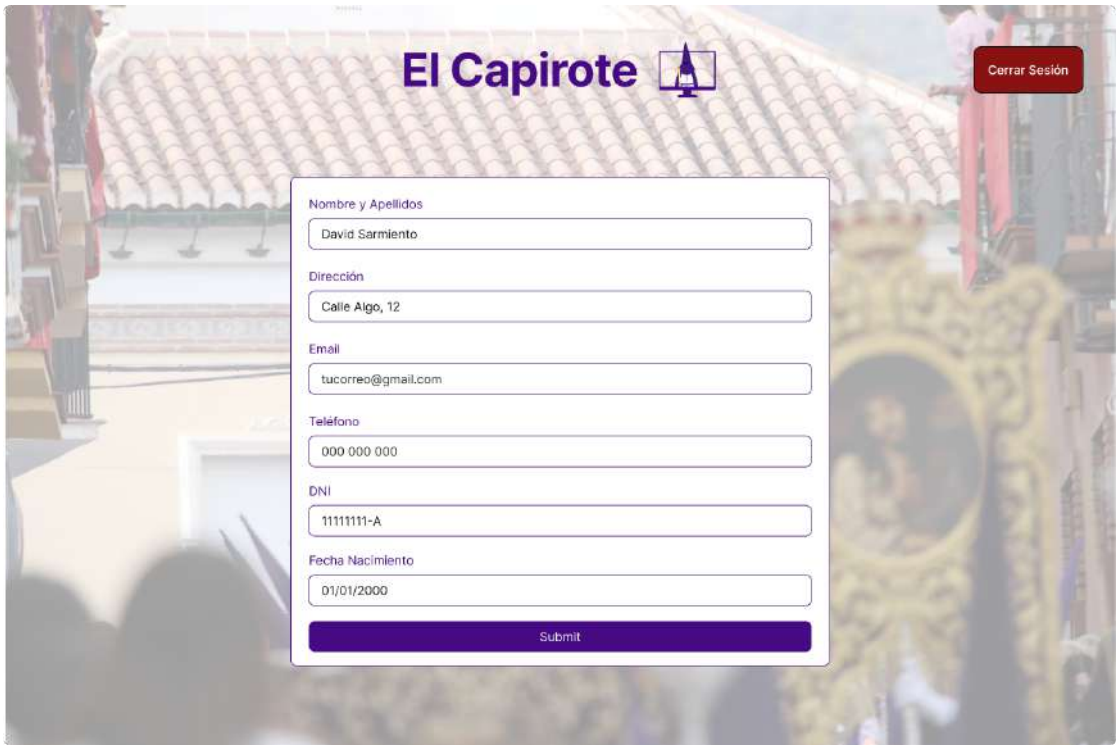


Figura 34: Pantalla de cambio de datos de usuario

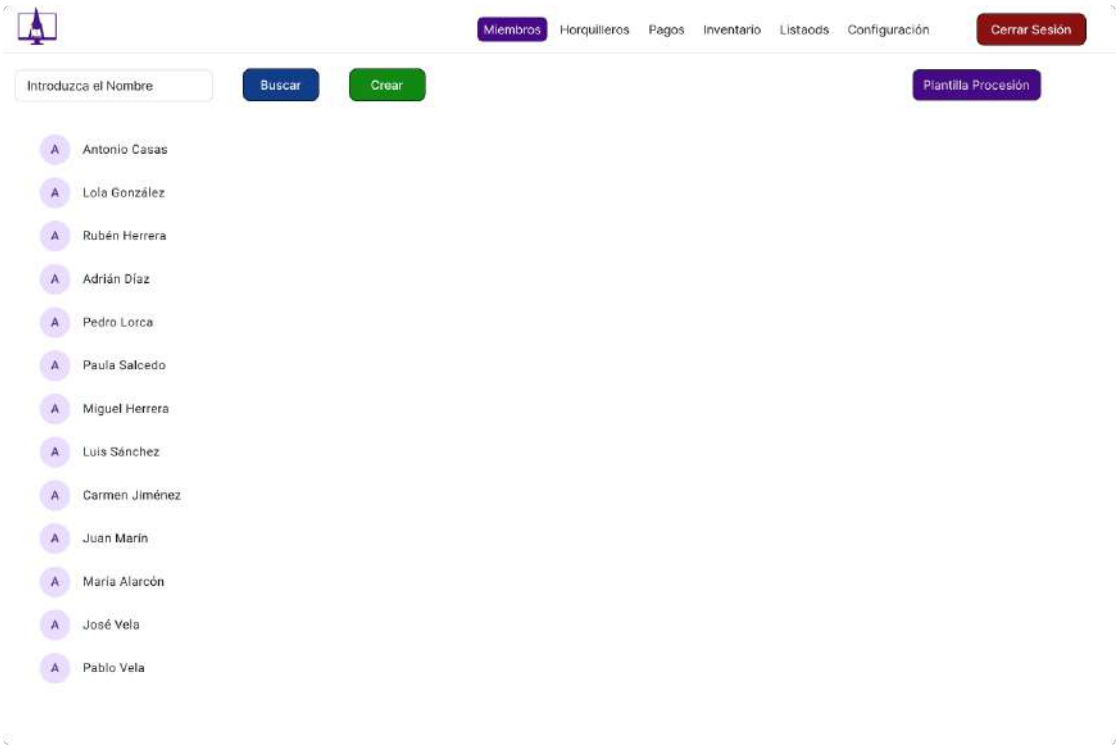


Figura 35: Gestión de miembros – listado principal

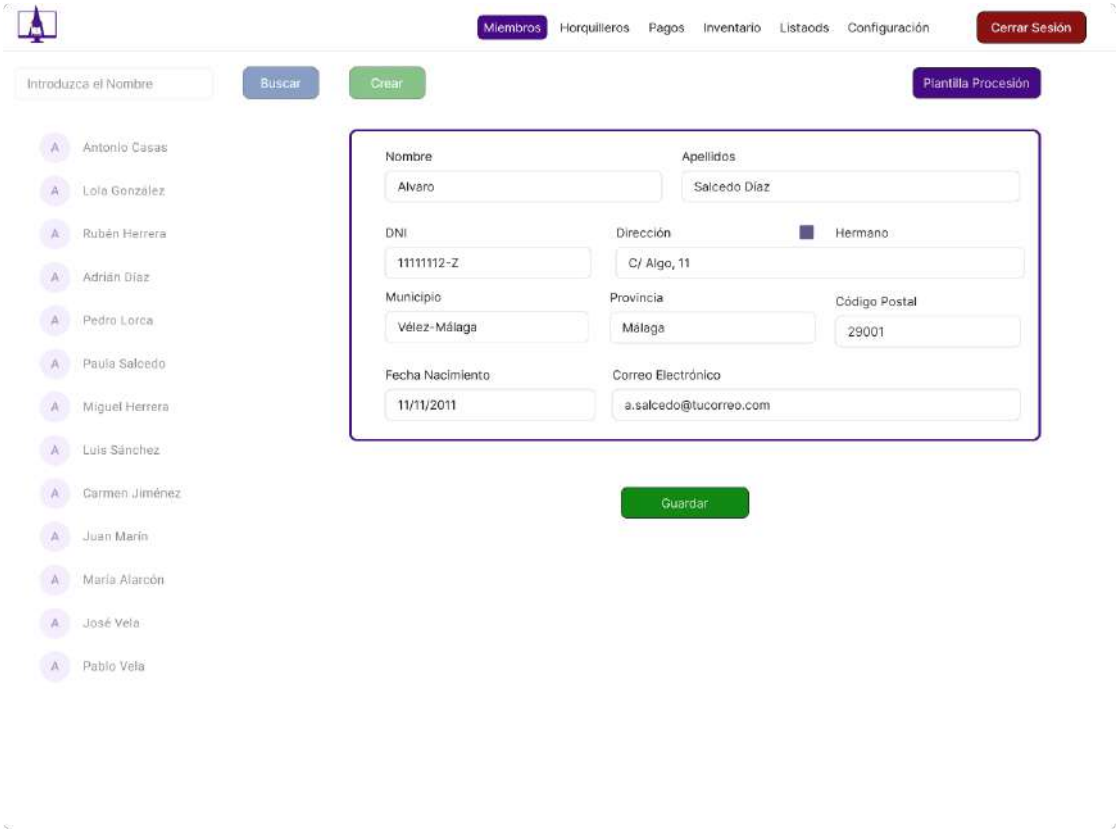


Figura 36: Formulario para crear un nuevo miembro

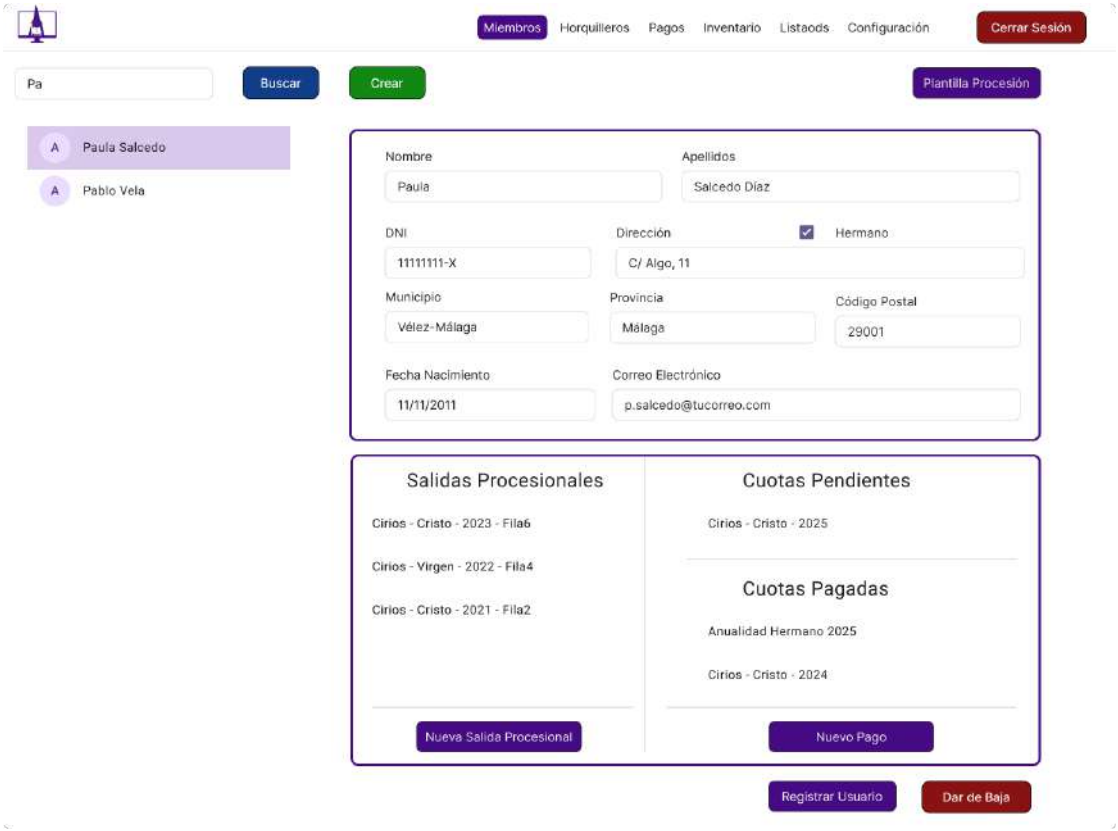


Figura 37: Búsqueda de miembros

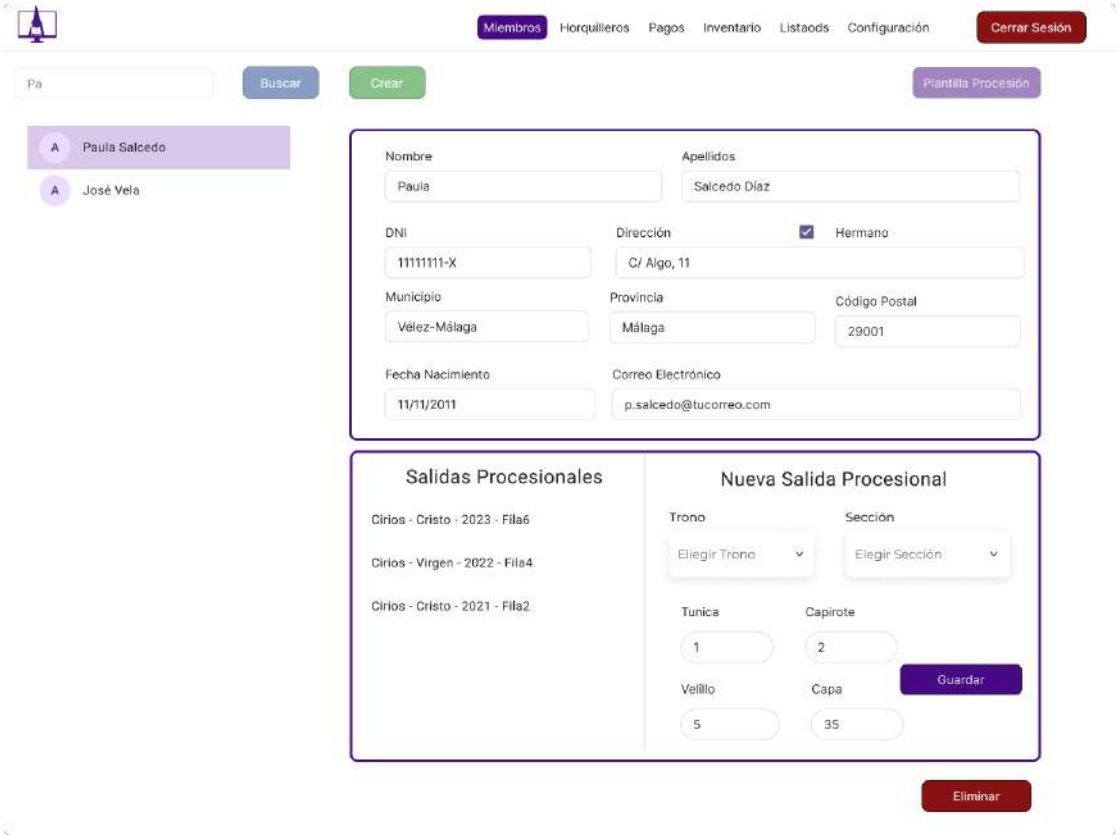


Figura 38: Asignación de una nueva salida a un miembro

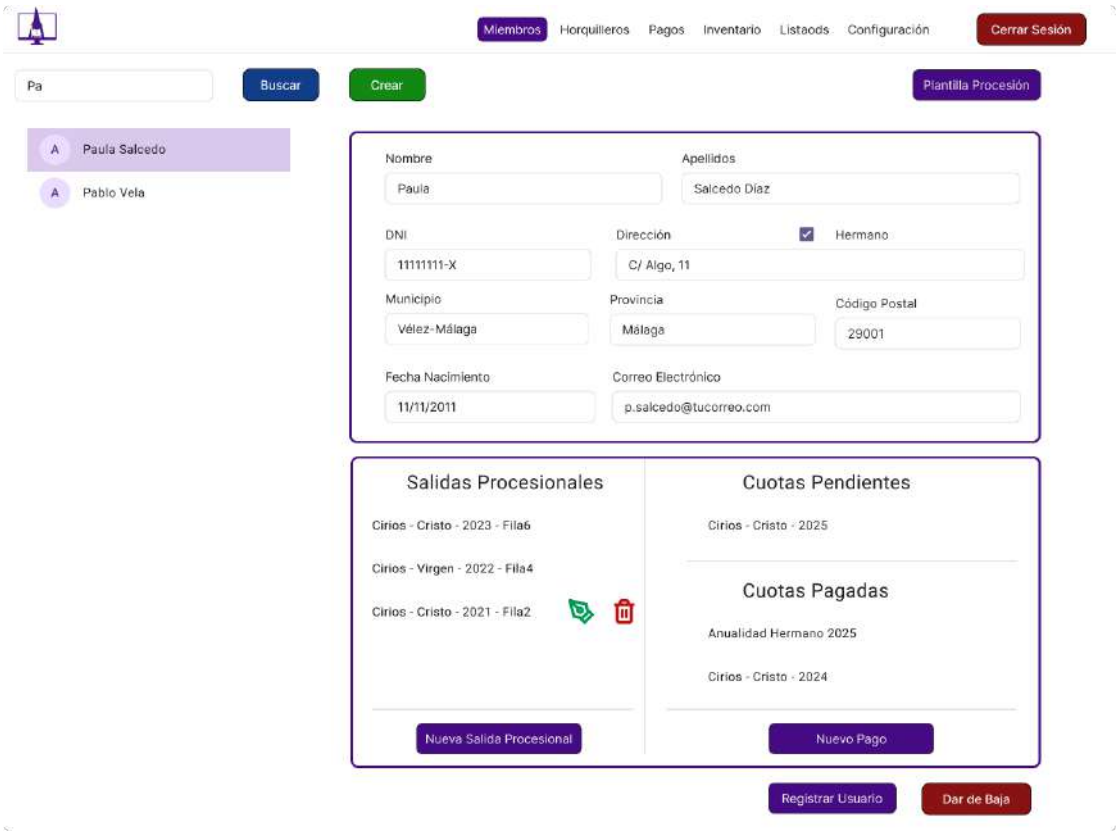


Figura 39: Vista de salidas procesionales – acción presionada

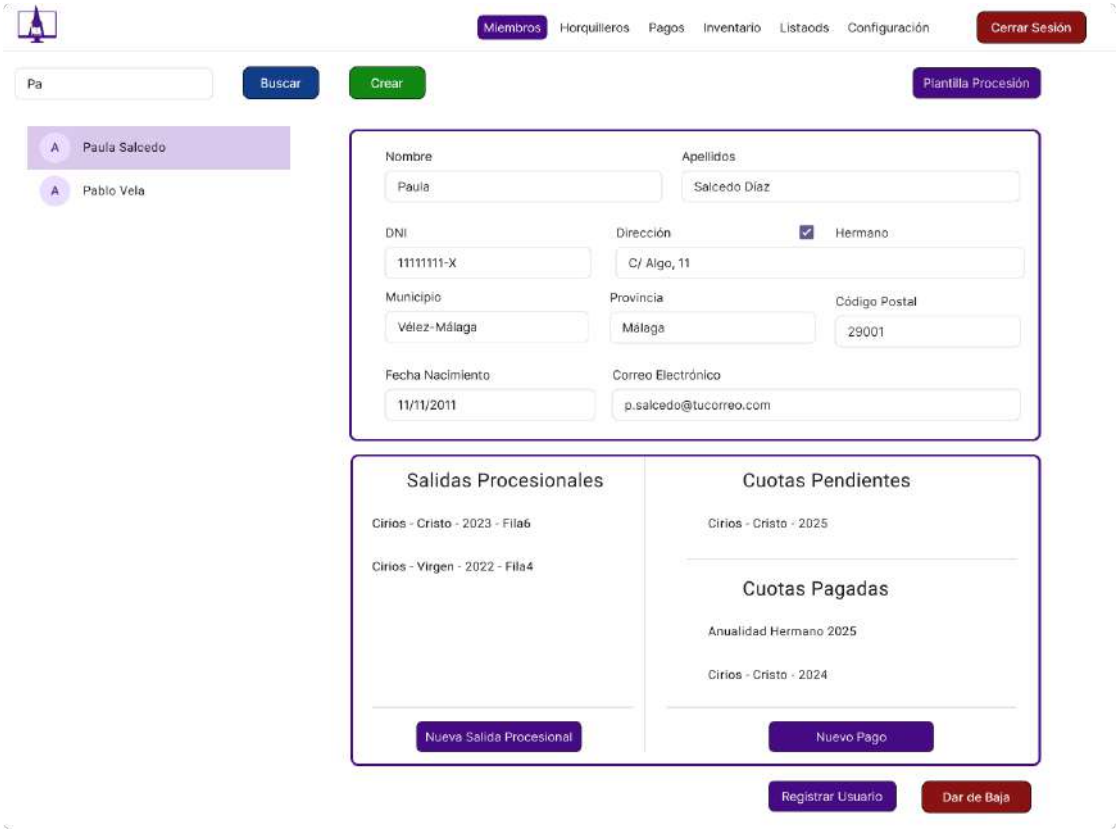


Figura 40: Vista de salidas procesionales – elemento eliminado

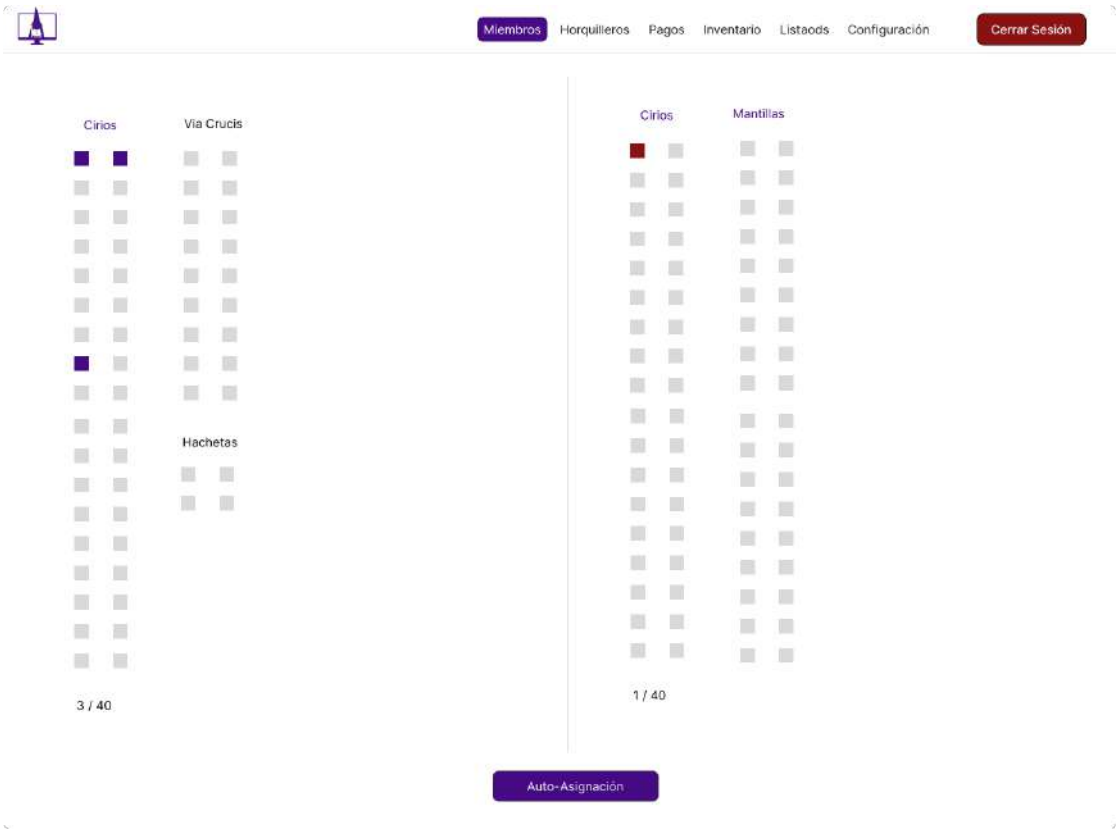


Figura 41: Gestión de plantilla de miembros

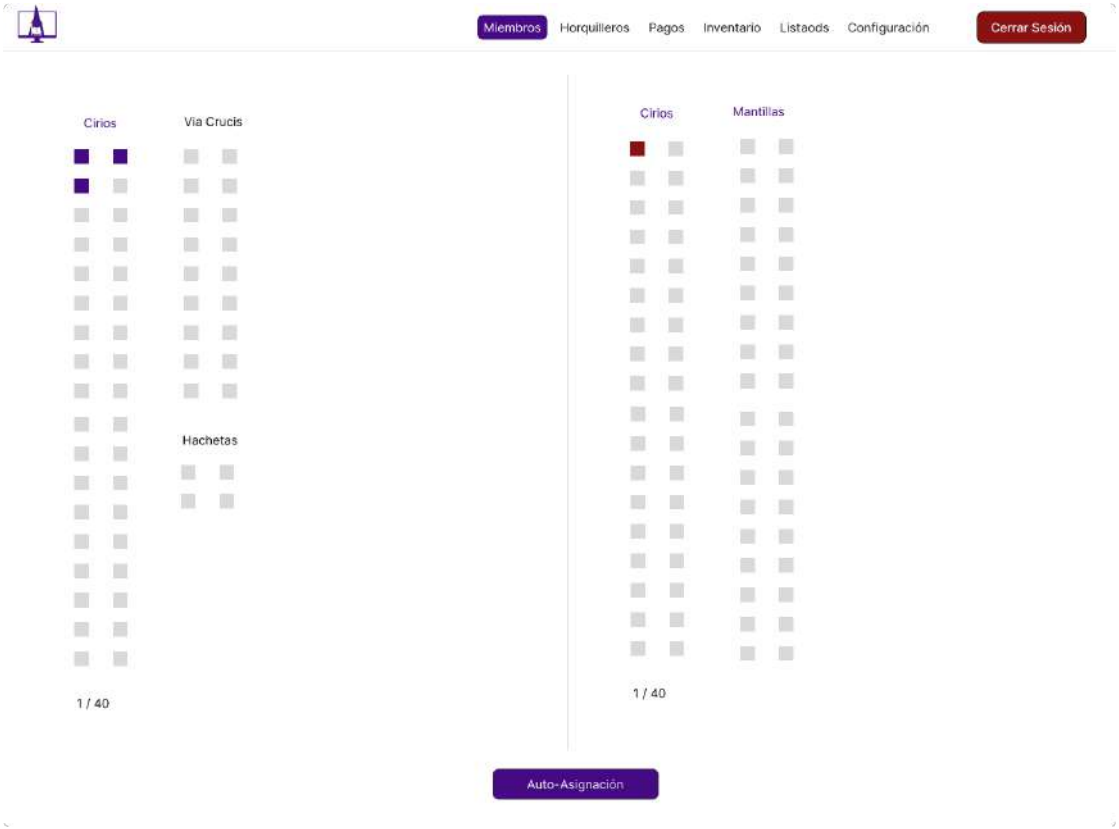


Figura 42: Gestión de plantilla – segunda vista

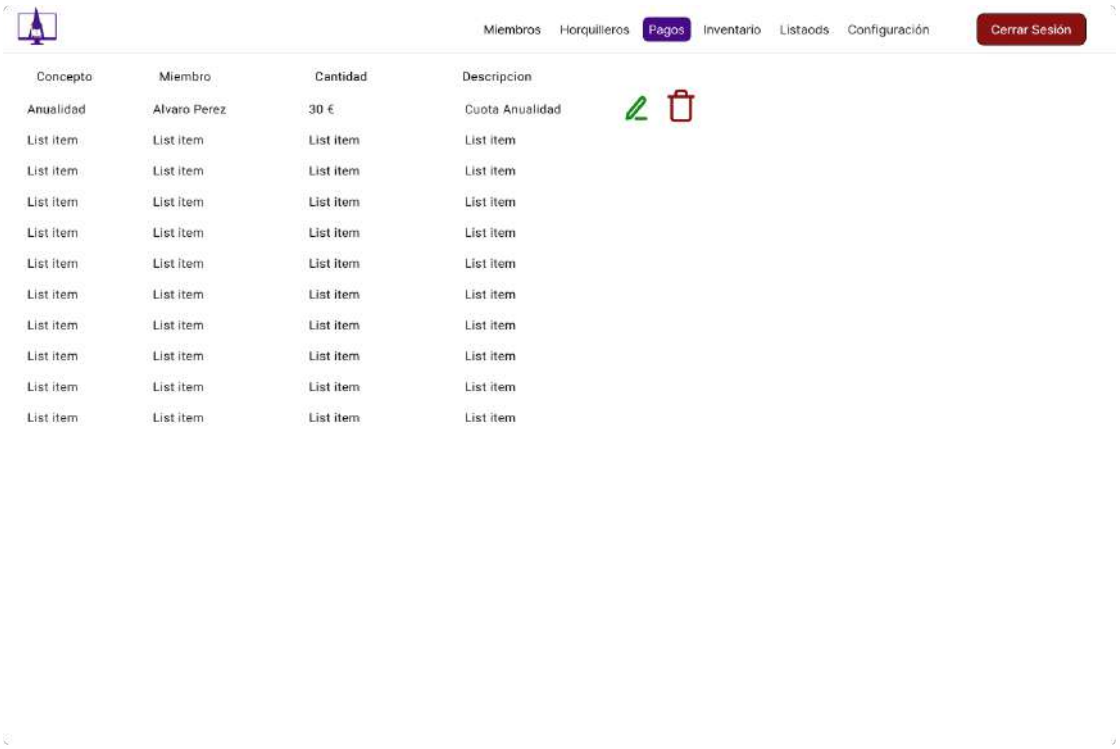


Figura 43: Detalle de pagos – acción presionada

A.3. Diagramas de Secuencia Adicionales

Este apartado del apéndice recopila los diagramas de secuencia complementarios que no se han incluido en el cuerpo principal de la memoria para evitar sobrecargarlo. En ellos se detallan otros casos de uso significativos de la aplicación, ilustrando el flujo de mensajes entre los distintos objetos y servicios implicados. Su consulta permite completar la visión global del sistema y comprender mejor la dinámica interna de sus procesos.

A.3.1. Baja de Miembro

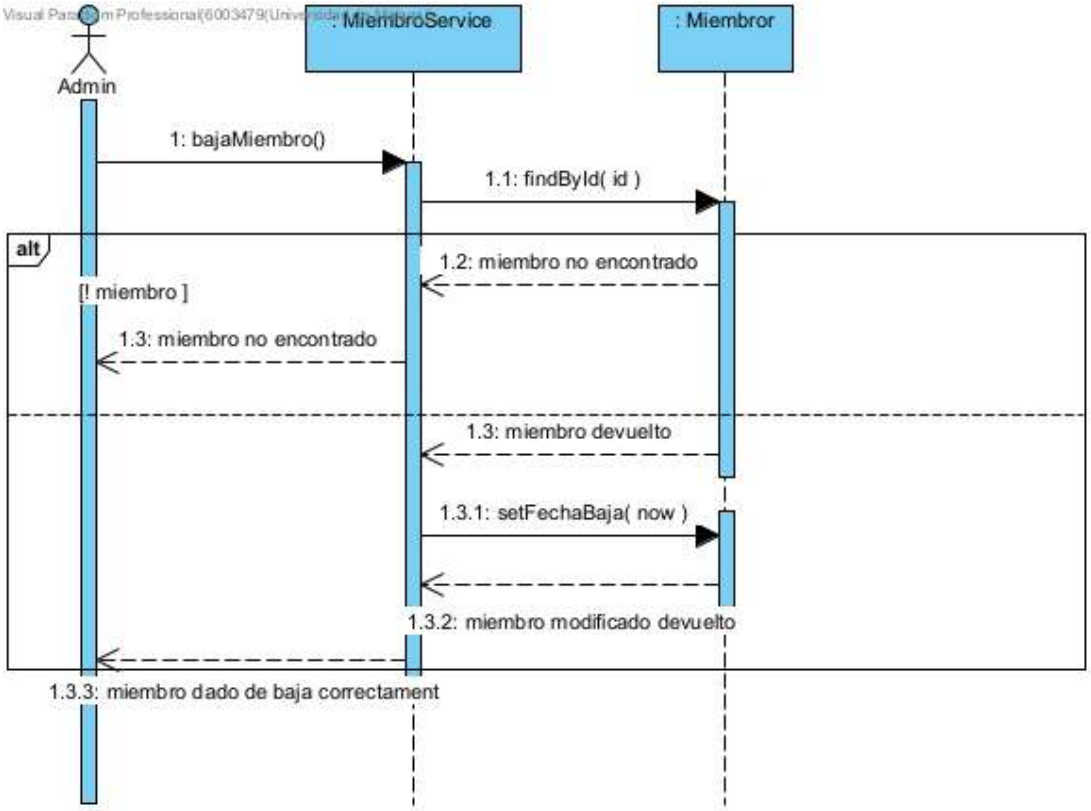


Figura 44: Diagrama de secuencia para la baja de un miembro.

Este diagrama refleja el proceso seguido cuando un administrador procede a dar de baja a un miembro del sistema. El flujo comienza con la invocación del método bajaMiembro(), que consulta la existencia del miembro mediante el servicio correspondiente. En caso de que el miembro no exista, se devuelve un mensaje de error. Si existe, se actualiza el atributo fechaBaja y se confirma que el miembro ha sido dado de baja correctamente. El uso de fragmentos alt permite contemplar ambos escenarios.

A.3.2. Edición de Datos Personales

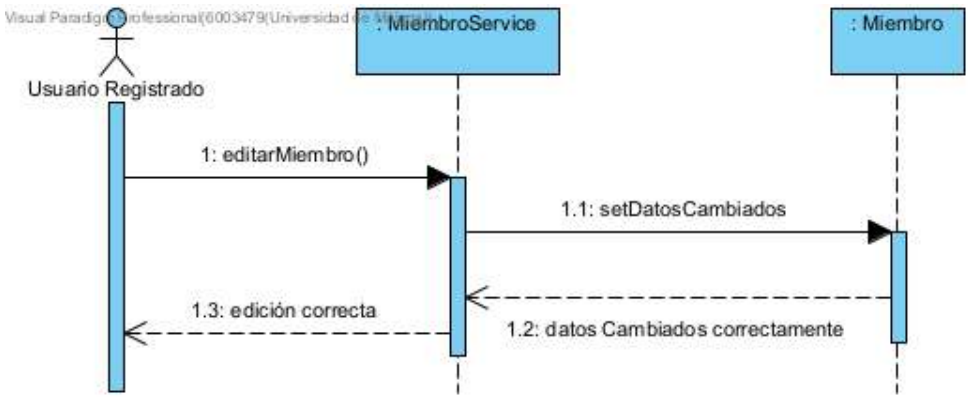


Figura 45: Diagrama de secuencia para la edición de datos personales.

Este diagrama corresponde al caso en que un usuario registrado edita sus propios datos personales. El flujo principal es sencillo: el usuario invoca el servicio editarMiembro(), el cual actualiza los atributos de la clase Miembro y confirma la edición exitosa. Al no existir ramificaciones alternativas, el diagrama refleja un escenario lineal.

A.3.3. Recuperación de Contraseña

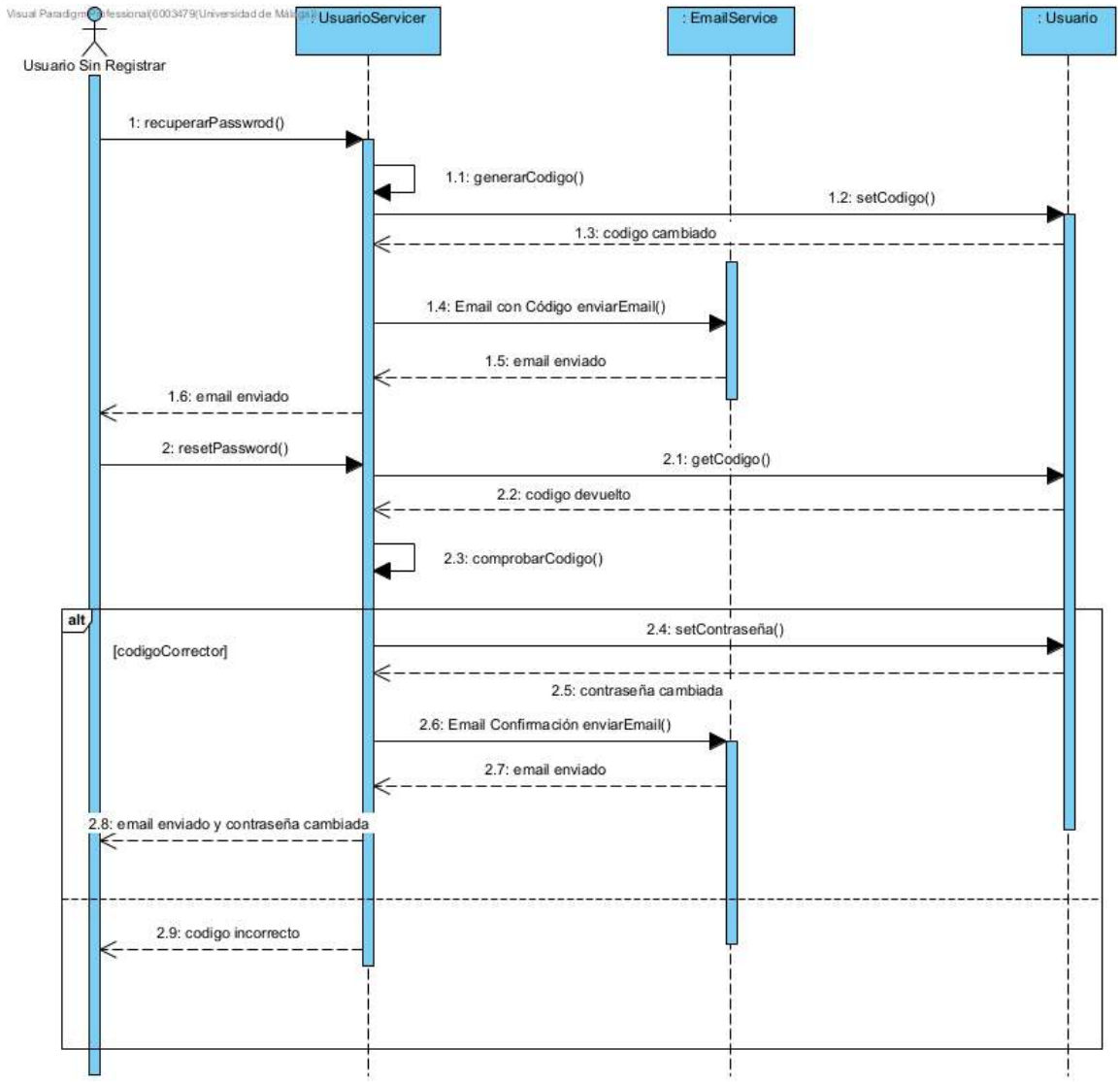


Figura 46: Diagrama de secuencia para la recuperación de contraseña.

Este diagrama describe el proceso completo de recuperación de contraseña para un usuario no registrado. Primero se genera un código de recuperación, que se asocia al usuario y se envía por correo electrónico. Posteriormente, el usuario introduce el código para validar la operación. Si este es correcto, se permite cambiar la contraseña, enviando un correo de confirmación. En caso contrario, se devuelve un mensaje de error. El uso de fragmentos alt diferencia ambos escenarios.

A.3.4. Registro de Usuario

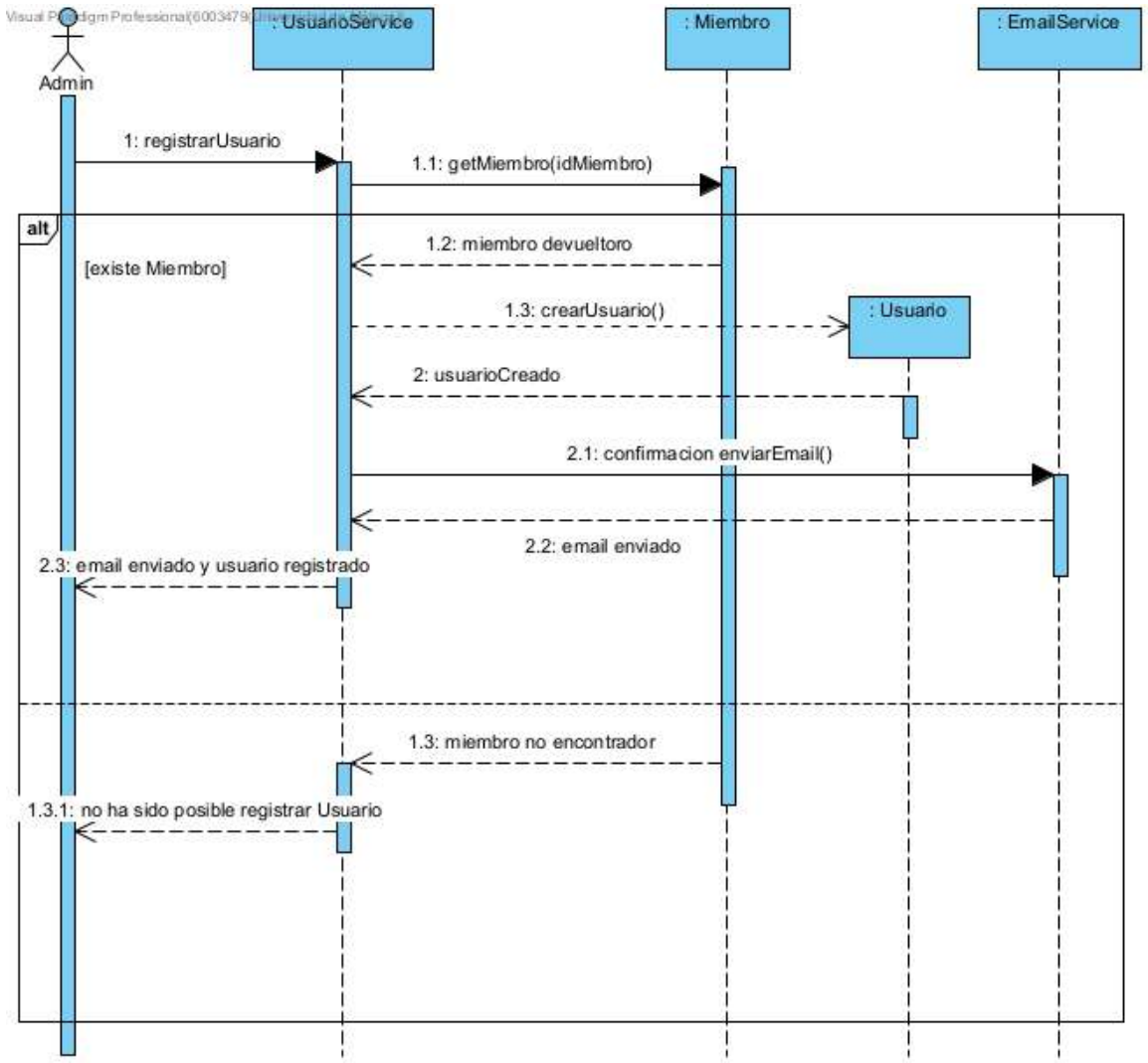


Figura 47: Diagrama de secuencia para el registro de usuario.

Por último, este diagrama refleja el proceso de registro de un nuevo usuario por parte de un administrador. El flujo verifica primero la existencia de un miembro en el sistema. Si existe, se crea el nuevo usuario y se envía un correo de confirmación. Si el miembro no existe, se interrumpe el proceso y se devuelve un error. El fragmento alt permite modelar claramente ambas posibilidades.

Apéndice B

Manual de Usuario

El presente manual tiene como objetivo guiar a los usuarios en el uso de la aplicación de gestión de cofradías y hermandades. Se proporciona una descripción práctica de las funcionalidades principales, con instrucciones paso a paso para que tanto administradores como usuarios registrados puedan realizar de forma sencilla las operaciones habituales: gestionar miembros, registrar pagos, asignar tallajes o consultar información relevante.

Este manual está diseñado para ser un documento de referencia rápida y accesible, de manera que cualquier usuario, incluso sin conocimientos técnicos avanzados, pueda desenvolverse correctamente en la aplicación. Asimismo, se incluyen capturas de pantalla representativas que facilitan la comprensión de cada apartado y permiten una navegación más intuitiva.

B.1. Inicio de Sesión

A continuación se presentan distintas capturas de la funcionalidad de **Inicio de sesión**. A partir de ellas se explica de manera detallada el procedimiento que debe seguir el usuario para acceder a la aplicación, así como las opciones disponibles en este proceso.

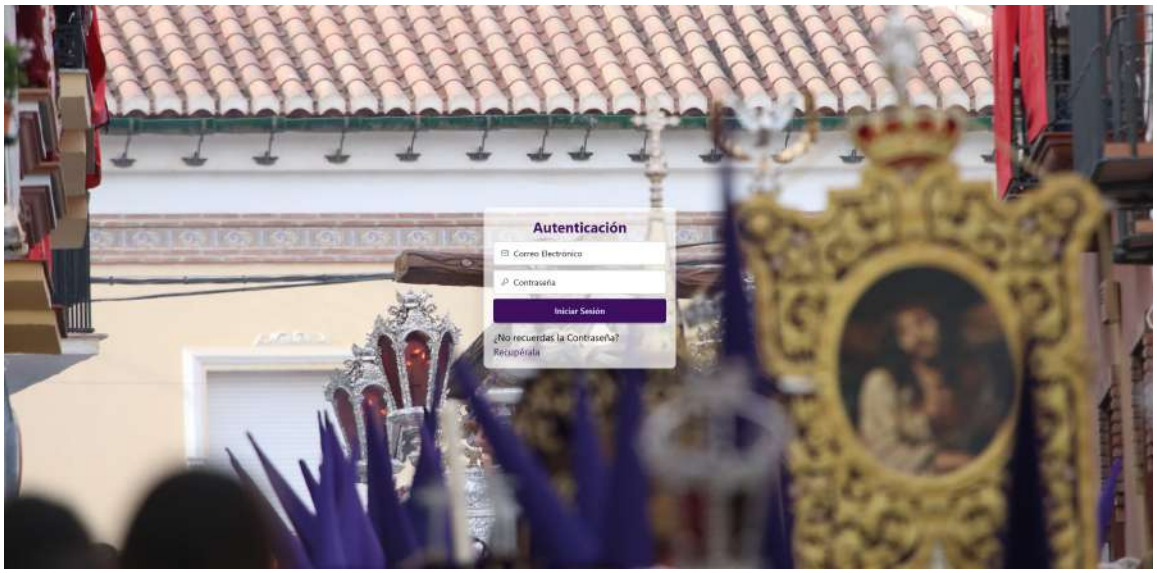


Figura 48: Captura de Pantalla del Inicio de Sesión.

En la Figura48 se ve como si se introducen los datos de correo y contraseña se podrá acceder a la aplicación, y de otra manera, si no se dispone de la contraseña correcta, se puede elegir recuperarla (Figura49).

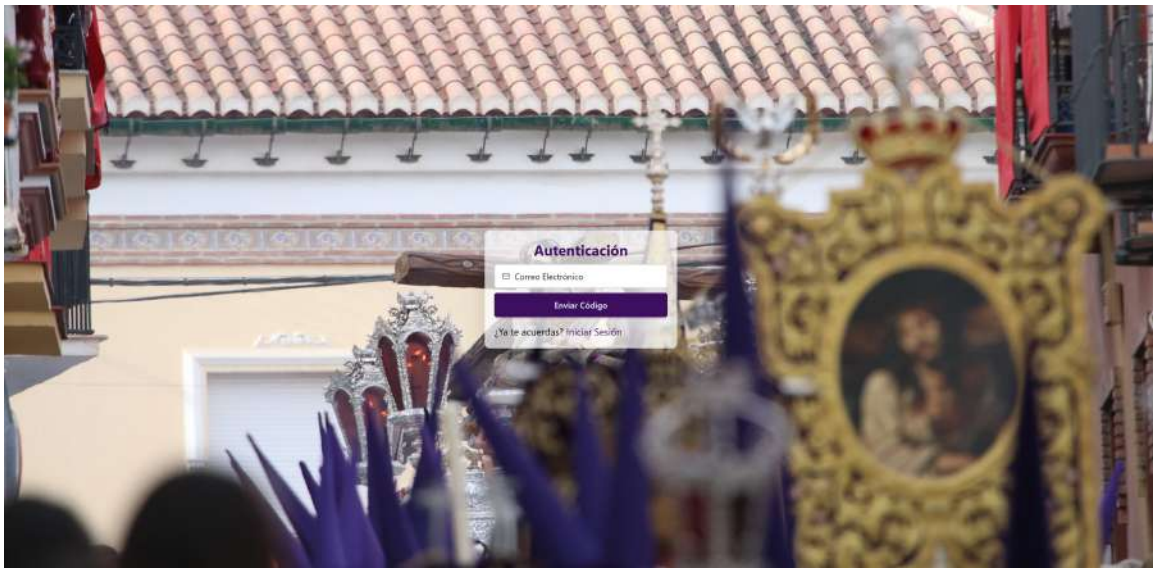


Figura 49: Captura de Pantalla de Recuperar Contraseña 1.

En esta pantalla se introducirá el correo electrónico del que se quiere recuperar la contraseña. Si existiese la cuenta con dicho correo, le llevará a la pantalla siguiente (Figura50).

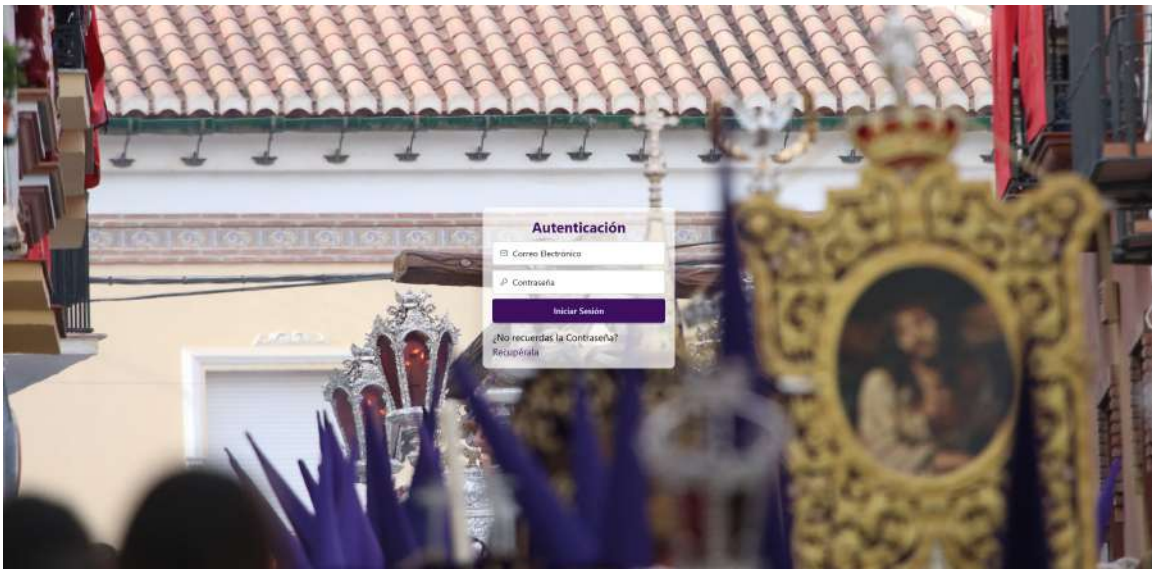


Figura 50: Captura de Pantalla de Recuperar Contraseña 2.

En esta pantalla se podrá indicar el correo electrónico del que se quiere recuperar la contraseña, el código enviado a dicho correo, y la nueva contraseña. Una vez finalizado nos llevará de nuevo al inicio de sesión (Figura48).

B.2. Perfil de Usuario Registrado

A continuación se presentan distintas capturas de la funcionalidad de **Perfil de Usuario**. A partir de ellas se explica de manera detallada cómo consultar la información personal registrada, así como las opciones para modificarla.

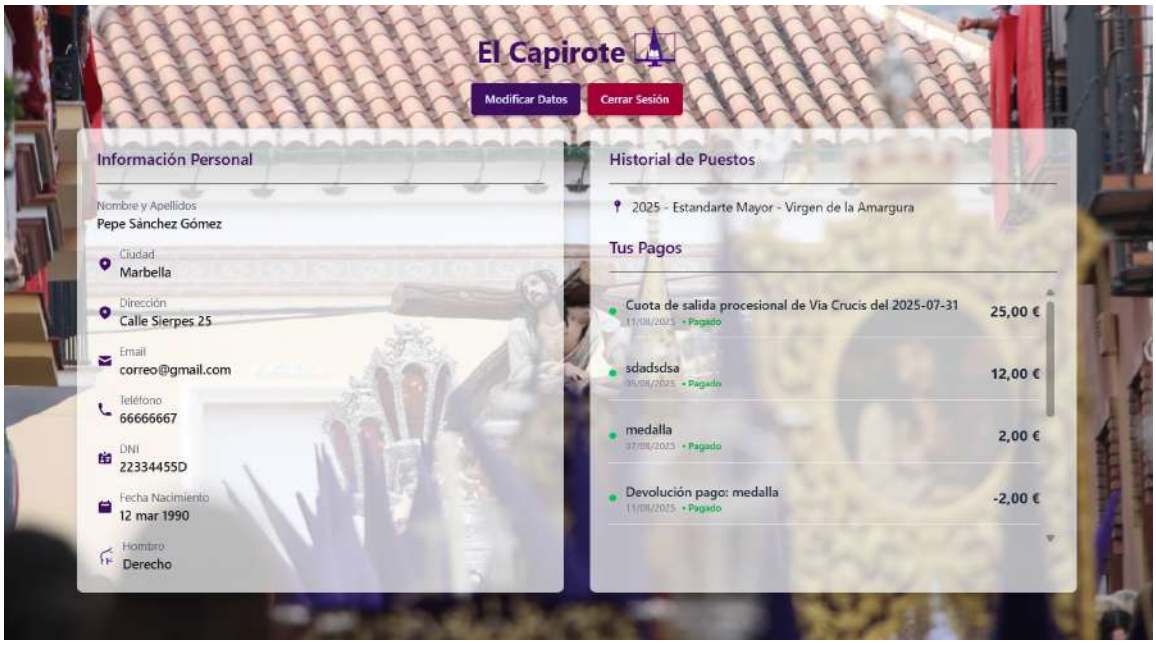


Figura 51: Captura de pantalla del Perfil de Usuario.

En la Figura 51 se muestra la vista principal del perfil de usuario. Aquí se puede consultar la información personal (nombre, apellidos, dirección, contacto, etc.), así como el historial de puestos y los pagos realizados o pendientes.

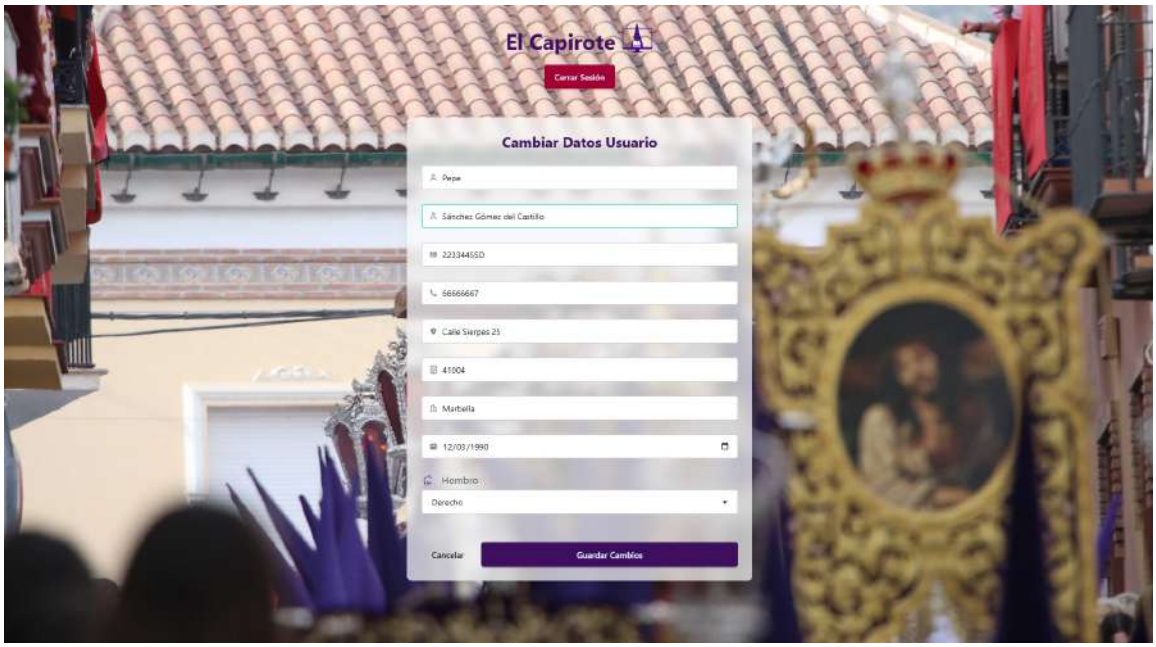


Figura 52: Captura de pantalla de la opción *Modificar Datos*.

En esta pantalla (Figura 52) el usuario puede actualizar los datos de su perfil, tales como

dirección, correo, teléfono o preferencia de hombro. Una vez completadas las modificaciones, deberá pulsar en *Guardar Cambios* para que el sistema los actualice.

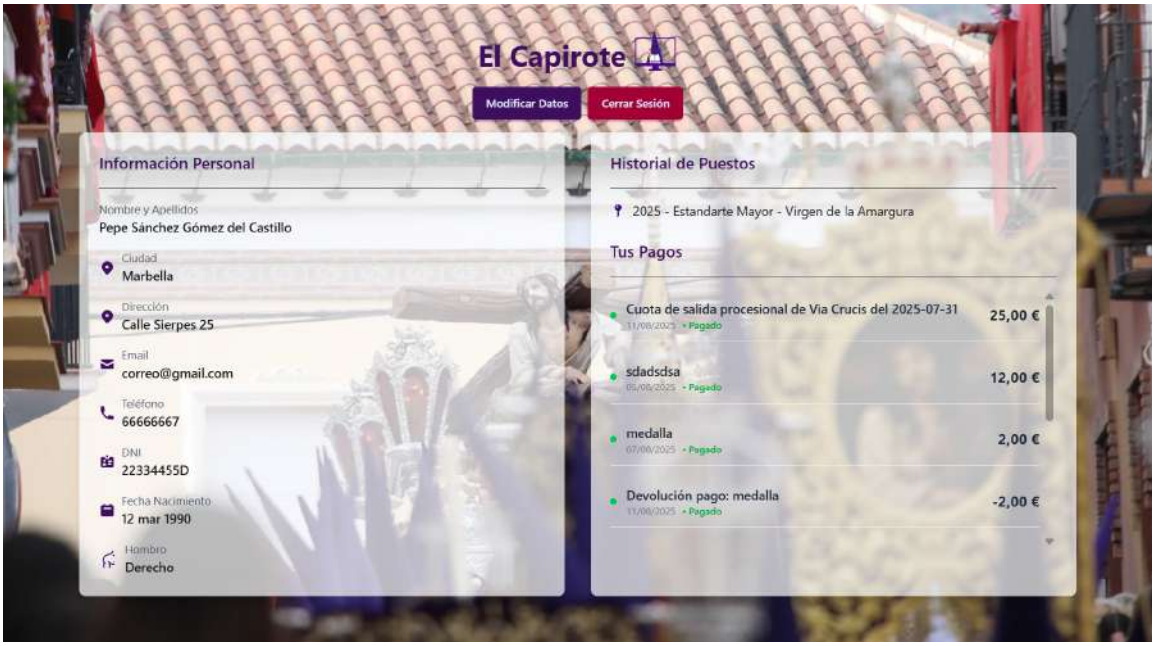


Figura 53: Captura de pantalla tras modificar los datos del usuario.

La Figura 53 refleja el resultado tras guardar los cambios. El sistema actualiza la información en la base de datos y redirige de nuevo al perfil del usuario, donde ya se muestran los datos corregidos.

B.3. Panel de Administración

El **panel de administración** constituye el núcleo de gestión de la aplicación, desde el cual el usuario con rol de administrador puede acceder a todas las funcionalidades principales: gestión de miembros, horquilleros, pagos, préstamos, listados, plantillas y configuración general del sistema.

En esta vista inicial se da la bienvenida al administrador y se ofrece una navegación clara y directa hacia cada módulo de trabajo. De esta forma, se centralizan en un único lugar las operaciones críticas para la cofradía, permitiendo un acceso ágil y organizado.

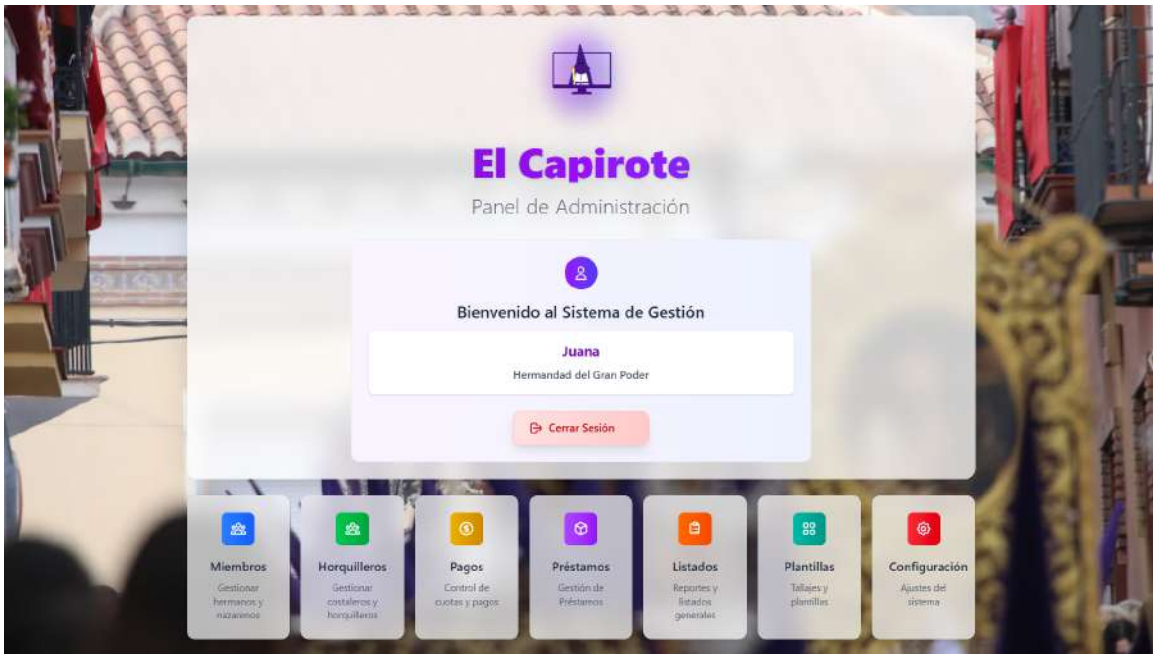


Figura 54: Vista general del panel de administración.

B.3.1. Gestión de Miembros

La funcionalidad de **Gestión de Miembros** permite al administrador crear, modificar, registrar, dar de baja y consultar información detallada de los hermanos y usuarios de la cofradía. A través de esta sección se centralizan todos los datos personales, las salidas procesionales y el estado de pagos de cada miembro, facilitando un control completo y actualizado.

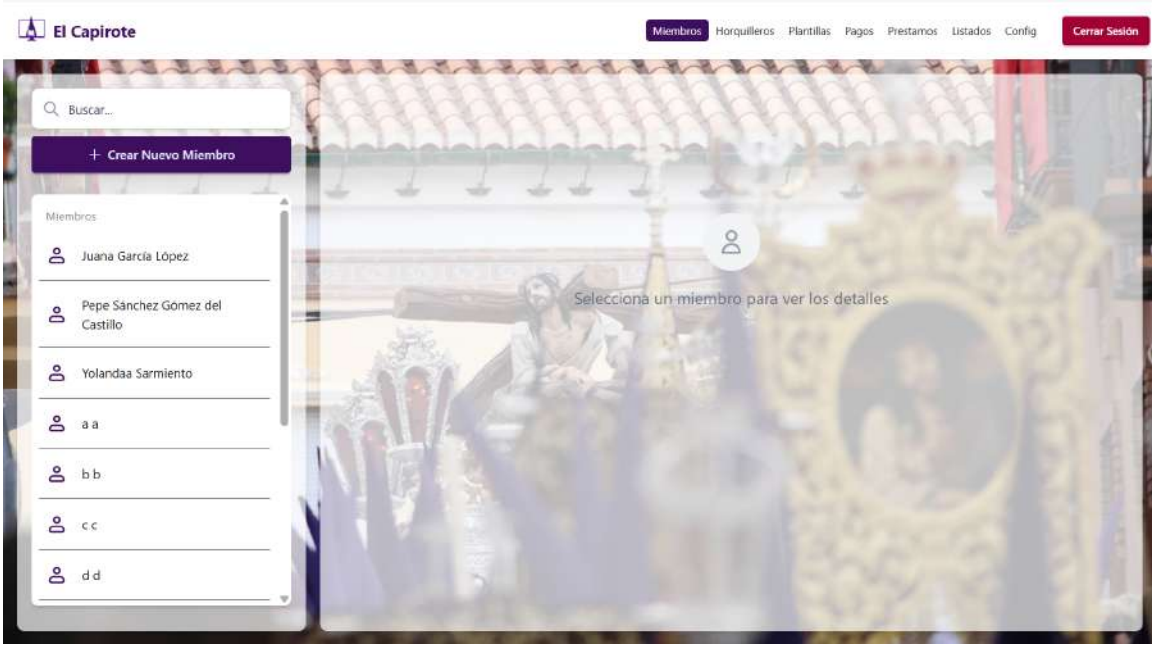


Figura 55: Pantalla principal de Miembros, con listado y opción de búsqueda.

En la Figura 55 se muestra la pantalla inicial de la sección, donde es posible visualizar el listado de miembros, buscar por nombre y acceder a los detalles de cada uno.

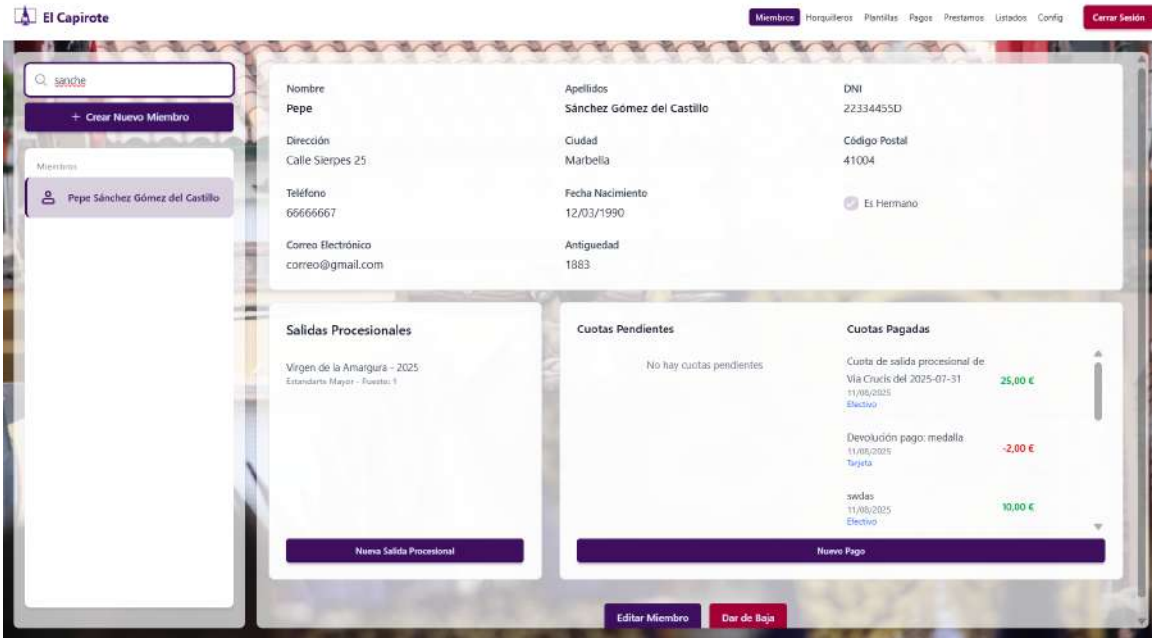


Figura 56: Detalle de un miembro con información personal, cuotas y salidas.

La Figura 56 presenta la ficha individual de un miembro, donde se visualizan sus datos

personales, las salidas procesionales en las que participa y el estado de sus cuotas (pagadas o pendientes). Desde esta pantalla es posible registrar un nuevo pago o asignar una salida procesional.

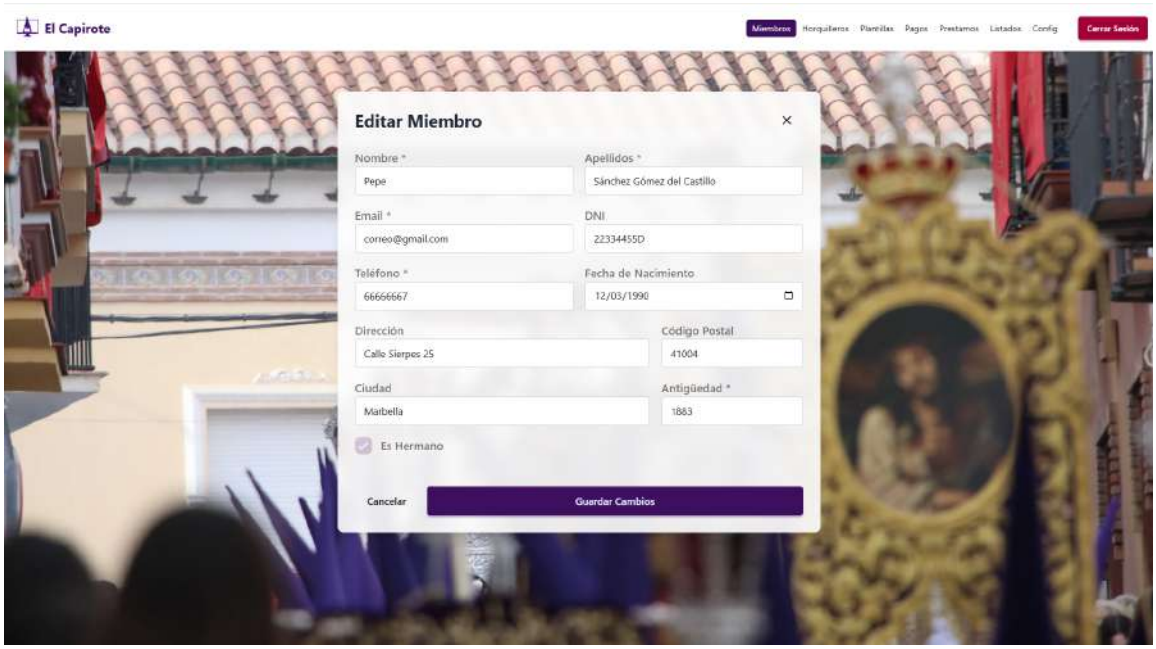


Figura 57: Formulario de edición de datos de un miembro.

Si se selecciona la opción *Editar Miembro*, se despliega el formulario mostrado en la Figura 57, donde el administrador puede actualizar cualquier dato del miembro, como dirección, correo electrónico o antigüedad.

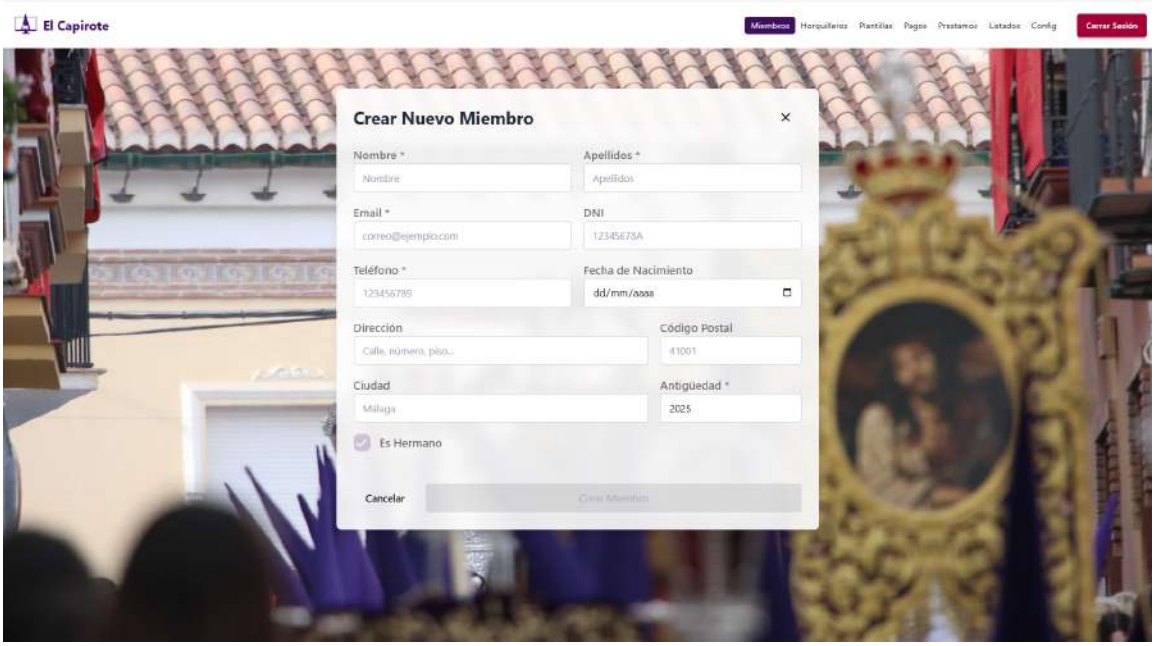


Figura 58: Formulario para la creación de un nuevo miembro.

Para dar de alta a un nuevo integrante de la hermandad, se utiliza la pantalla de la Figura 58, que permite introducir todos los datos necesarios y almacenarlos en el sistema.

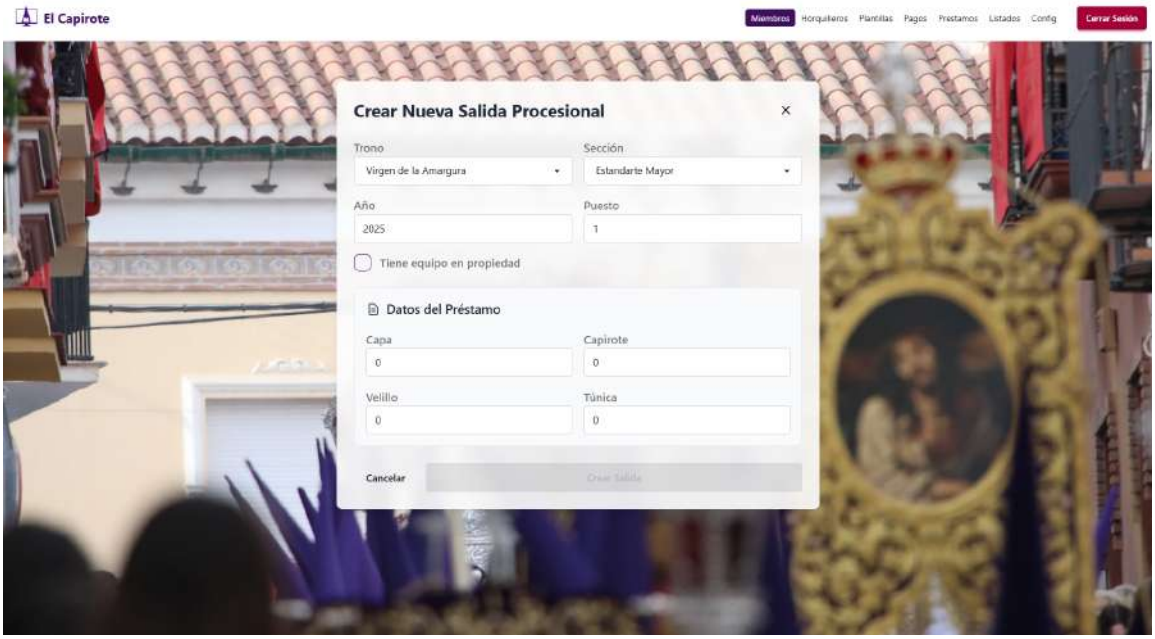


Figura 59: Asignación de una salida procesional a un miembro.

La Figura 59 muestra el formulario para crear una nueva salida procesional, donde se selecciona trono, sección, año y puesto. También se pueden indicar los elementos de vestimenta

prestados por la cofradía.

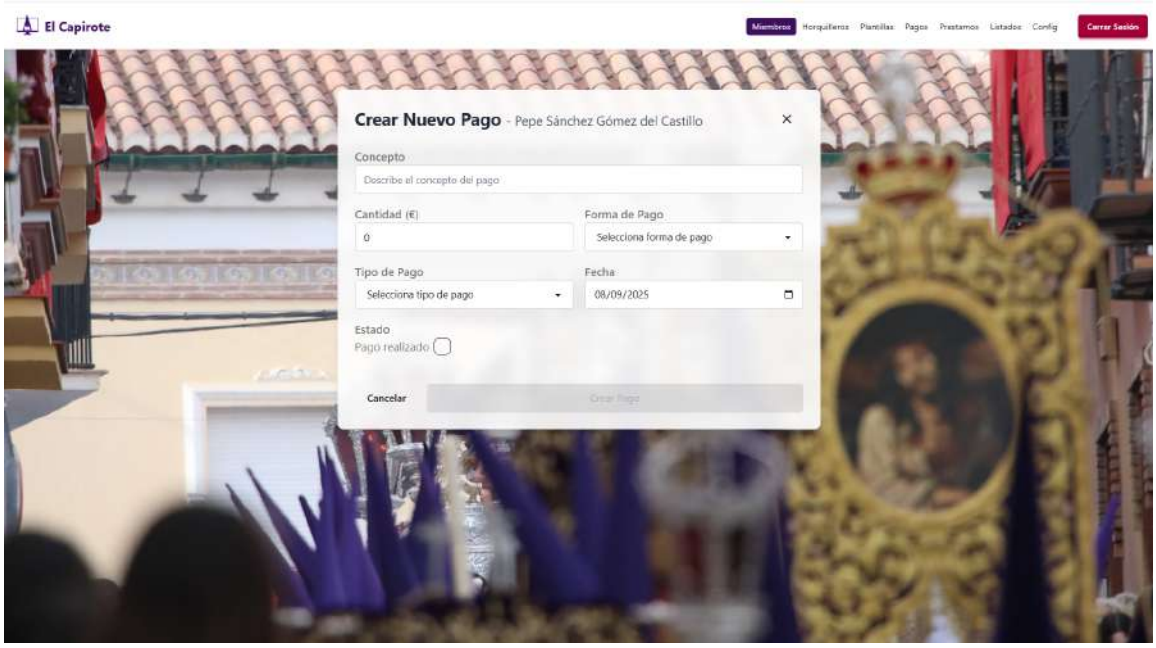


Figura 60: Creación de un nuevo pago asociado a un miembro.

En la Figura 60 se observa la pantalla para registrar un pago, donde se define el concepto, la cantidad, el tipo y la forma de pago, así como la fecha.

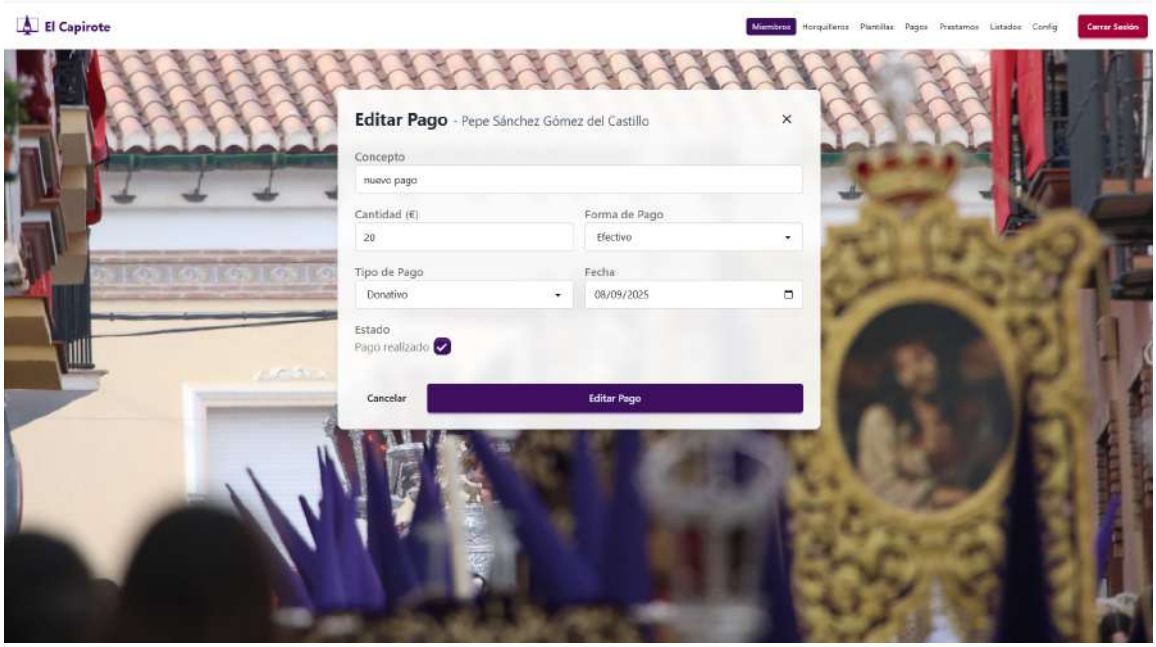


Figura 61: Edición de un pago existente.

Los pagos previamente creados pueden editarse, como se aprecia en la Figura 61, permitiendo modificar su importe, concepto o estado.

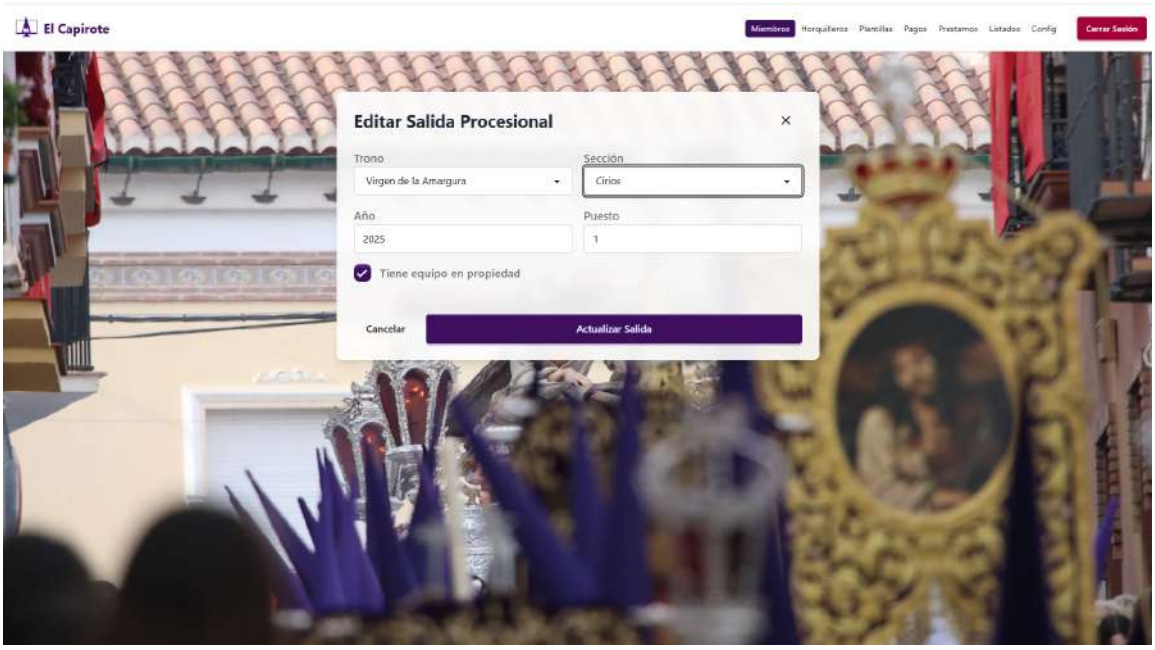


Figura 62: Edición de una salida procesional existente.

De igual forma, la Figura 62 refleja cómo un administrador puede actualizar la información de una salida procesional ya registrada.

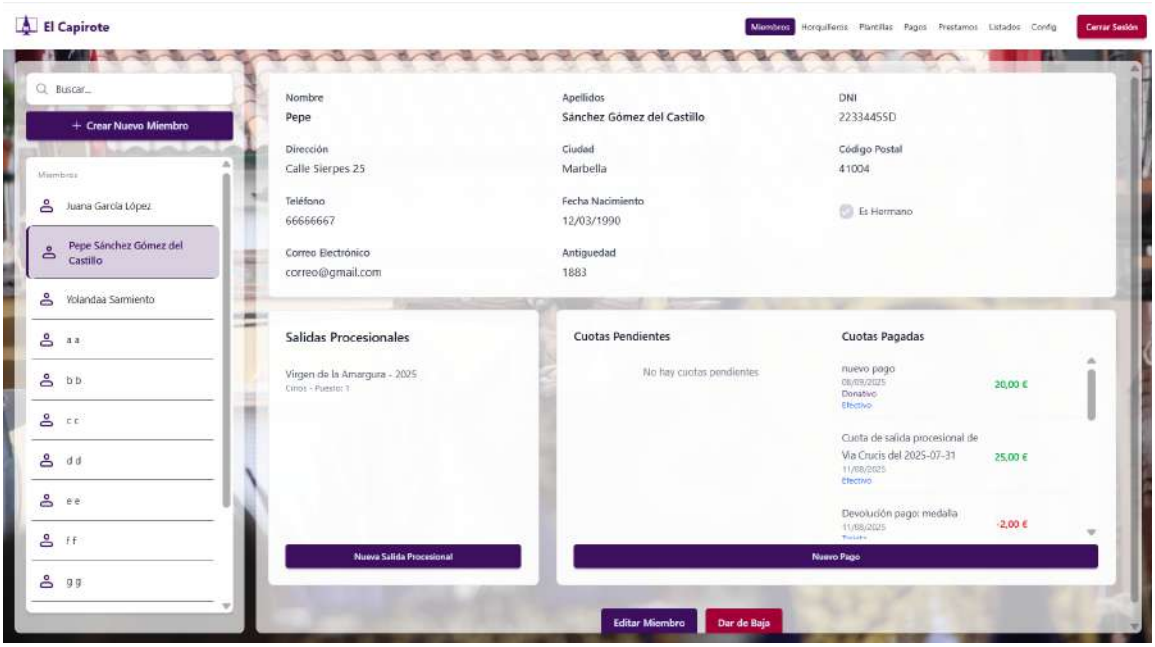


Figura 63: Resultado de Salida Procesional y Pago Editados.

Tras las ediciones del pago y la salida procesional, en la Figura 63 se puede observar como se han guardado los cambios correctamente.

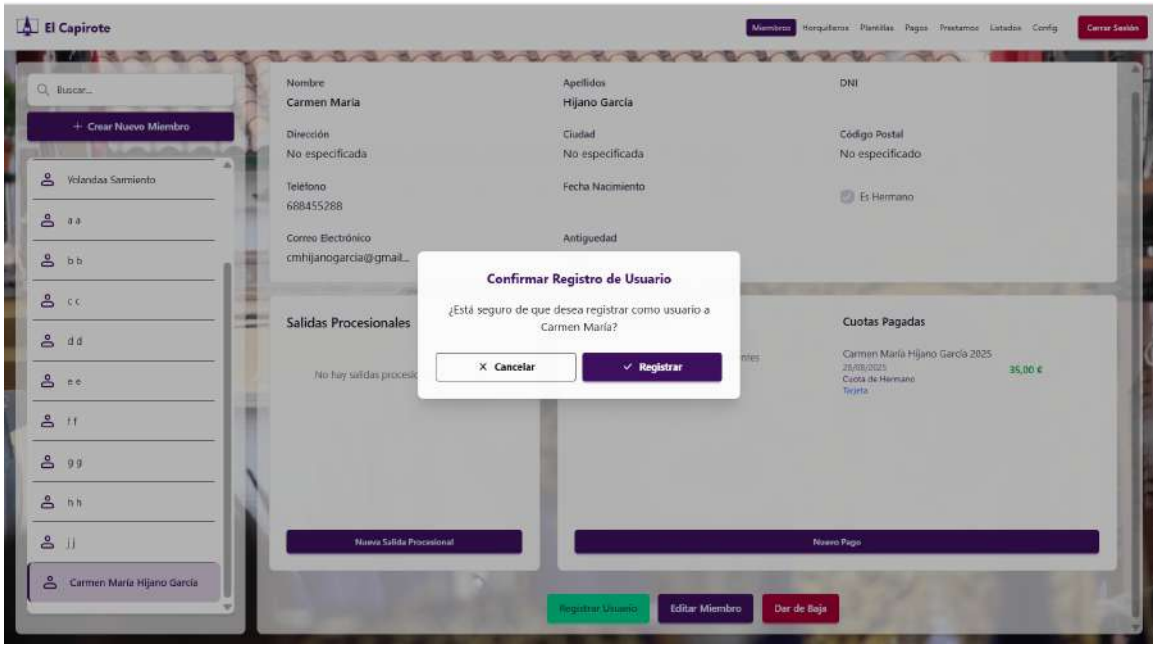


Figura 64: Confirmación para registrar a un miembro como usuario de la aplicación.

En caso de querer dar acceso a la aplicación a un miembro registrado, se utiliza la opción *Registrar Usuario*, cuya confirmación se aprecia en la Figura 64. Dicha opción estará solo permitida para los miembros que no sean ya usuarios, de tal forma que se les pueda dar de alta.

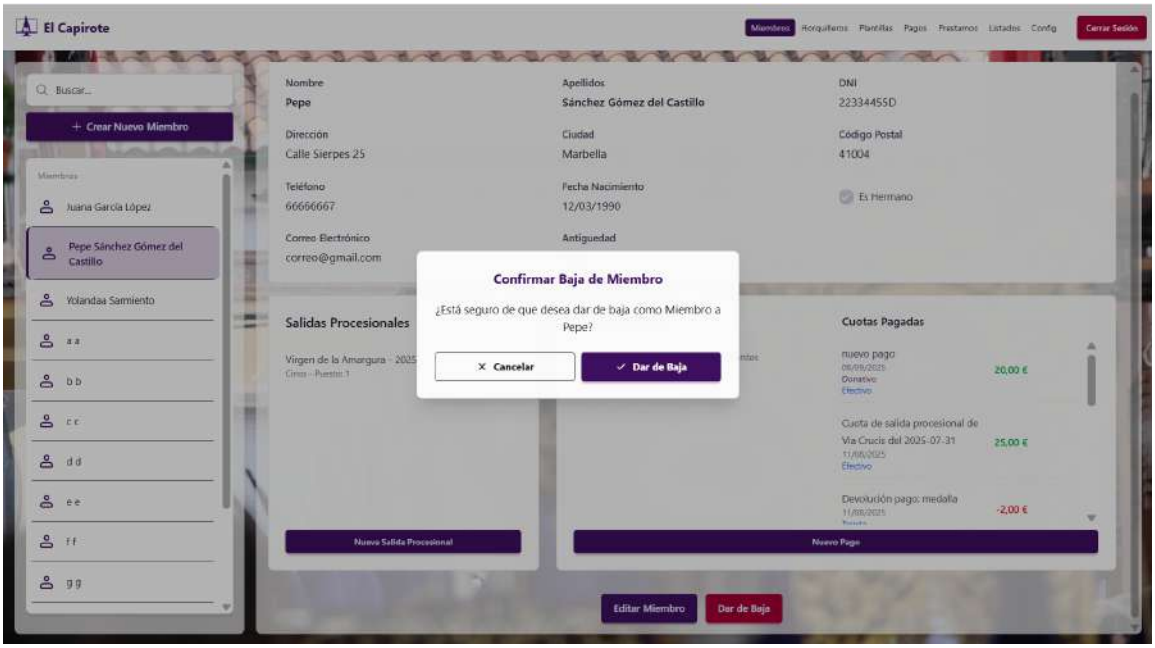


Figura 65: Confirmación de la baja de un miembro.

Por último, si un miembro debe ser dado de baja, se despliega el cuadro de confirmación de la Figura 65, lo que asegura que esta acción no se ejecute de manera accidental.

B.3.2. Gestión de Horquilleros

En este apartado se presenta la funcionalidad de **Horquilleros**. A diferencia del módulo de Miembros, en esta sección se sustituyen las salidas procesionales por los **tallajes**, manteniendo además campos específicos como el hombro de carga y observaciones adicionales.

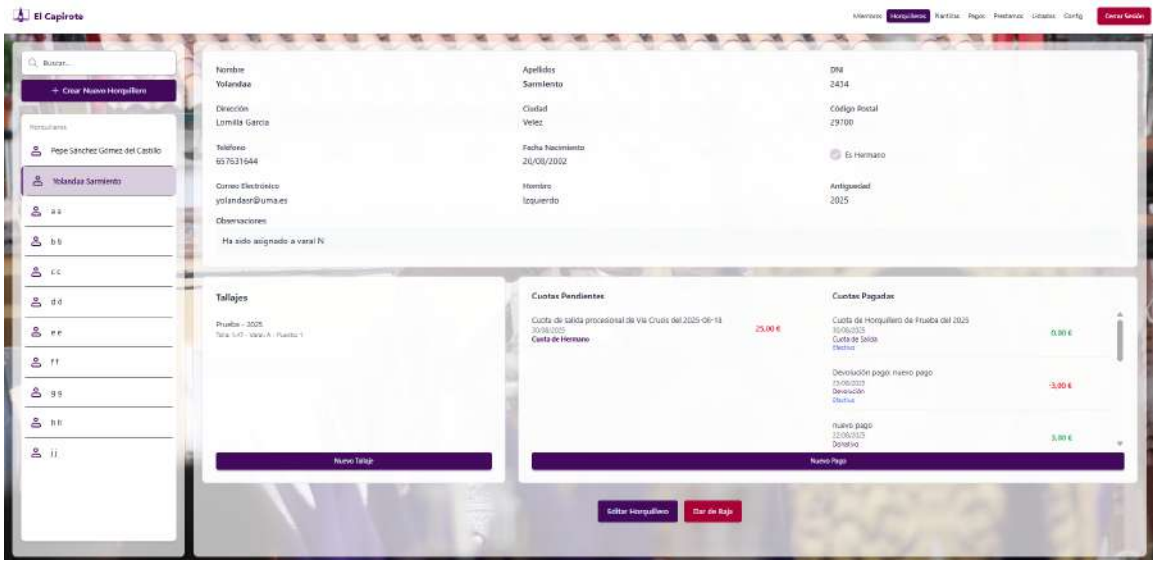


Figura 66: Listado de Horquilleros y ficha detallada.

En la Figura 66 se muestra el listado de horquilleros registrados, así como la ficha detallada de uno de ellos. En esta vista se observa la información personal, los **tallajes**, las cuotas pendientes y las cuotas pagadas, junto con las opciones de editar datos, dar de baja o registrar pagos.

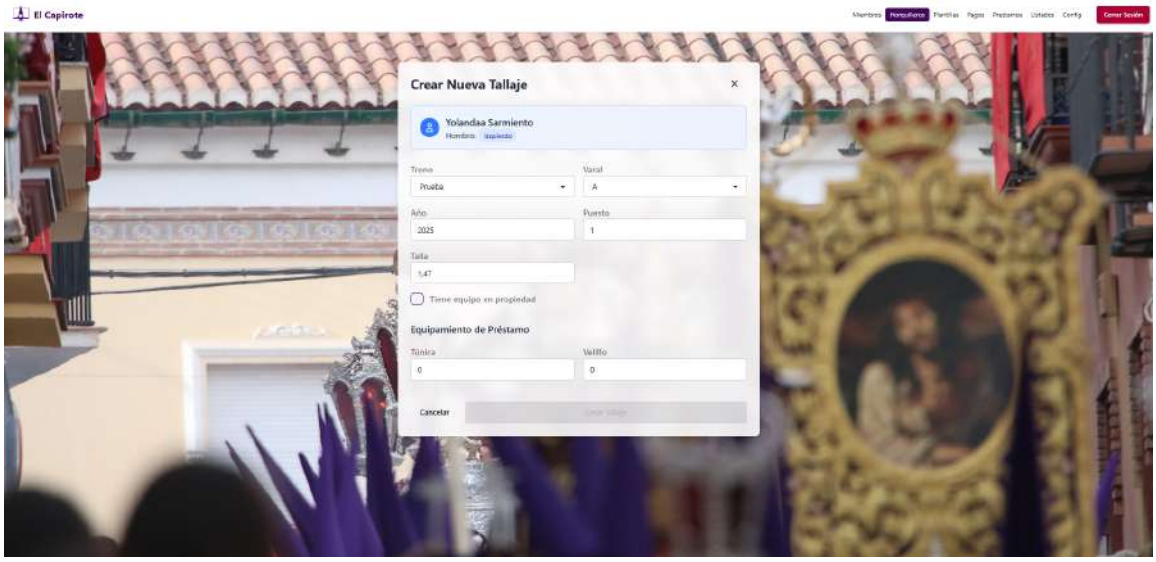


Figura 67: Creación de un nuevo tallaje.

En la Figura 67 se aprecia cómo crear un nuevo tallaje, indicando trono, varal, año, puesto y talla. Adicionalmente, se permite registrar **equipamiento en préstamo**, limitado en este

caso a túnica y velillo.

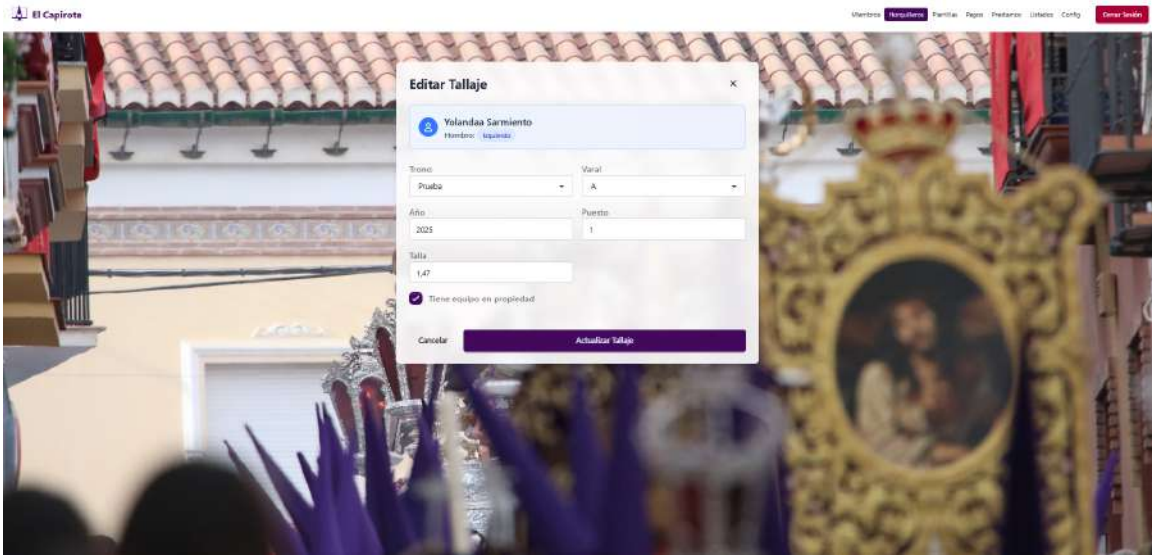


Figura 68: Edición de un tallaje existente.

En la Figura 68 se observa el proceso de edición de un tallaje ya creado, en el cual es posible modificar los campos de talla, varal, puesto y trono, así como la información de equipamiento.

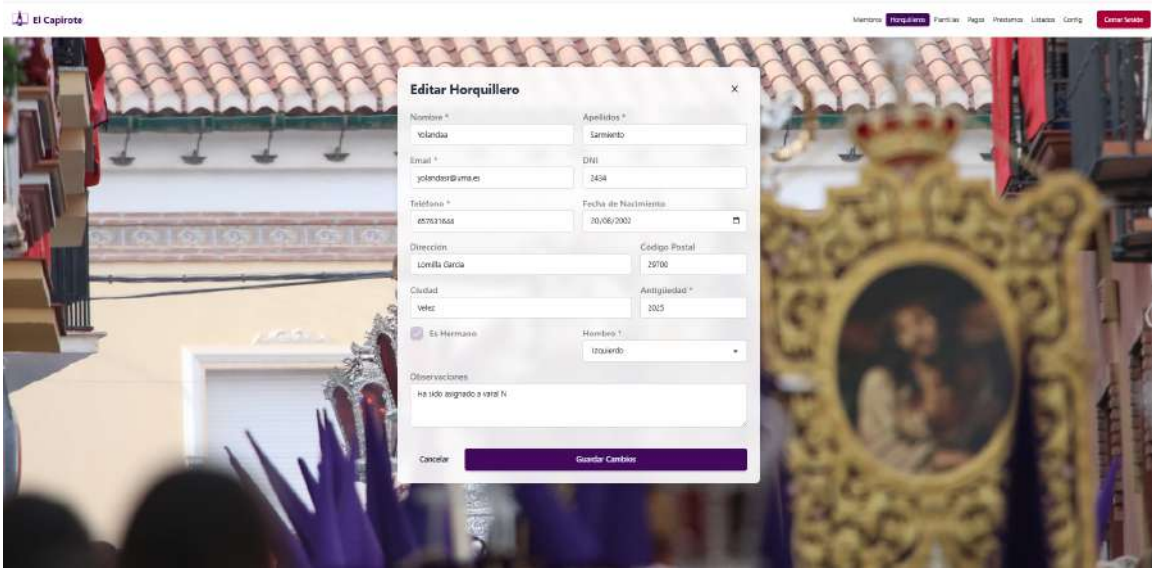


Figura 69: Edición de datos de un horquillero.

La Figura 69 muestra la edición de la ficha de un horquillero, donde se añaden los campos específicos de **hombro** y **observaciones**, además de los datos básicos compartidos con el módulo de miembros.

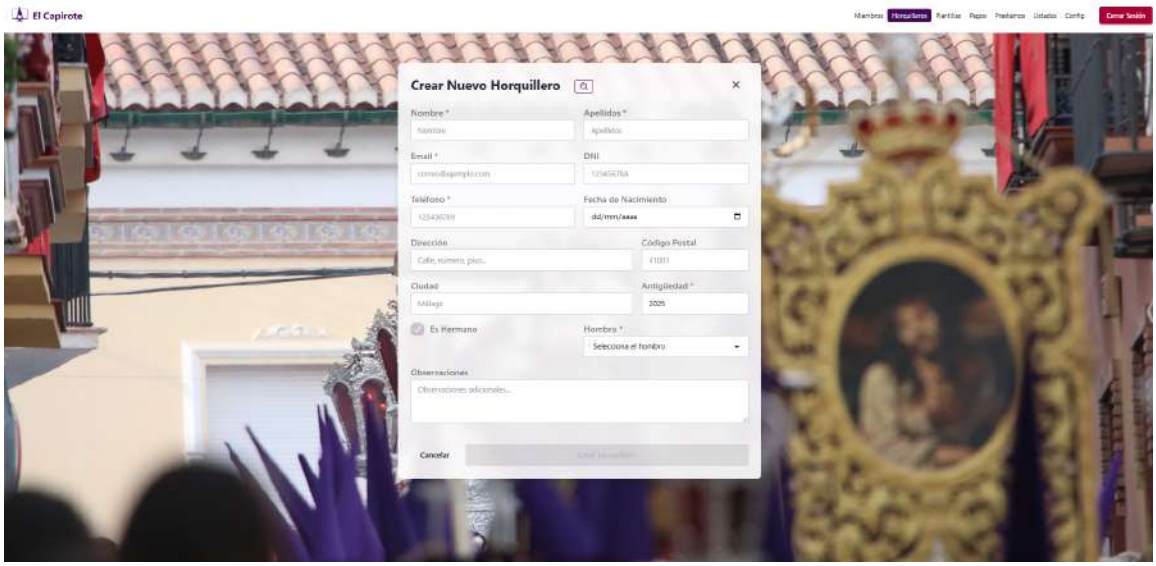


Figura 70: Creación de un nuevo horquillero.

En la Figura 70 se muestra el formulario para crear un nuevo horquillero. Este proceso permite, mediante el icono de lupa, **buscar un miembro previamente registrado** y recuperar sus datos automáticamente, facilitando la conversión de un miembro en horquillero.

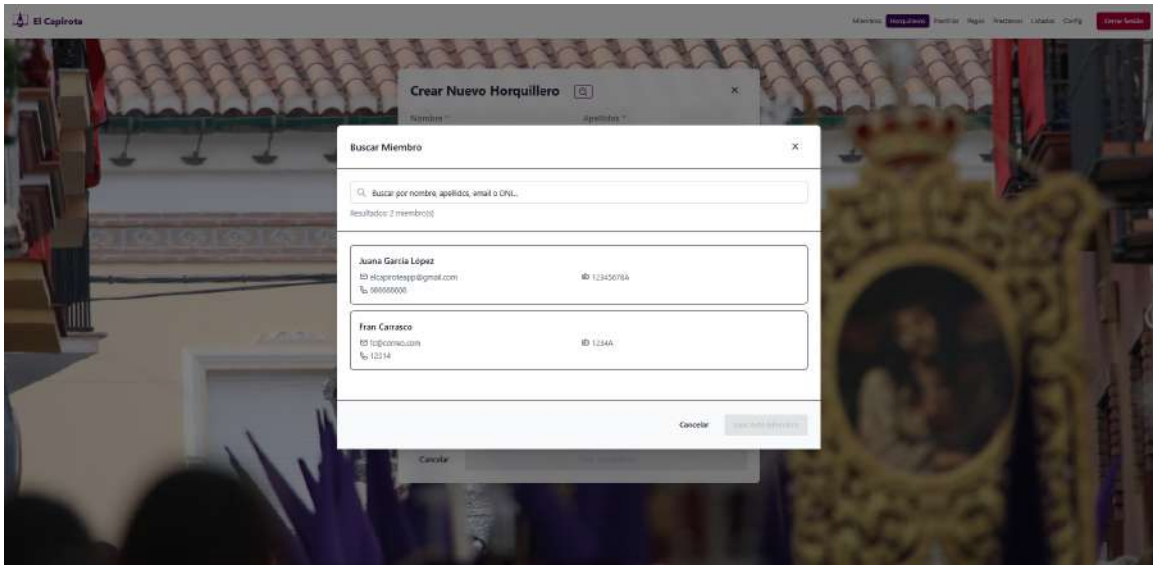


Figura 71: Selección de miembro existente para convertir en horquillero.

En la Figura 71 se visualiza el listado de miembros disponibles para convertir en horquillero. Una vez seleccionado, se accede a la pantalla siguiente donde se pueden editar sus datos (Figura 70).

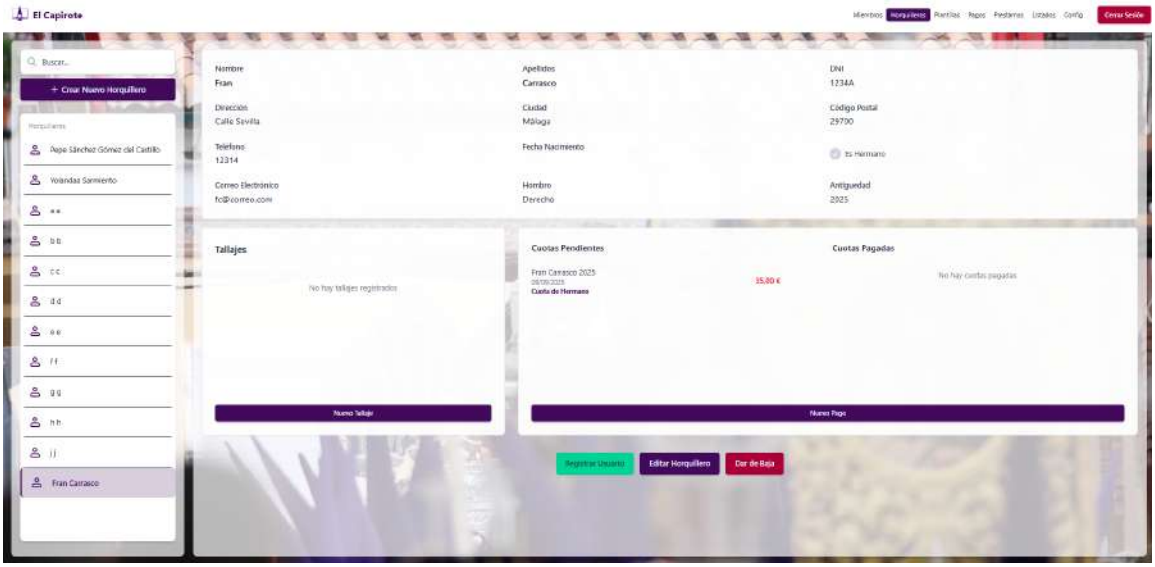


Figura 72: Ficha de horquillero tras recuperar los datos de un miembro.

Por último, la Figura 72 muestra la ficha final del horquillero tras haber recuperado la información de un miembro existente, completando el proceso de creación.

B.3.3. Plantillas

La sección de plantillas permite visualizar y organizar la distribución de horquilleros en los distintos tronos y secciones de la cofradía. Se muestran los **tallajes registrados en el año en curso**, así como la **ocupación actual** de cada sección o trono.

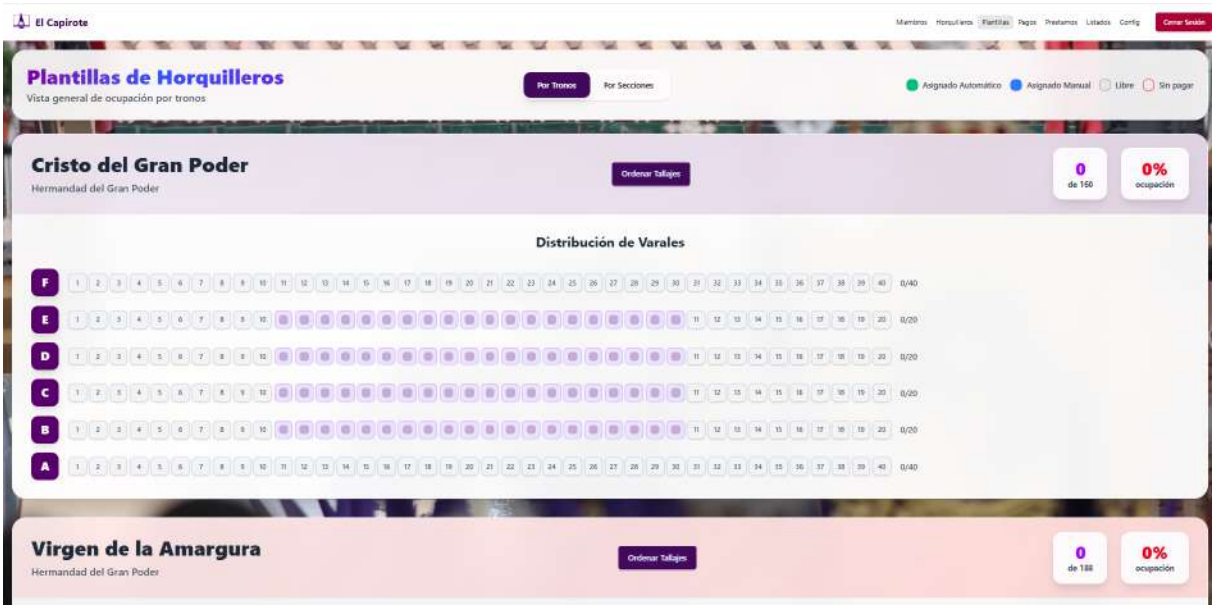


Figura 73: Vista general de los tronos con su distribución de varales y ocupación.

Dentro de cada trono, la aplicación representa la **distribución de varales** mediante un esquema visual, indicando los puestos ocupados y disponibles. Desde esta vista es posible ejecutar la opción de **ordenar tallajes**, lo cual abre un cuadro de confirmación (Figura 74) antes de aplicar el algoritmo de reorganización automática.

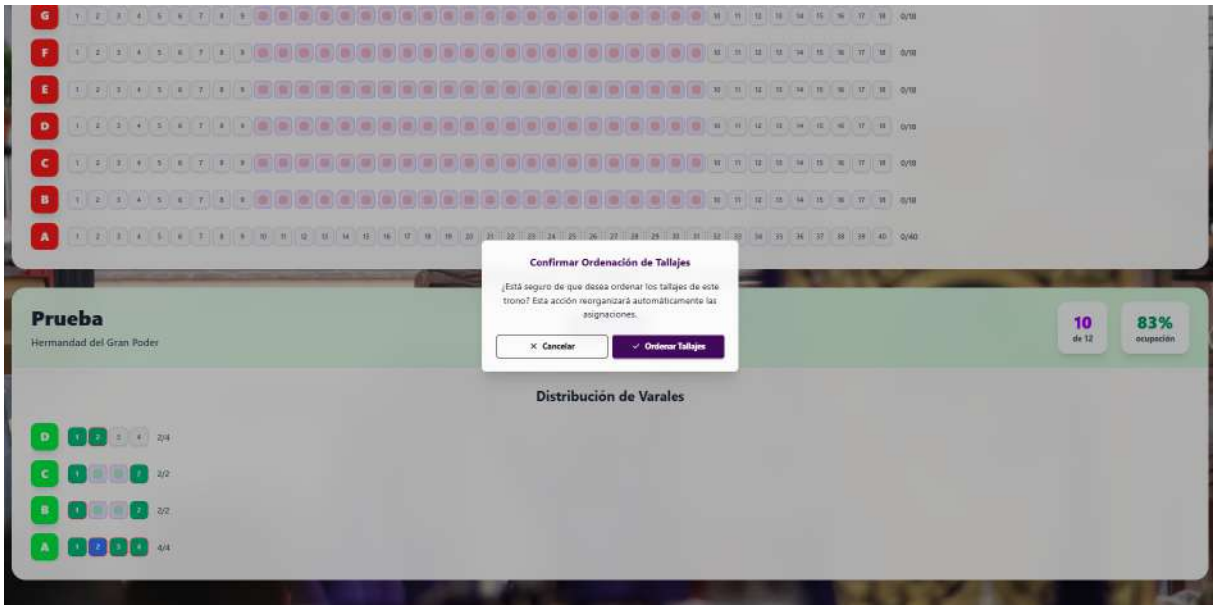


Figura 74: Modal de confirmación para la ordenación automática de tallajes.

Además de la vista por tronos, el sistema ofrece la posibilidad de consultar las **plantillas por secciones**. En este caso, la interfaz muestra cada sección de la procesión junto con la ocupación de sus filas y puestos, diferenciando entre asignaciones automáticas, manuales, libres o sin pagar.

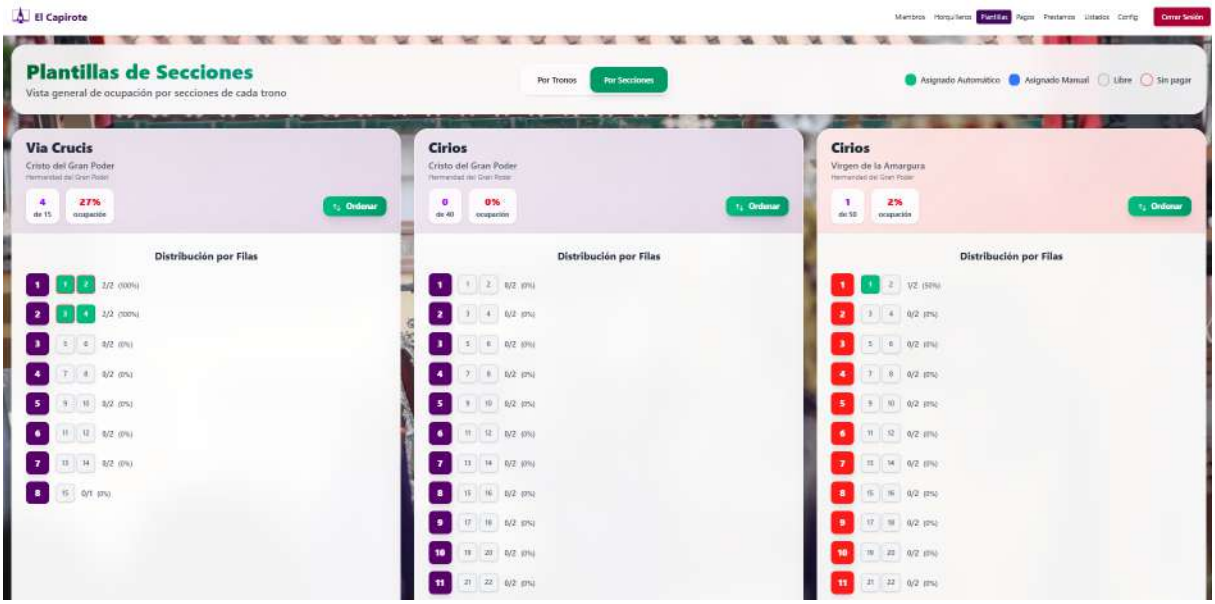


Figura 75: Distribución de horquilleros por secciones en la procesión.

B.3.4. Gestión de Pagos

La sección de **Pagos** permite al administrador llevar un control detallado de todas las cuotas, donativos y aportaciones de los miembros y horquilleros. La interfaz principal ofrece un **resumen diario** que muestra el número total de pagos, los pagos ya abonados y los que se encuentran pendientes, así como el balance económico correspondiente. Además, se presenta un listado con todos los pagos realizados o pendientes, ordenados cronológicamente.

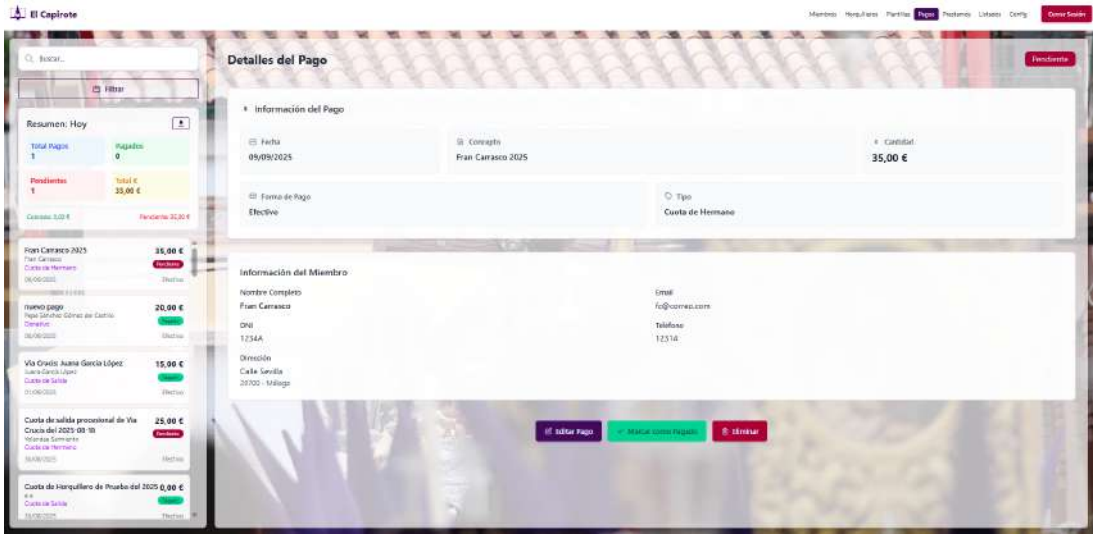


Figura 76: Interfaz principal de la sección de Pagos con el resumen diario y listado de operaciones.

El sistema permite aplicar filtros por rango de fechas para visualizar únicamente los pagos comprendidos en un período concreto. Una vez aplicado el filtro, tanto el listado de pagos como el resumen económico se actualizan automáticamente, mostrando únicamente la información correspondiente al rango establecido.

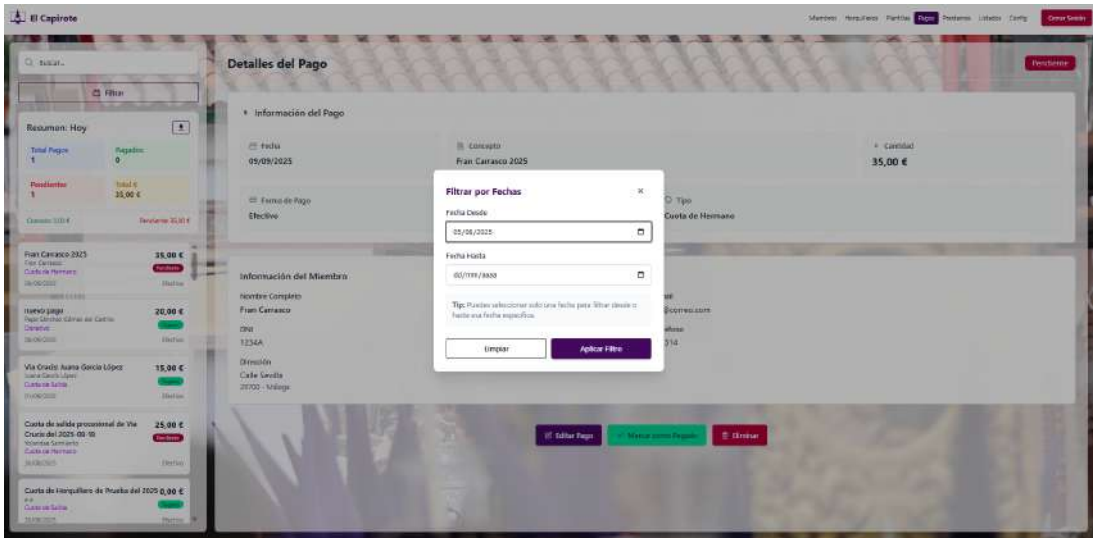


Figura 77: Filtro de pagos por fechas y actualización automática del resumen.

Cada pago puede ser gestionado individualmente desde su ficha detallada, donde se muestran los datos principales: concepto, tipo, forma de pago, cantidad y estado. Desde esta vista,

el administrador dispone de las siguientes acciones:

- **Editar pago**: permite modificar los datos del registro (concepto, importe, forma de pago, etc.).

- **Marcar como pagado**: cambia el estado del pago a *Pagado*, quedando reflejado en el listado y en el resumen.

- **Eliminar pago**: elimina el registro del sistema.

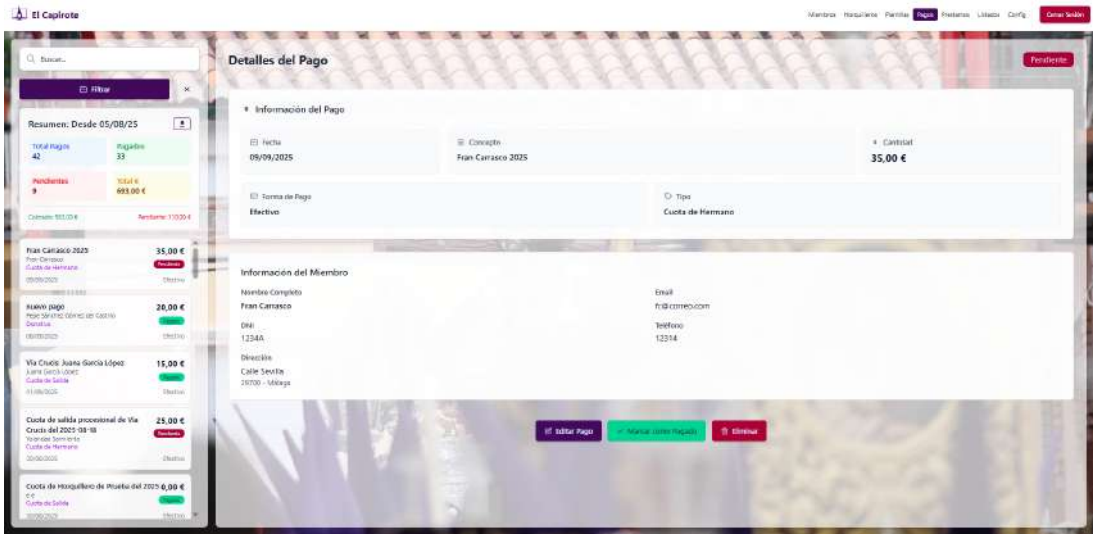


Figura 78: Detalle de un pago con opciones de edición, marcado y eliminación.

Al marcar un pago como pagado, el sistema solicita confirmación mediante un cuadro de diálogo. Una vez aceptada, el pago queda registrado como *Pagado* y se actualiza automáticamente en el listado y en el resumen general.

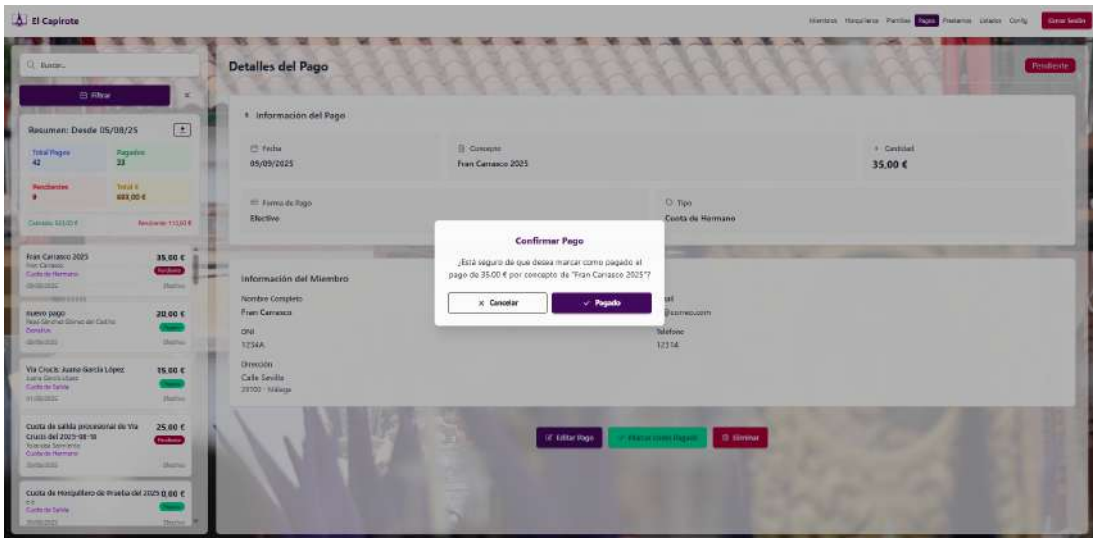


Figura 79: Confirmación de marcado de un pago como realizado.

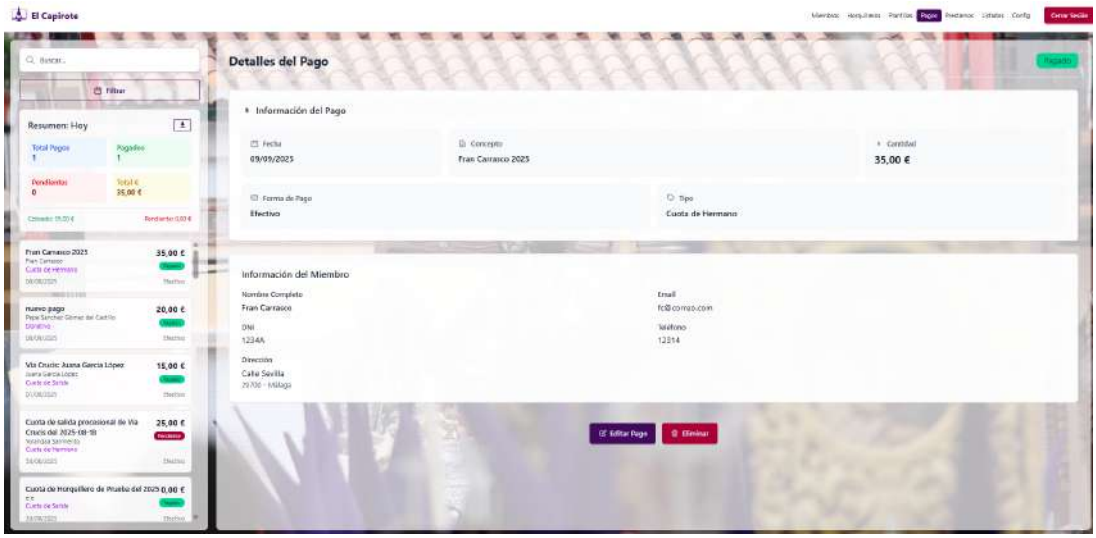


Figura 80: Visualización de un pago marcado como pagado.

Además, desde el cuadro de resumen existe la opción de **descargar** un listado en formato PDF con los pagos que aparecen en pantalla, ya sea el resumen diario o el filtrado por fechas. El documento generado incluye la información detallada de cada pago, agrupada por tipo y forma de pago, junto con los subtotales y el total general.

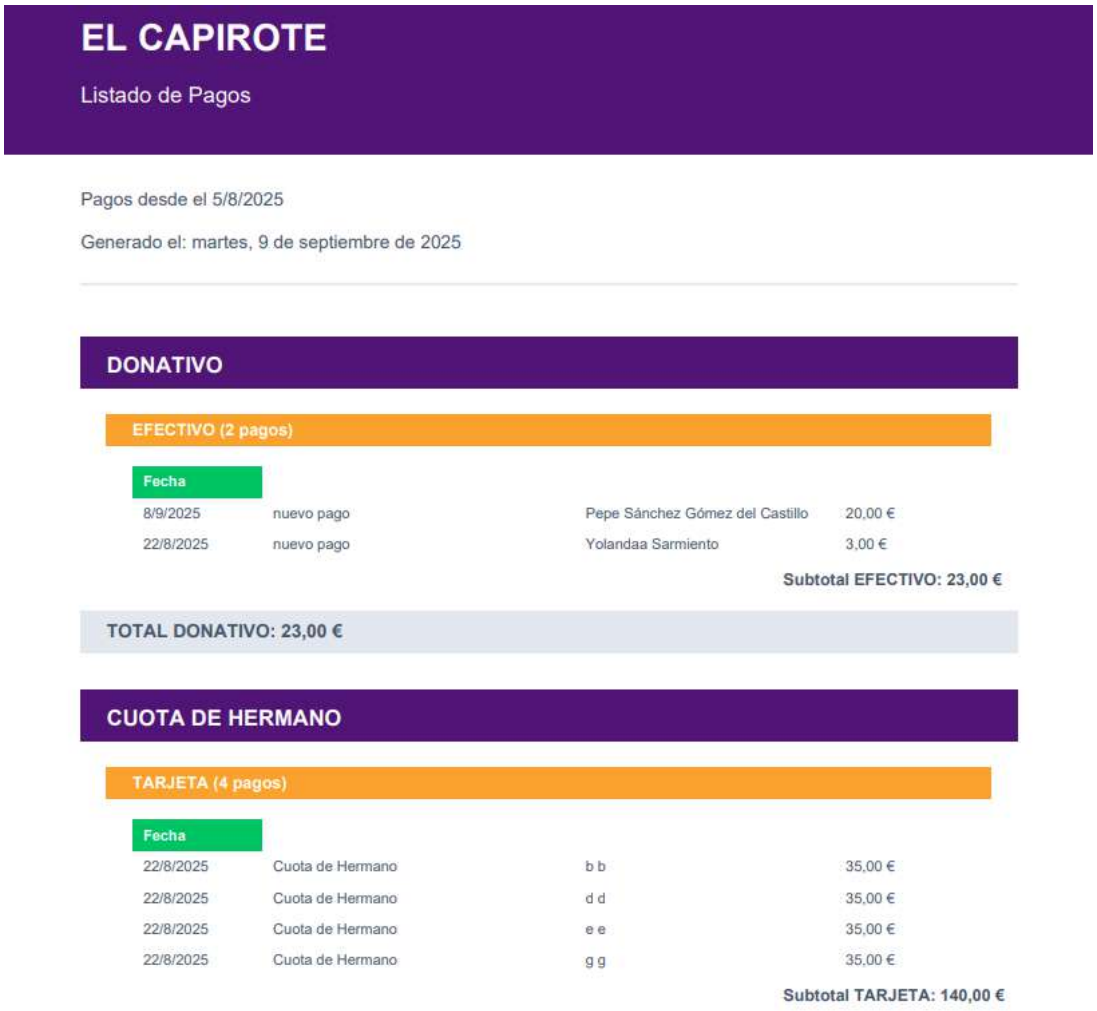


Figura 81: Ejemplo de parte de un PDF generado con el listado de pagos agrupados por tipo y forma de pago.

B.3.5. Gestión de Préstamos

En el módulo de **Préstamos** se gestionan los equipos entregados a los miembros. La interfaz principal muestra todos los préstamos registrados, con la posibilidad de **filtrar por fechas** o seleccionar únicamente los **préstamos activos**.

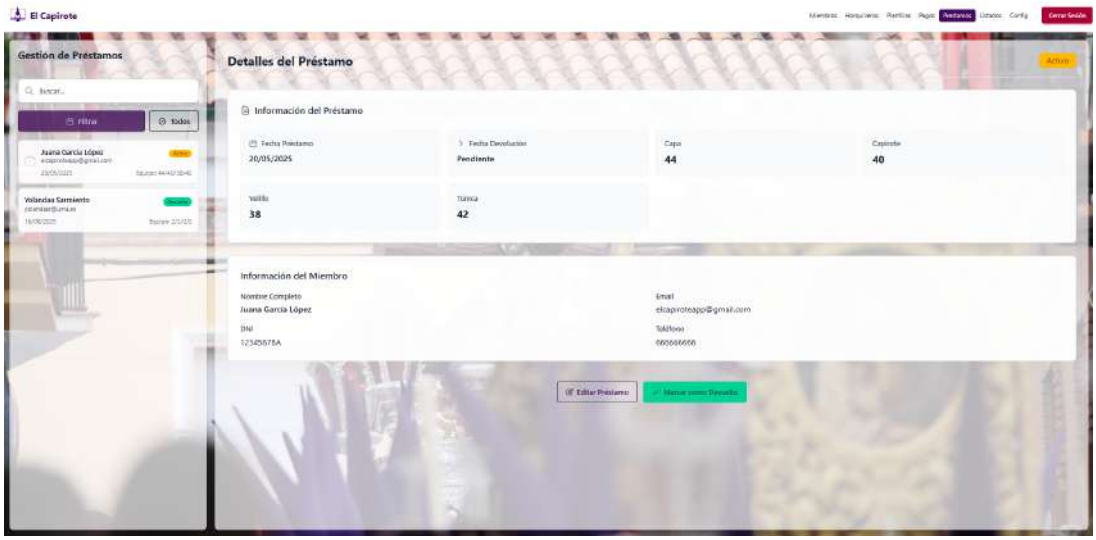


Figura 82: Vista general de préstamos con listado y detalles del seleccionado.

Cuando se visualizan solo los activos, aparece la opción de **Devolución Masiva**, que permite marcar todos los préstamos como devueltos con un solo clic, tras la confirmación correspondiente.

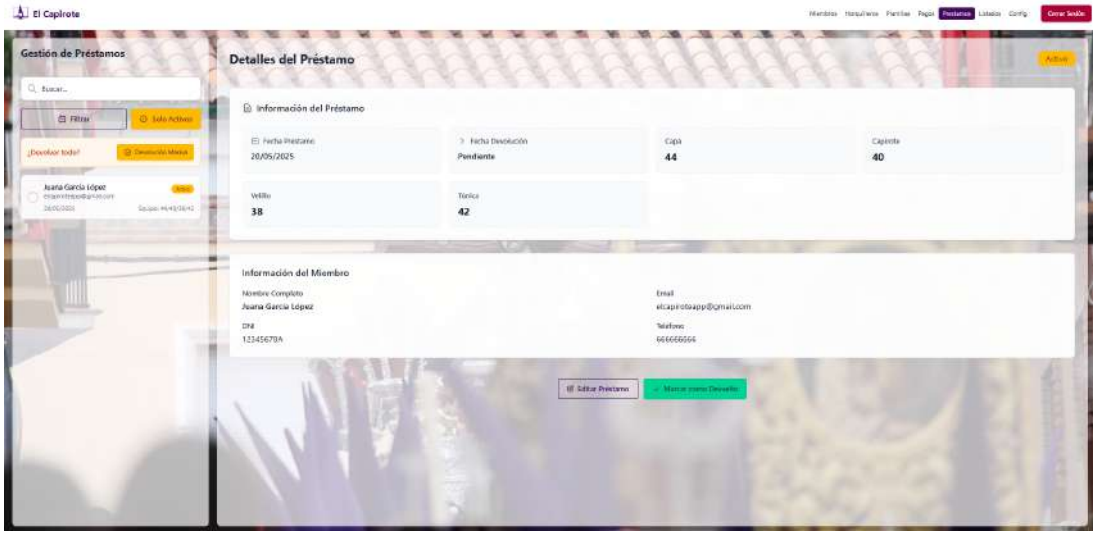


Figura 83: Filtrado de préstamos activos.

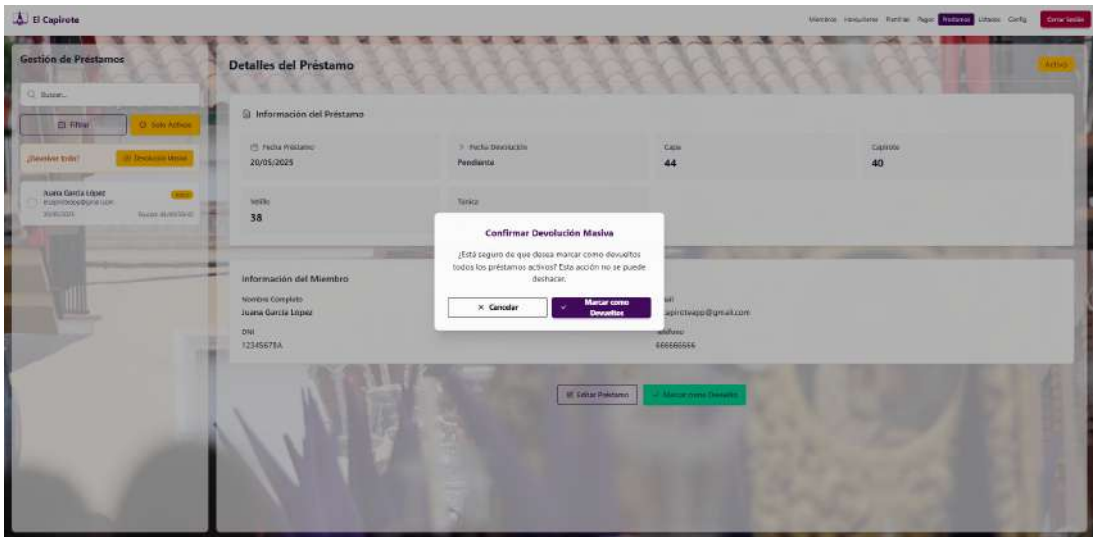


Figura 84: Confirmación de devolución masiva de préstamos.

Alternativamente, es posible seleccionar préstamos individuales en la lista y devolverlos de manera puntual.

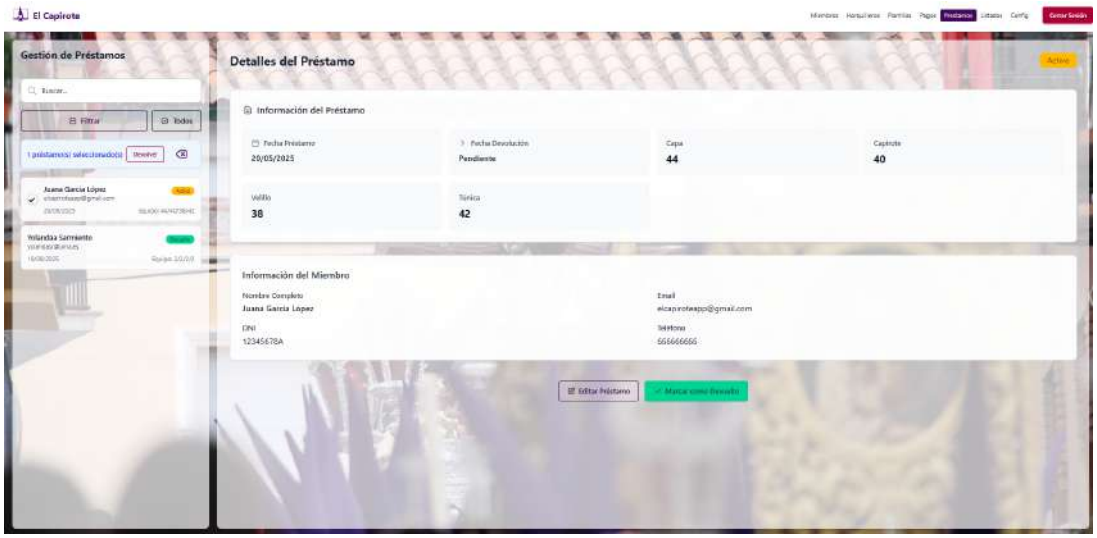


Figura 85: Selección individual de préstamos para devolución.

Dentro del detalle de cada préstamo se incluye la información del miembro y del equipo asignado (túnica, capa, capirote, velillo), junto con la **fecha de préstamo** y la **fecha de devolución** (si aplica). Desde esta vista se pueden realizar dos acciones principales: **marcar como devuelto** o **editar los datos del préstamo**.

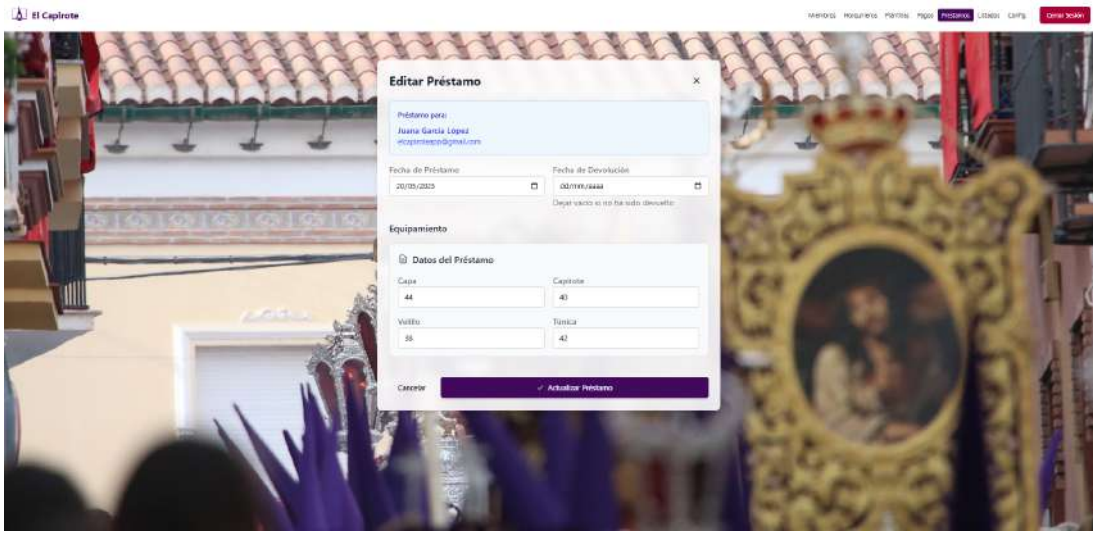


Figura 86: Interfaz para la edición de un préstamo.

La edición de un préstamo permite actualizar los números de las piezas de equipo, así como asignar la fecha de devolución cuando el material ha sido entregado.

B.3.6. Listados

El módulo de **Listados** ofrece una herramienta flexible para la generación de informes personalizados a partir de la base de datos de la hermandad. La interfaz está dividida en dos paneles principales: el **Constructor de Listados** y la sección de **Resultados**.

En el constructor, el usuario puede:

- Seleccionar la **tabla** sobre la que desea trabajar (por ejemplo, *Miembros*, *Horquilleros*, etc.).
- Escoger los **campos a mostrar** dentro del listado (como nombre, apellidos, DNI, dirección, teléfono, entre otros).
- Definir una o varias **condiciones de filtrado**, por ejemplo, mostrar únicamente a los horquilleros cuyo hombro sea igual a “Derecho”.

Una vez configurados los parámetros, al presionar el botón *Ejecutar Consulta*, el sistema genera un listado en el panel derecho con los resultados obtenidos, indicando el total de registros encontrados.

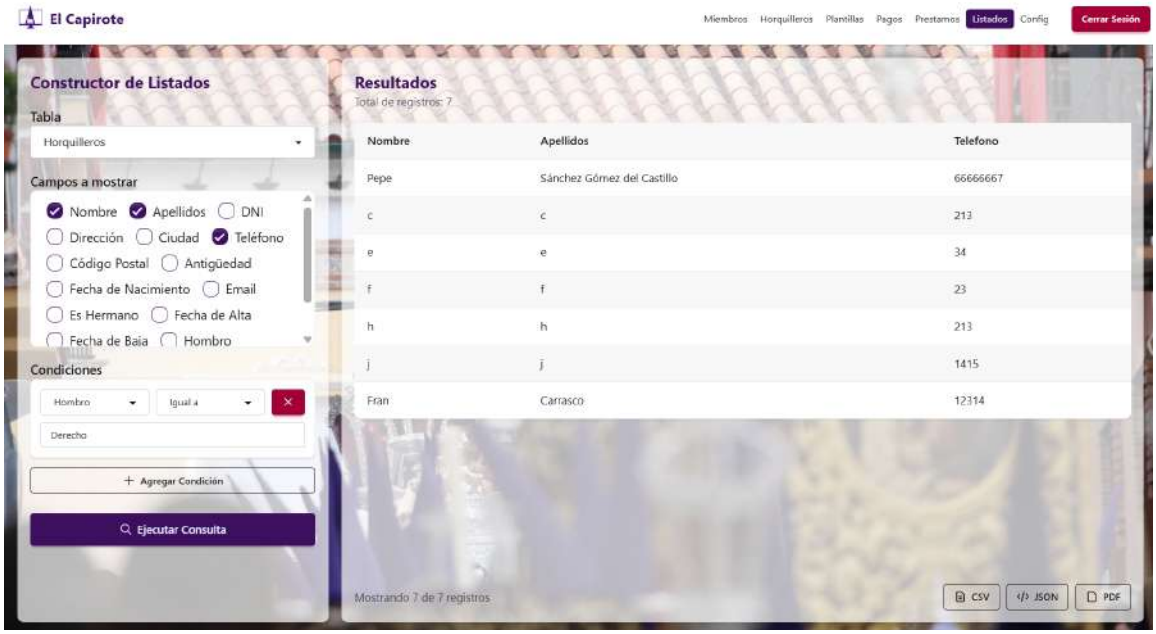


Figura 87: Constructor de listados y resultados filtrados.

Además, el listado resultante puede ser **exportado en distintos formatos**, incluyendo **CSV**, **JSON** o **PDF**, lo que permite al usuario adaptar la información a diferentes usos o integrarla con otros sistemas externos.

B.3.7. Configuración del sistema

La sección de **Config** centraliza todos los ajustes de la cofradía. El acceso abre un panel con cuatro bloques principales:

- **Configuración general**: datos básicos de la cofradía y reglas financieras.
- **Gestión de Tronos**: altas, edición y baja de tronos, con sus capacidades y cuota de horquillero.
- **Gestión de Secciones**: definición de secciones, capacidades y organización por trono.
- **Recursos dados de baja**: consulta y restauración de miembros, tronos y secciones eliminados; incluye limpieza definitiva.

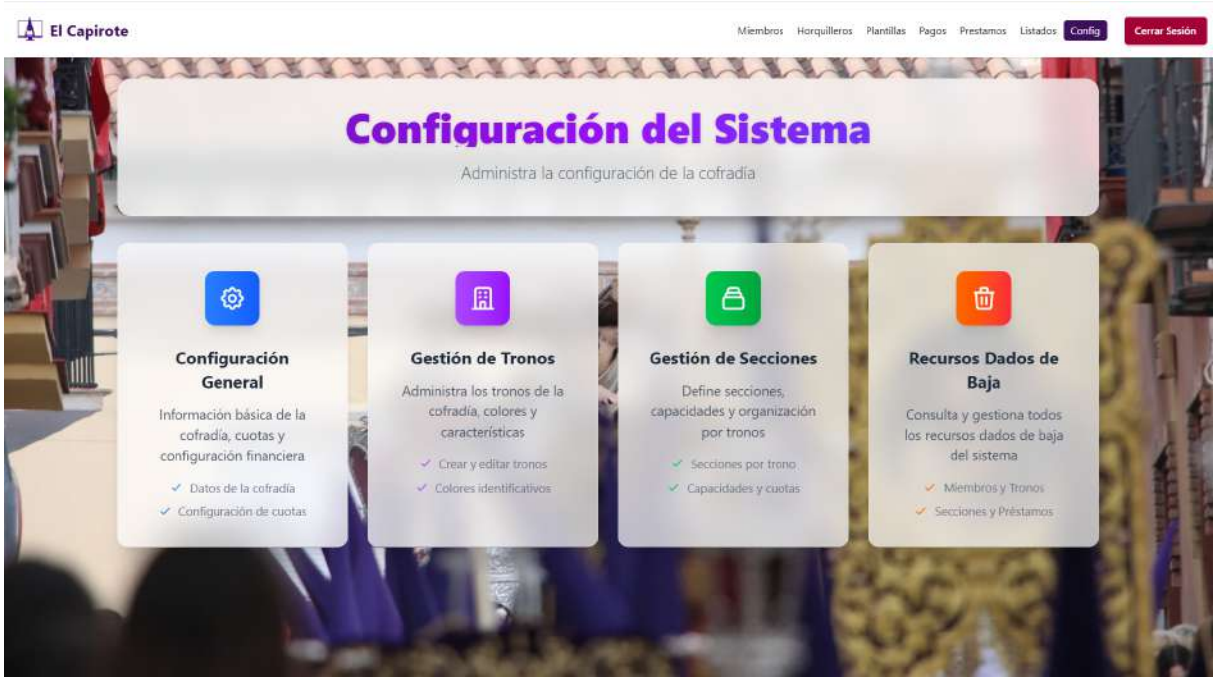


Figura 88: Portada de Configuración del sistema.

Configuración general. Desde aquí se establecen la *Cuota de Hermano (€/año)* y el *Máximo de baja (meses)*. Además, hay dos opciones conmutables:

- *Obligatoriedad de Hermano*: si es necesario ser hermano para participar.

- *Visibilidad de Puestos*: si los miembros pueden ver sus puestos asignados.

La pantalla incluye acciones de *Restablecer*, *Volver* y *Guardar configuración*, junto a un aviso que recuerda que los cambios pueden afectar al funcionamiento.

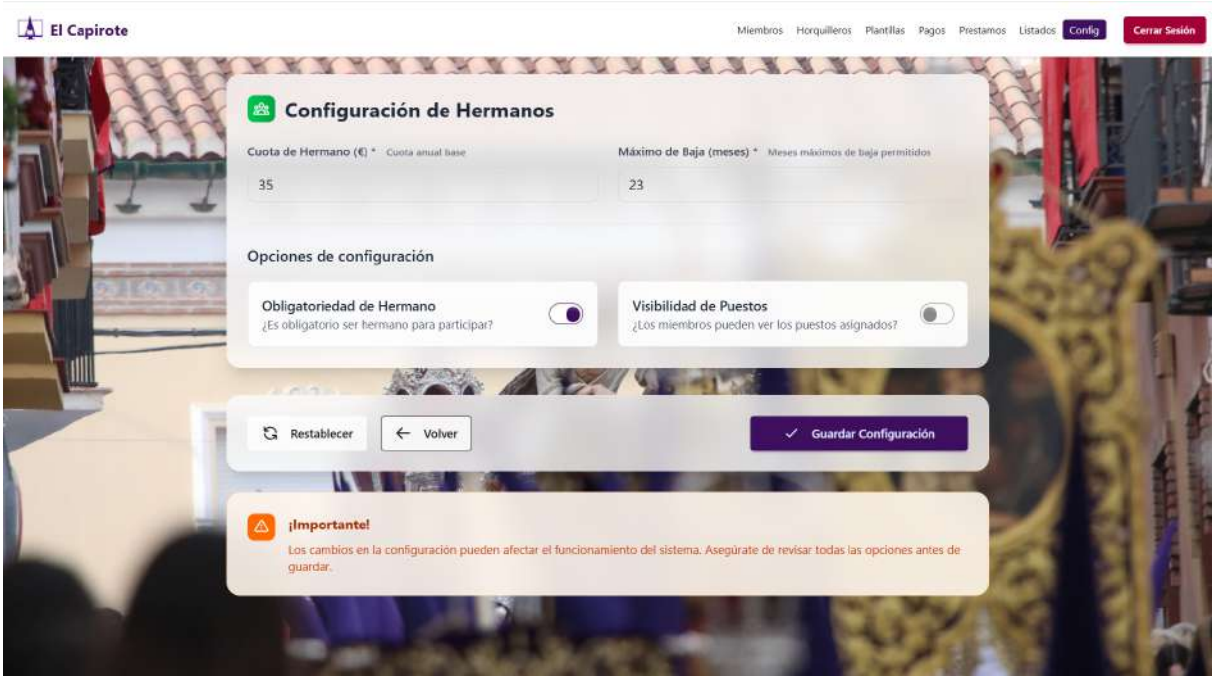


Figura 89: Parámetros generales de la cofradía.

Gestión de Tronos. Se muestran tarjetas por trono con métricas clave: *Capacidad de varales exteriores*, *interiores*, *Capacidad de mesa*, *Total de varales* y *Cuota de horquillero*. Desde *Nuevo Trono* se crean registros y, en el menú de cada tarjeta, se puede *Editar* o *Dar de baja*.

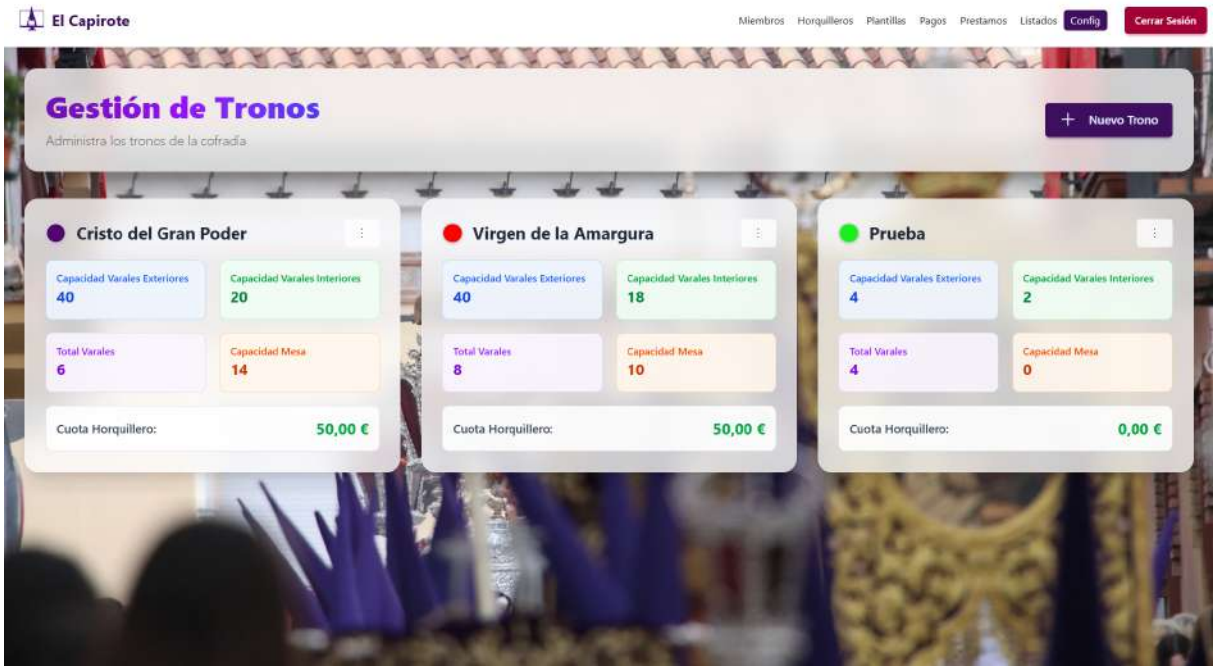


Figura 90: Listado de tronos y sus capacidades.

Al editar un trono se ajustan: *Nombre*, *Varales fuera*, *Varales dentro*, *Número de varales*, *Capacidad de mesa*, *Cuota de horquillero (€)* y *Color identificativo*. Los cambios se confirman con *Actualizar Trono*.

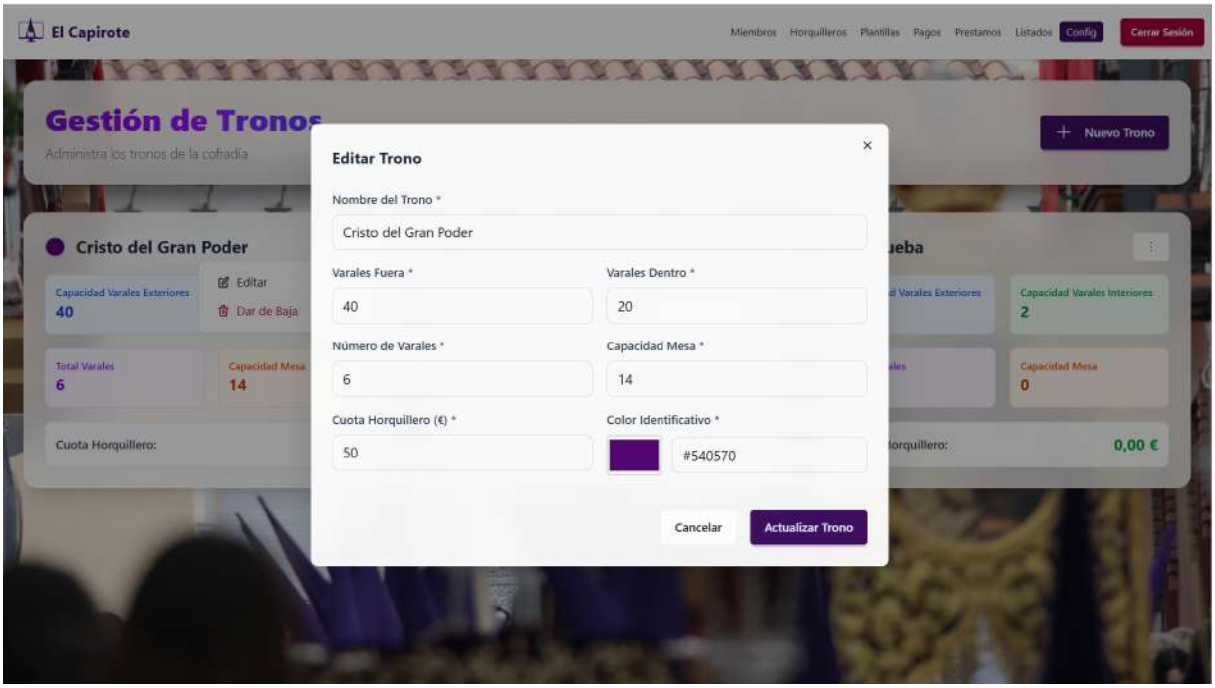


Figura 91: Edición de un trono.

Gestión de Secciones. Con una interfaz análoga a la de tronos, permite definir secciones por trono y sus *capacidades/cuotas*. Se pueden crear, editar y dar de baja secciones en cualquier momento.

Recursos dados de baja. Pantalla para consultar y gestionar elementos previamente dados de baja. Dispone de pestañas para *Miembros*, *Tronos* y *Secciones*, buscador por texto y acción *Restaurar* por elemento. Además, el botón *Limpieza de bajas* ejecuta una eliminación **permanente** de recursos con más de los meses de antigüedad definidos en la configuración, requiriendo doble confirmación para evitar errores.

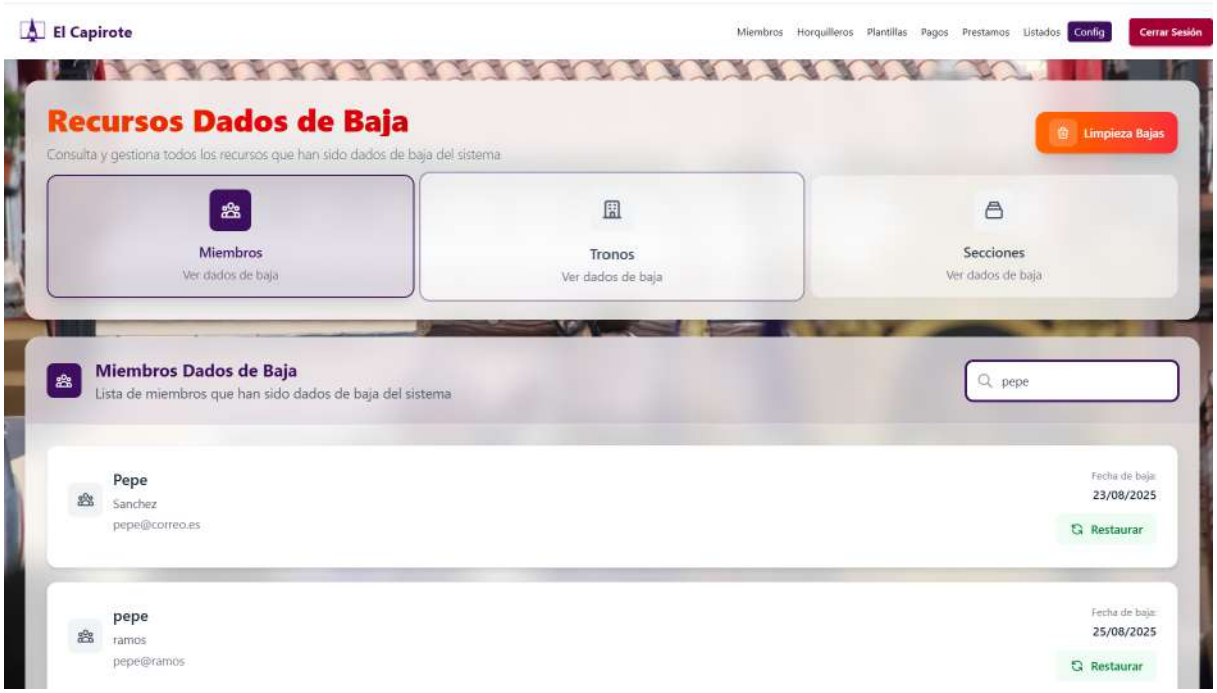


Figura 92: Recursos dados de baja: búsqueda y restauración.

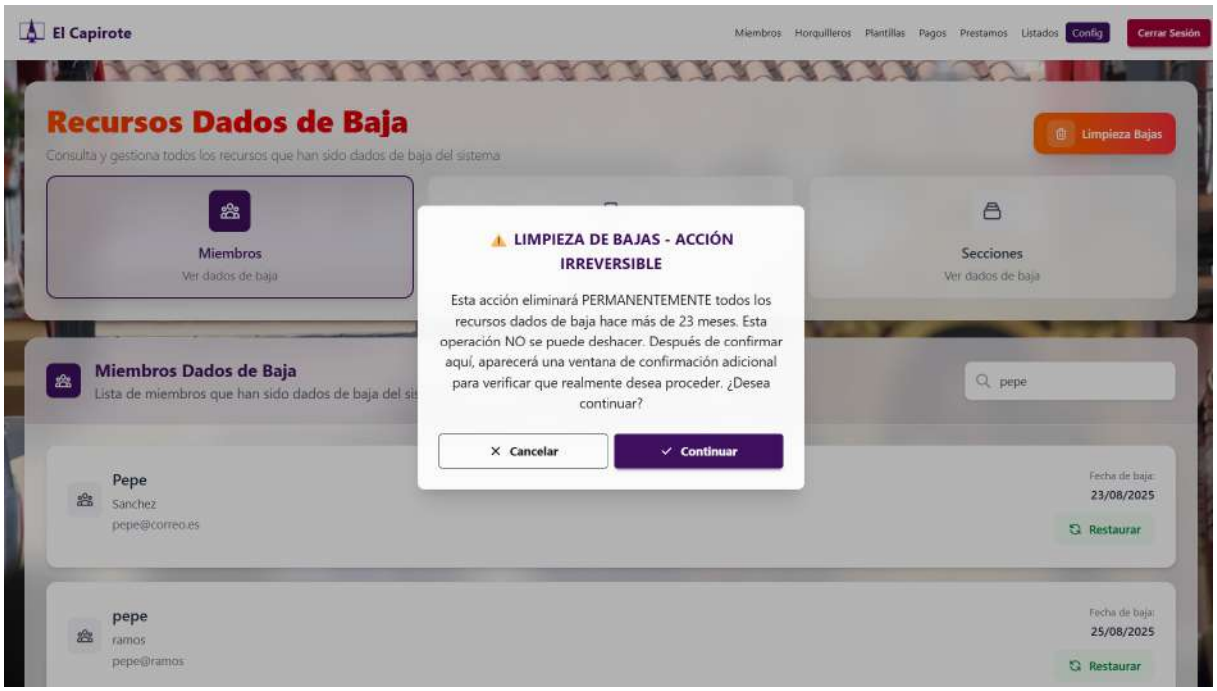


Figura 93: Diálogo de confirmación para la limpieza irreversible de bajas.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA