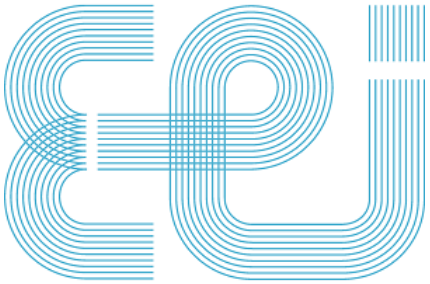


Escuela de Ingeniería Industrial

Trabajo Fin de Grado

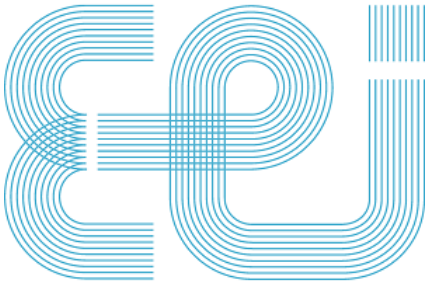
Diseño de una aplicación informática para la gestión de un almacén sin automatización con distintas interfaces para los operarios

Grado en Ingeniería en Electrónica Industrial y Automática

Alumno: José Tomás Torre Pedroarena

Directores: Joaquín lópez Fernández

Universidad de Vigo



Escuela de Ingeniería Industrial

Trabajo Fin de Grado

Diseño de una aplicación informática para la gestión de un almacén sin automatización con distintas interfaces para los operarios

Grado en Ingeniería en Electrónica Industrial y Automática

Documento

Memoria

Universidad de Vigo

Índice

Glosario de siglas

API Interfaz de Programación de Aplicaciones.

ERP Planificación de Recursos Empresariales.

GUI Interfaz Gráfica de Usuario.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

JSON JavaScript Object Notation.

REST Transferencia de Estado Representacional.

SQL Lenguaje de Consulta Estructurado.

WMS Sistema de Gestión de Almacenes.

XML Extensible Markup Language.

Definiciones

Backend Conjunto de servidores y componentes software encargados de proporcionar servicios, gestionar la lógica de negocio y permitir el acceso a recursos y datos desde aplicaciones cliente.

Base de datos relacional Una base de datos relacional es un conjunto de una o más tablas estructuradas en registros (líneas) y campos (columnas), que se vinculan entre sí por campos en común que poseen las mismas características en ambas tablas, como por ejemplo el nombre de campo, tipo y longitud. A estos campos generalmente se les denomina campos identificadores (ID) o campos clave. A esta forma de construir bases de datos se le denomina modelo relacional..

DAO Data Access Object, patrón de diseño que encapsula el acceso a la base de datos, separar el acceso a la base de datos de la lógica de negocio y proporciona una interfaz para realizar operaciones CRUD sobre una base de datos.

Framework Estructura de desarrollo compuesta por componentes de software reutilizables, librerías y convenciones que facilitan la creación de aplicaciones de forma estandarizada y eficiente.

FXML Es un archivo XML que permite describir el diseño y la estructura visual de una aplicación JavaFX.

JDBC Java Database Connectivity, API estándar de Java que permite a las aplicaciones acceder y gestionar bases de datos relacionales mediante la ejecución de consultas SQL y la manipulación de los resultados obtenidos.

ResultSet Estructura de datos utilizada en JDBC para representar el conjunto de resultados devuelto por una consulta a una base de datos, permitiendo recorrer fila a fila los registros obtenidos y acceder a los valores de cada columna.

SKU Stock Keeping Unit, identificador único de un producto en el inventario.

1 Introducción

La optimización de los procesos industriales y logísticos constituye uno de los pilares fundamentales en los entornos productivos actuales. Aspectos como el control operativo, la trazabilidad de los materiales o la reducción de errores influyen de manera directa tanto en los costes como en la calidad del servicio ofrecido. Dentro de este escenario, el almacén adquiere un papel determinante en la cadena de suministro, al actuar como punto de conexión entre producción, transporte y distribución.

Entre los distintos modelos de almacenamiento existentes, los almacenes de consolidación desempeñan una función específica centrada en la agrupación, organización y preparación de mercancía procedente de múltiples orígenes para su expedición conjunta. Este tipo de instalaciones presenta retos operativos propios, entre los que destacan un elevado número de movimientos, una rotación continua de productos y la necesidad de mantener un seguimiento preciso de palets, ubicaciones y pedidos.

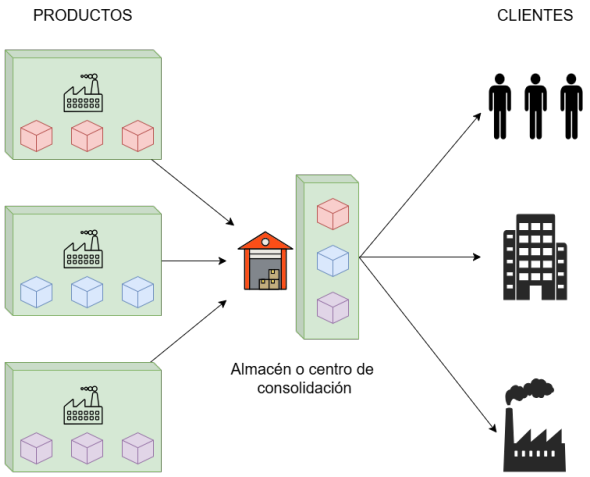


Figura 1: Almacén de consolidación en entorno industrial

La incorporación de sistemas digitales en la gestión de estos almacenes permite un mayor control del inventario, una localización más precisa de los materiales y una coordinación más ordenada de los flujos de entrada y salida. En este contexto se enmarca el presente Trabajo Fin de Grado, cuyo propósito es el desarrollo de una solución técnica orientada de forma específica a la gestión de almacenes de consolidación, concebida como apoyo directo a la operativa industrial diaria.

El proyecto aborda el diseño e implementación de un sistema compuesto por una aplicación de escritorio, una aplicación móvil para dispositivos Android y un backend responsable del tratamiento y almacenamiento de la información. El planteamiento adoptado responde a un enfoque práctico, alineado con los conocimientos adquiridos en la titulación de Ingeniería en Electrónica Industrial y Automática, y orientado a la aplicación real de tecnologías software como soporte a procesos logísticos.

1.1 Contexto y motivación del proyecto

En entornos industriales y logísticos, la gestión del almacén condiciona de forma directa la continuidad de la producción y el cumplimiento de los plazos establecidos. En los almacenes de consolidación, una gestión deficiente puede derivar en desajustes de inventario, tiempos improductivos durante la preparación de pedidos, errores en la agrupación de mercancía o dificultades para conocer el estado real de los materiales.

Existen soluciones comerciales destinadas a la gestión de almacenes, aunque muchas de ellas están orientadas a grandes centros logísticos generalistas o incorporan un nivel de complejidad elevado. Esta situación complica su implantación en instalaciones de menor escala o con flujos de trabajo muy concretos, donde la inversión en infraestructura, formación o consultoría resulta poco viable.

La motivación principal de este proyecto surge de la necesidad de contar con una herramienta accesible y comprensible, pensada para entornos que carecen de soluciones específicas para la gestión de almacenes de consolidación. La propuesta se centra en cubrir las funciones esenciales de este tipo de instalaciones e integrarse de forma natural en la rutina diaria del almacén mediante interfaces claras y orientadas al usuario. La combinación de una aplicación de escritorio para tareas de planificación y una aplicación móvil para la operativa en planta responde a situaciones habituales en entornos industriales reales.

Desde el punto de vista académico, el desarrollo del proyecto permite aplicar conocimientos relacionados con sistemas de información, arquitectura de software, gestión de datos y diseño de interfaces, reforzando la formación técnica en automatización y electrónica industrial con una visión orientada a la resolución de problemas logísticos.

1.2 Problema a resolver

El problema abordado en este Trabajo Fin de Grado se centra en el desarrollo de sistemas integrados y adaptados a la realidad de los almacenes de consolidación en entornos industriales. En concreto, se plantea la necesidad de gestionar de manera estructurada la información asociada a productos, palets, ubicaciones, movimientos y pedidos.

En numerosas instalaciones, estas tareas se apoyan en herramientas genéricas, soluciones parciales o procedimientos manuales, lo que incrementa el riesgo de errores y dificulta el acceso a información actualizada y coherente. Este escenario puede traducirse en pérdidas de material, desajustes de inventario y una trazabilidad limitada, especialmente en instalaciones con alta rotación de mercancía.

El sistema propuesto plantea una respuesta a esta problemática mediante una solución que:

- Centraliza la información del almacén de consolidación en una base de datos común.
- Permite el control estructurado del inventario y la gestión de ubicaciones.
- Apoya la gestión de pedidos y el registro de movimientos durante los procesos de consolidación.

El alcance del proyecto se limita al desarrollo de un sistema de apoyo a la gestión del almacén, sin intervenir en el control directo de equipos automatizados. Aun así, la arquitectura planteada contempla la posibilidad de integración futura con sistemas de control de planta o automatización industrial.

1.3 Objetivos del TFG

El objetivo general de este Trabajo Fin de Grado consiste en el desarrollo de un sistema de gestión orientado a almacenes de consolidación en entornos industriales, con el fin de mejorar el control, la organización y la trazabilidad de los materiales mediante aplicaciones software integradas.

A partir de este objetivo general, se definen los siguientes objetivos específicos:

- Estudiar las necesidades operativas propias de un almacén de consolidación industrial.
- Definir una arquitectura que permita la comunicación entre aplicaciones de escritorio, móviles y un backend común.

- Desarrollar la gestión de productos, palets, ubicaciones y pedidos.
- Implementar un sistema de almacenamiento de datos que asegure la coherencia y disponibilidad de la información.
- Diseñar interfaces orientadas a la operativa diaria del personal del almacén.

2 Estado del arte

En este capítulo se presenta el estado del arte relacionado con los sistemas de gestión de almacenes, con especial atención a los almacenes de consolidación. Se describen los conceptos generales asociados a los Sistemas de Gestión de Almacenes (WMSs), las funcionalidades habituales que incorporan este tipo de aplicaciones y algunas de las soluciones comerciales existentes, con el objetivo de contextualizar la solución propuesta en este Trabajo Fin de Grado.

2.1 Sistemas de gestión de almacenes (WMS)

Un Sistema de Gestión de Almacenes, conocido como WMS (*Warehouse Management System*), es una herramienta software diseñada para apoyar y optimizar las operaciones que se llevan a cabo en un almacén. Su principal función es gestionar de forma estructurada la información relacionada con el inventario, las ubicaciones, los movimientos de mercancía y los procesos de entrada y salida de materiales.

Los sistemas WMS actúan como un elemento central dentro de la cadena logística, proporcionando visibilidad sobre el estado del almacén y facilitando la toma de decisiones operativas. En entornos industriales, estos sistemas permiten reducir errores asociados a la gestión manual, mejorar la trazabilidad de los productos y optimizar el uso del espacio y de los recursos disponibles.

En función del tipo de instalación, los WMS pueden adaptarse a diferentes escenarios, como almacenes de producción, centros de distribución o almacenes de consolidación. En este último caso, el sistema debe gestionar un elevado volumen de movimientos, coordinar la agrupación de mercancía procedente de diferentes orígenes y garantizar una correcta trazabilidad de palets y pedidos hasta su expedición.

Desde un punto de vista funcional, un WMS suele estructurarse en distintos módulos que cubren aspectos como la gestión de inventario, el control de ubicaciones, la planificación de operaciones y la generación de informes. Asimismo, estos sistemas pueden integrarse con otros sistemas de información, como sistemas de planificación de recursos empresariales (ERP) o sistemas de control industrial, dependiendo del nivel de automatización de la instalación.

2.2 Funcionalidades habituales en aplicaciones de gestión de almacenes

Las aplicaciones de gestión de almacenes incorporan un conjunto de funcionalidades orientadas a cubrir las necesidades operativas básicas de una instalación logística o industrial. Aunque estas funcionalidades pueden variar en función del tipo de almacén y del sector, existen una serie de características comunes ampliamente presentes en los sistemas WMS.

Entre las funcionalidades más habituales se encuentran la gestión de la recepción de mercancía, que permite registrar la entrada de productos en el almacén y asociarlos a ubicaciones específicas. Asimismo, la gestión de ubicaciones resulta fundamental para organizar el espacio disponible y facilitar la localización de materiales durante las operaciones diarias.

Otra funcionalidad clave es la gestión del inventario, que proporciona información actualizada sobre las existencias disponibles y permite detectar desajustes o incidencias. En el caso de los almacenes de consolidación, esta funcionalidad adquiere especial relevancia debido a la alta rotación de productos y a la necesidad de coordinar múltiples movimientos de entrada y salida.

Las aplicaciones WMS también suelen incluir herramientas para la gestión de pedidos y la preparación de expediciones, facilitando la agrupación de mercancía y el seguimiento de los envíos. La trazabilidad de los materiales, especialmente a nivel de palet, constituye otro aspecto esencial, ya que permite conocer el historial de movimientos y garantizar un control adecuado de la

mercancía.

Finalmente, muchas aplicaciones incorporan funcionalidades de apoyo como la generación de informes, el control de usuarios y permisos, y la integración con dispositivos móviles, lo que resulta especialmente útil para la operativa en planta y para la reducción de errores en tareas repetitivas.

2.3 Aplicaciones comerciales de gestión de almacenes

En el mercado existen numerosas soluciones comerciales orientadas a la gestión de almacenes, que ofrecen un amplio abanico de funcionalidades y niveles de complejidad. Estas soluciones suelen formar parte de plataformas logísticas o sistemas empresariales más amplios, y están diseñadas para adaptarse a diferentes tipos de instalaciones y volúmenes de operación.

Algunas de estas aplicaciones están orientadas a grandes centros logísticos o a entornos altamente automatizados, incorporando funcionalidades avanzadas de optimización y control. Sin embargo, este enfoque puede suponer una barrera para su adopción en instalaciones de menor escala o en almacenes de consolidación con procesos específicos, debido a la complejidad de configuración y a los costes asociados a su implantación y mantenimiento.

Otras soluciones comerciales ofrecen versiones más simplificadas, pero aun así suelen estar basadas en flujos de trabajo genéricos que no siempre se ajustan de forma óptima a las particularidades de cada instalación. En muchos casos, la adaptación a procesos concretos requiere desarrollos adicionales o modificaciones en la operativa existente, lo que reduce la flexibilidad del sistema.

En este contexto, se observa una brecha entre las soluciones altamente especializadas y complejas, y la necesidad de herramientas más accesibles y adaptables a entornos industriales concretos, como los almacenes de consolidación que no cuentan con sistemas de gestión avanzados.

2.4 Justificación de la solución propuesta

A partir del análisis del estado del arte, se pone de manifiesto la necesidad de disponer de soluciones de gestión de almacenes que, sin alcanzar la complejidad de los grandes sistemas comerciales, permitan cubrir de forma eficaz las necesidades operativas de instalaciones concretas, como los almacenes de consolidación.

La solución propuesta en este Trabajo Fin de Grado se justifica por su enfoque específico hacia este tipo de almacenes, priorizando la simplicidad de uso, la adaptación a flujos de trabajo reales y la integración de distintas plataformas de acceso. El sistema se concibe como una herramienta de apoyo a la operativa industrial, facilitando la gestión de inventario, la trazabilidad de palets y la coordinación de pedidos sin imponer una reestructuración completa de los procesos existentes.

Además, la combinación de una aplicación de escritorio para tareas de gestión y planificación, junto con una aplicación móvil para la operativa en planta, permite cubrir distintos escenarios de uso habituales en entornos industriales. Este enfoque contribuye a mejorar la accesibilidad a la información y a reducir errores derivados de la introducción manual de datos.

En conjunto, la propuesta se sitúa como una alternativa intermedia entre soluciones comerciales complejas y la ausencia de herramientas específicas, ofreciendo una base flexible que puede evolucionar e integrarse en el futuro con otros sistemas de automatización industrial.

3 Marco teórico y tecnológico

En este capítulo se presentan los fundamentos teóricos y tecnológicos que se emplearon en el desarrollo del sistema propuesto. Se abordan los conceptos relacionados con los sistemas de gestión de almacenes, las arquitecturas cliente-servidor, el desarrollo de aplicaciones multiplataforma, las aplicaciones móviles Android, las aplicaciones de escritorio y los mecanismos de comunicación entre aplicaciones, incluyendo el uso de Interfaz de Programación de Aplicacioness (APIs) de tipo Transferencia de Estado Representacional (REST) en determinados componentes del sistema.

3.1 Aplicaciones multiplataforma

¿Cómo logramos que el jefe de almacén y el operario empleen el mismo sistema e interactúen con los mismos datos? La respuesta está en el desarrollo multiplataforma. En un entorno donde conviven ordenadores de oficina y terminales móviles, es vital que la lógica del sistema no se fragmente. Al compartir una base común (base de datos) ambos dispositivos deben interactuar entre sí y con el servidor de manera coherente.

Desde un punto de vista funcional, la adopción de soluciones multiplataforma facilita la reutilización de la lógica de negocio y la centralización de la información, reduciendo los costes de mantenimiento y mejorando la coherencia del sistema. En el ámbito de la gestión de almacenes, este enfoque permite que distintos perfiles de usuario accedan al sistema desde interfaces adaptadas a sus necesidades manteniendo una única fuente de datos consistente, por ejemplo: un operario necesita una dispositivo movil para poder realizar sus actividades, por tanto, usará un dispositivo móvil mientras que el jefe de almacén, que requiere una interfaz más completa, accederá desde un ordenador de oficina.

No obstante, el desarrollo multiplataforma también plantea retos, como la adaptación de la interfaz de usuario a diferentes dispositivos o la necesidad de garantizar un comportamiento homogéneo, es decir, que las operaciones realizadas desde un dispositivo se reflejen correctamente en el otro. Estos aspectos deben ser considerados cuidadosamente durante el diseño y la implementación del sistema.

3.2 Arquitecturas cliente-servidor

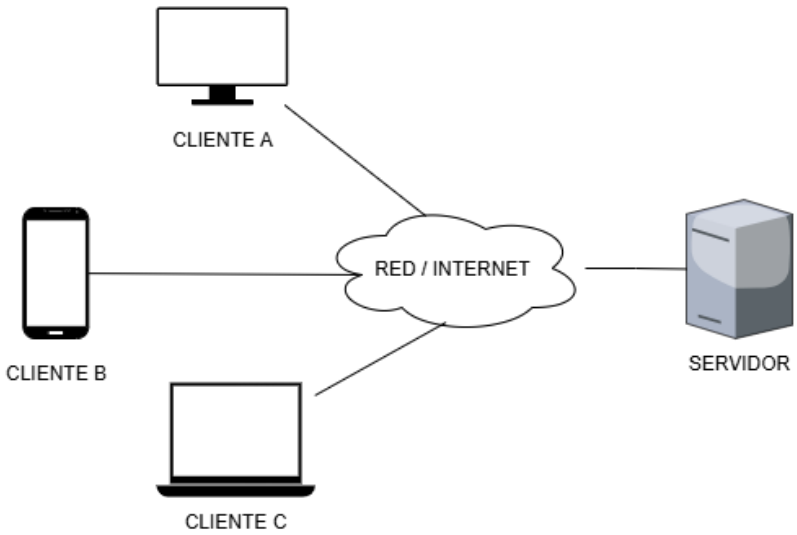


Figura 2: Arquitectura cliente-servidor

La arquitectura cliente-servidor es un modelo ampliamente utilizado en sistemas distribuidos[1], en el que las funciones del sistema se reparten entre componentes cliente y un servidor central. Los clientes se encargan de la interacción con el usuario, mientras que el servidor gestiona la lógica de negocio y la persistencia de los datos.

Este modelo resulta adecuado y conveniente para sistemas de gestión en entornos industriales, dado que permite centralizar la información y garantizar el acceso concurrente desde múltiples dispositivos. El sistema desarrollado se despliega en una red interna privada, prescindiendo del acceso a Internet, donde tanto la aplicación de escritorio como las dispositivos móviles Android funcionan como clientes que se comunican con el servidor central.

Además, este tipo de arquitectura favorece la escalabilidad del sistema y su posible integración futura con otros sistemas, como plataformas empresariales o sistemas de automatización industrial, al proporcionar un punto centralizado de acceso a los servicios y a los datos.

3.3 Aplicaciones móviles Android

Android es uno de los sistemas operativos más extendidos en dispositivos móviles, "un 78,8 % de los españoles con smartphone tuvieron Android como sistema operativo"[2].

Según la documentación oficial de Android este es una arquitectura basada en componentes como actividades, servicios, entre otros, que permiten gestionar la interfaz de usuario y el ciclo de vida de la aplicación [3].

En entornos industriales y logísticos, el uso de aplicaciones móviles facilita el acceso a la información en tiempo real y reduce la necesidad de desplazamientos innecesarios o registros manuales. En particular, en almacenes de consolidación, las aplicaciones Android permiten registrar movimientos, consultar inventario o identificar palets directamente en el punto de operación.

Dentro del sistema desarrollado, la aplicación android pretende ser una herramienta de apoyo para los operarios del almacén, proporcionando una interfaz sencilla para el registro de las operaciones que estos realizan en planta.

3.4 Aplicaciones de escritorio

Las aplicaciones de escritorio continúan desempeñando un papel fundamental en entornos industriales para tareas de gestión, supervisión y planificación. Estas aplicaciones suelen ejecutarse en estaciones de trabajo fijas y permiten ofrecer interfaces más completas para la visualización y administración de grandes volúmenes de información.

En el sistema desarrollado, la aplicación de escritorio se concibe como la herramienta principal para la gestión del almacén de consolidación, permitiendo la administración de productos, ubicaciones, pedidos y usuarios. Este tipo de aplicación resulta especialmente adecuada para tareas que requieren una visión global del sistema y un mayor nivel de detalle en la información presentada.

Asimismo, la aplicación integra todas las funcionalidades de gestión y administración y está destinada principalmente a usuarios con privilegios elevados (SysAdmin, Gestor Almacén, Administración). La gestión de roles y permisos se detalla en la sección 4.2. La aplicación de escritorio también puede ser usada por operarios en caso de ser necesario.

[1]Universidade de Vigo. "Arquitectura Cliente/Servidor." Material docente de la asignatura Sistemas Cliente/Servidor. Último acceso: enero 2026. dirección: `https://ccia.esei.uvigo.es/docencia/SCS/1011/transparencias/Tema1.pdf` p. 1

[2]Comisión Nacional de los Mercados y la Competencia. "Panel de hogares: usos de Internet." Gráfica "Sistema operativo del smartphone". Último acceso: enero 2026. dirección: `https://www.cnmc.es/prensa/panel-hogares-usos-internet-20231103` Fragmento: gráfica "Sistema operativo del smartphone"

La combinación de aplicaciones de escritorio y móviles permite adaptar la interacción con el sistema a distintos contextos de trabajo, teniendo en cuenta que los perfiles de operario y gestor de almacén presentan necesidades diferenciadas. Esta aproximación contribuye a una experiencia de uso más ajustada y a un desarrollo más ordenado de la actividad dentro del almacén.

3.5 Comunicación entre aplicaciones (APIs REST)

La comunicación entre los distintos componentes del sistema puede realizarse mediante interfaces de programación de aplicaciones (APIs). Una API puede definirse como un " [...] conjunto de comandos, funciones y protocolos informáticos que permiten crear programas que interactúen con otras aplicaciones [...]"[3]. Este mecanismo resulta especialmente adecuado en escenarios donde existen clientes heterogéneos que requieren acceso controlado a la información del sistema.

En este contexto, las APIs de tipo REST constituyen uno de los enfoques más utilizados para la comunicación entre sistemas distribuidos. REST es un conjunto de principios arquitectónicos orientados al diseño de interfaces entre sistemas, basado en el uso del protocolo Hypertext Transfer Protocol (HTTP), o bien Hypertext Transfer Protocol Secure (HTTPS) como medio de comunicación, sin añadir capas adicionales de abstracción [4].

Entre las principales características de las arquitecturas REST se encuentra el empleo de HTTP para la realización de operaciones sobre los recursos del sistema, así como el intercambio de datos mediante formatos estructurados, siendo JavaScript Object Notation (JSON) y Extensible Markup Language (XML) los más habituales. Los sistemas que implementan estos principios se denominan sistemas RESTful.

Una API REST, o API RESTful, es por tanto una interfaz que expone los recursos de un sistema siguiendo los principios REST, permitiendo que los clientes accedan a dichos recursos mediante peticiones HTTP estandarizadas y reciban las respuestas en formatos estructurados. Este enfoque favorece la interoperabilidad, el desacoplamiento entre clientes y servidor y la evolución independiente de los distintos componentes del sistema.

En entornos industriales y logísticos, el uso de APIs REST permite integrar aplicaciones de escritorio, aplicaciones móviles y sistemas externos, garantizando un acceso coherente y controlado a la información del sistema. Además, este tipo de interfaces facilita la escalabilidad de la aplicación y su posible integración futura con otros sistemas de información o plataformas de automatización industrial.

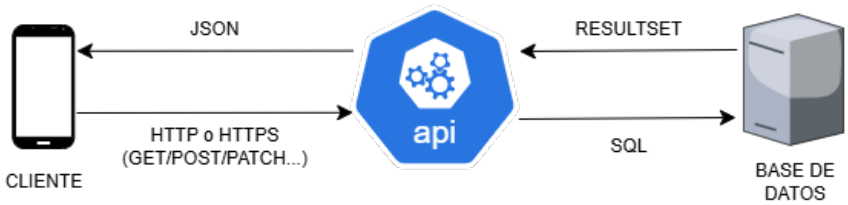


Figura 3: Comunicación mediante APIs REST

En el sistema desarrollado, el uso de una API REST se aplica específicamente en la comunicación entre la aplicación móvil Android y el backend del sistema, implementado mediante Spring Boot. La aplicación de escritorio, por su parte, se conecta directamente al servidor de base de datos, actuando como un cliente especializado para tareas de gestión y administración. Esta aproxima-

[3]Asociación Española de Banca. "APIs." Documento institucional. Último acceso: enero 2026. dirección: `https://s1.aebanca.es/wp-content/uploads/2018/05/apis.pdf` p. 1

[4]Iniciativa Aporta. "Buenas prácticas en el diseño de APIs y linked data." Gobierno de España. dirección: `https://datos.gob.es/elearning/Unidades_Didacticas/Unidad_8/contenidos/descargas/unidad_imprimible.pdf` p. 10

ción permite adaptar el mecanismo de comunicación a las necesidades y características de cada tipo de cliente.

3.6 Backend y framework Spring Boot

Spring Boot es un Framework de desarrollo basado en el lenguaje Java que facilita la creación de aplicaciones de tipo Backend y servicios web. Está diseñado para simplificar la configuración y el despliegue de aplicaciones, proporcionando una estructura predefinida y mecanismos de autoconfiguración que reducen la necesidad de configuraciones manuales [6], [7], [8].

En el contexto de sistemas distribuidos, Spring Boot se utiliza habitualmente para implementar APIs de tipo REST, permitiendo exponer la lógica de negocio y el acceso a los datos mediante servicios accesibles por distintos clientes. Su integración con el ecosistema Java y su compatibilidad con tecnologías de persistencia lo convierten en una opción adecuada para el desarrollo de Backends en entornos industriales y empresariales.

3.7 Acceso a la base de datos en la aplicación de escritorio

En el sistema desarrollado se ha adoptado una estrategia diferenciada para el acceso a los datos en función del tipo de aplicación cliente. Mientras que la aplicación móvil Android accede a la información del sistema a través de una API de tipo REST, la aplicación de escritorio establece una conexión directa con el servidor de base de datos.

Esta decisión se fundamenta en el contexto de uso de la aplicación de escritorio, concebida como una herramienta de gestión interna destinada a tareas de administración, planificación y supervisión del almacén. Dicha aplicación se ejecuta en ordenadores de escritorio dentro de la red interna de la instalación y es utilizada por perfiles de usuario con permisos elevados (SysAdmin, Gestor Almacén, Administración), lo que permite asumir un entorno controlado desde el punto de vista de seguridad y acceso.

El acceso directo a la base de datos, implementado mediante tecnologías estándar de conectividad en Java, JDBC, permite simplificar la arquitectura del sistema para este tipo de cliente y reducir la latencia asociada a la comunicación a través de servicios intermedios.

No obstante, esta arquitectura implica un mayor grado de acoplamiento entre la aplicación de escritorio y el modelo de datos, lo que puede limitar la flexibilidad ante modificaciones en el esquema de la base de datos. Por este motivo, el acceso directo se ha restringido exclusivamente a la aplicación de escritorio y a un entorno controlado, evitando su uso en clientes móviles o en escenarios con mayores requisitos de interoperabilidad.

En conjunto, la implementación de ambos mecanismos de acceso a los datos, conexión directa a la base de datos para la aplicación de escritorio y API REST para la aplicación móvil, permite adaptar la arquitectura del sistema a las necesidades específicas de cada tipo de cliente. Esta estrategia, combinada con la gestión de roles y permisos, contribuye a reforzar la integridad, la eficiencia y la coherencia global del sistema.

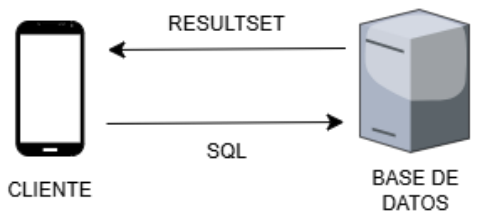


Figura 4: Comunicación directa entre la aplicación de escritorio y la base de datos

4 Análisis del sistema

4.1 Descripción general del sistema

El sistema desarrollado tiene como objetivo servir de herramienta de apoyo para la gestión de un almacén de consolidación en un entorno industrial. La aplicación permite centralizar la información relacionada con productos, palets, ubicaciones, movimientos y pedidos, facilitando la operativa diaria del almacén y el seguimiento de la mercancía a lo largo de los distintos procesos internos.

La solución se concibe como un sistema distribuido que da soporte a distintos perfiles de usuario mediante interfaces diferenciadas. Por un lado, una aplicación de escritorio orientada a tareas de gestión, planificación y supervisión; por otro, una aplicación móvil destinada a la operativa en planta, permitiendo el acceso a la información y el registro de acciones directamente en el entorno de trabajo.

El sistema está específicamente diseñado para almacenes de consolidación, donde se agrupan mercancías procedentes de distintos proveedores con el fin de preparar envíos conjuntos a clientes. En consecuencia, el alcance funcional se centra en procesos como la gestión de inventario, la consolidación de pedidos y la trazabilidad de los movimientos internos, sin abordar otros tipos de almacén ni el control directo de sistemas automatizados.

4.2 Identificación de usuarios

Funcionalidad	SysAdmin	Gestor Almacén	Operario	Administración
Acceso a la gestión general del almacén	Sí	Sí	Sí	No
Consulta y gestión del inventario	Sí	Sí	Sí	No
Gestión de pedidos	Sí	Sí	No	Sí
Paletización de mercancía	Sí	Sí	Sí	No
Gestión de envíos	Sí	Sí	Sí	No
Consulta del calendario de operaciones	Sí	Sí	Sí	Sí
Registro de movimientos de mercancía	Sí	Sí	Sí	No
Actualización de información de palets	Sí	Sí	No	No
Gestión de órdenes de compra	Sí	Sí	No	Sí
Exportación de datos del sistema	Sí	No	No	Sí
Creación de pedidos	Sí	Sí	No	Sí
Edición de pedidos	Sí	Sí	No	Sí
Eliminación de pedidos	Sí	Sí	No	Sí
Creación de productos	Sí	Sí	No	No
Creación de tipos de producto	Sí	Sí	No	No
Creación de usuarios	Sí	No	No	Sí
Edición de usuarios	Sí	No	No	Sí
Eliminación de usuarios	Sí	No	No	Sí

Tabla 1: Permisos de acceso a funcionalidades del sistema según rol de usuario.

El sistema contempla distintos tipos de usuarios o roles, definidos en función de su rol dentro del almacén. La aplicación de los roles se aplica exclusivamente a la aplicación de escritorio, donde se concentran las funcionalidades de gestión, administración y supervisión del sistema. En la aplicación móvil Android, orientada a la operativa en planta, todos los usuarios actúan bajo un perfil funcional equivalente al de operario, independientemente de su rol dentro de la organización.

En la aplicación de escritorio se han definido cuatro roles principales: Administrador del sistema (SysAdmin), Gestor de almacén, Operario y Administración. Cada uno de estos roles dispone de un conjunto específico de permisos que determinan el acceso a las distintas funcionalidades del

sistema. La Tabla 1 recoge de forma resumida los permisos asignados a cada rol en relación con las funcionalidades disponibles. Por ejemplo: Un SysAdmin puede crear, editar y eliminar usuarios, mientras que un Operario no tiene acceso a estas funciones.

El rol de SysAdmin dispone de acceso completo a todas las funcionalidades del sistema, incluyendo la gestión de usuarios, la configuración general, la administración de productos, pedidos y la exportación de datos. Este perfil está orientado a tareas de administración avanzada y mantenimiento del sistema.

El Gestor de almacén es responsable de la planificación y supervisión de la operativa diaria. Este rol puede gestionar inventario, pedidos, palets, envíos y órdenes de compra, así como realizar tareas relacionadas con la organización del almacén, pero no dispone de permisos para la gestión de usuarios ni para la exportación de datos del sistema.

El rol de Administración está orientado a tareas de carácter administrativo y documental. Este perfil puede gestionar pedidos, órdenes de compra, usuarios y exportar información del sistema, pero no interviene directamente en la operativa física del almacén ni en la gestión de movimientos de mercancía.

Por último, el Operario de almacén se encarga de ejecutar las tareas operativas relacionadas con la manipulación y movimiento de mercancías. En la aplicación de escritorio, su acceso se limita a funcionalidades operativas como la consulta de información, la paletización, el registro de movimientos y la gestión de envíos, sin permisos para tareas de administración o gestión avanzada. En la aplicación móvil Android, este rol constituye el único perfil de uso, permitiendo registrar movimientos, consultar inventario y apoyar los procesos de consolidación directamente en planta.

Esta estructura de roles y permisos permite limitar las funcionalidades y las interfaces del sistema a las responsabilidades de cada tipo de usuario, mejorando la eficiencia operativa, reforzando el control de acceso y reduciendo la probabilidad de errores derivados del uso indebido de funcionalidades no autorizadas.

4.3 Requisitos del sistema

4.3.1 Requisitos funcionales

El sistema debe cumplir los siguientes requisitos funcionales:

- Permitir la gestión de productos y sus características asociadas.
- Gestionar palets y su ubicación dentro del almacén.
- Registrar movimientos internos de mercancía y cambios de ubicación.
- Facilitar la preparación y consolidación de pedidos.
- Permitir el acceso al sistema a distintos perfiles de usuario.
- Proporcionar interfaces diferenciadas para tareas de gestión y operativa en planta.
- Consultar el estado del inventario y la trazabilidad de los productos.

4.3.2 Requisitos no funcionales

Además de los requisitos funcionales, el sistema debe cumplir una serie de requisitos no funcionales:

- El sistema debe ser fácil de usar por personal con distintos niveles de experiencia.
- La información debe mantenerse consistente y actualizada en todo momento.
- El acceso a los datos debe estar controlado según el perfil de usuario.
- El sistema debe ser mantenible y permitir futuras ampliaciones funcionales.

- La solución debe ser compatible con entornos industriales habituales.
- El sistema debe garantizar la seguridad de las credenciales de los usuarios, evitando el almacenamiento de contraseñas en texto plano y aplicando mecanismos de protección adecuados frente a accesos no autorizados.

4.4 Restricciones técnicas

El desarrollo del sistema ha estado condicionado por una serie de restricciones técnicas y de alcance propias de un Trabajo Fin de Grado.

En primer lugar, la aplicación no contempla el control directo de maquinaria ni de sistemas de automatización industrial, centrándose exclusivamente en la gestión de la información asociada al almacén. Asimismo, el sistema ha sido diseñado específicamente para un entorno de almacén de consolidación, por lo que no resulta aplicable de forma directa a otros tipos de almacén con procesos logísticos diferentes.

Por último, se han considerado las limitaciones temporales y de recursos inherentes a un Trabajo Fin de Grado, desarrollado por un único autor y dentro de un periodo de tiempo acotado. Como consecuencia, se ha priorizado la implementación de las funcionalidades esenciales para la gestión del almacén de consolidación, quedando fuera del alcance del proyecto aspectos como la integración con sistemas externos, el control directo de equipos automatizados o el desarrollo de funcionalidades avanzadas de optimización.

5 Diseño del sistema

En este capítulo se describe el diseño del sistema desarrollado, detallando la estructura general del sistema, la estructura de sus distintos componentes y clases, asi como las decisiones adoptadas para implementar cada una de las funcionalidades. El objetivo de este capítulo es definir cómo se materializan, desde un punto de vista técnico, los requisitos identificados en la sección 4.3, sirviendo de base para la posterior fase de implementación.

5.1 Arquitectura general del sistema

El sistema ha sido diseñado siguiendo una arquitectura distribuida de tipo cliente-servidor, en la que los estaciones de trabajo fijas y telefonos móviles (clientes) interactúan con el servidor central.

La aplicación está compuesta por tres elementos principales: una aplicación de escritorio, una aplicación móvil Android y un servidor central MySQL que contiene la base de datos. Ambos clientes operan dentro de una red interna privada del almacén, lo que permite un entorno controlado desde el punto de vista de seguridad y acceso.

La aplicación de escritorio está destinada a la gestión y administración del sistema. En ella se podrá generar algunos fiheros necesarios para la realización de los envíos (Etiquetas de los palets) o bien para la gestion administrativa (información del sistema en formato excel) que podrán ser tratados posteriormente por otros sistemas externos.

La aplicación móvil Android se comunica con el servidor mediante una API de tipo REST, mientras que la aplicación de escritorio establece una conexión directa con la base de datos. Esta separación responde a criterios de simplicidad, rendimiento y adecuación al entorno operativo de cada dispositivo que empleamos como cliente.

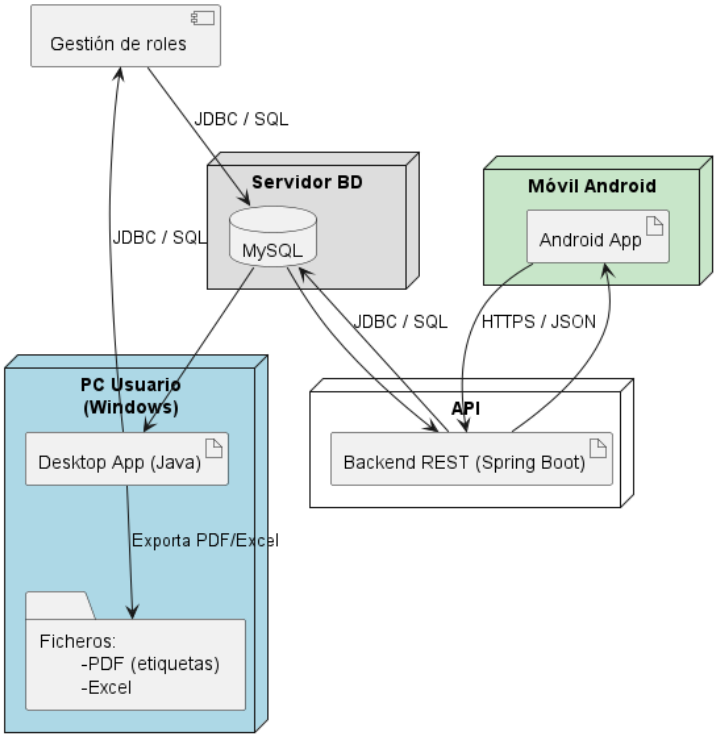


Figura 5: Arquitectura general del sistema cliente-servidor

5.2 Diseño de la aplicación Desktop

La aplicación de escritorio se ha diseñado en torno a una ventana principal que actúa como punto central de interacción con el sistema. Desde esta vista principal se proporciona acceso a las diferentes funcionalidades operativas y de gestión mediante. Las funcionalidades accesibles desde la ventana principal son las siguientes:

- **Almacén**: Representación visual 3D del almacén con filtros aplicables (pestaña por defecto).
- **Inventario**: Consulta del inventario de palets.
- **Ver pedidos**: Asignación de pedidos a los diferentes usuarios.
- **Paletizar**: Ejecución de los pedidos en curso asignados a los usuarios.
- **Envíos**: Gestión de envíos de pedidos.
- **Calendario**: Visualización del calendario de operaciones y eventos.

Determinadas acciones se ejecutan directamente dentro de la ventana principal, mientras que otras se gestionan a través de ventanas secundarias específicas. Estas ventanas emergentes están orientadas a tareas concretas que requieren un flujo de trabajo independiente, como la generación de órdenes de compra o la administración de usuarios (creación, modificación y eliminación), entre otros. Esta organización permite mantener una interfaz clara y evitar la sobrecarga de información en la vista principal. Estas ventanas son:

- **Movimiento de palets**: Pertmite tanto transladar un palet de un aubicación a otra como registrar la entrada o salida de un palet del almacén.
- **Actualizar palets**: Modifica la cantidad de producto dentro de un palet existente, así como sus dimensiones físicas.
- **Orden compra**: Crea una solicitud de compra de productos en la base de datos.
- **Export**: Exporta la información del inventario o de los pedidos a un fichero Excel.
- **Crear pedido**: Genera un nuevo pedido en la base de datos.
- **Editar pedido**: Modifica un pedido existente.
- **Eliminar pedido**: Elimina un pedido existente.
- **Crear usuario**: Genera un nuevo usuario en la base de datos.
- **Editar usuario**: Modifica un usuario existente.
- **Eliminar usuario**: Elimina un usuario existente.
- **Crear tipo**: Crea un nuevo tipo de producto en la base de datos.
- **Crear producto**: Genera un nuevo producto en la base de datos, con la información por defecto proveniente del proveedor.
- **Enviar pedidos**: Selección del transportista para el envio.
- **Detalle pedidos**: Visualización detallada de los pedidos a realizar.

Desde el punto de vista estructural, la aplicación se ha diseñado siguiendo una separación clara de responsabilidades entre la capa de presentación y la capa de acceso a datos. La capa de presentación, implementada mediante JavaFX, se encarga de la gestión de la interfaz gráfica y de la interacción con el usuario, mientras que la capa de acceso a datos centraliza la comunicación con la base de datos mediante el uso del conector MySQL.

Nota: La aplicación de escritorio no realiza compras a proveedores externos, sino que se limita a la generación de órdenes de compra que deben ser gestionadas posteriormente por el departamento correspondiente.

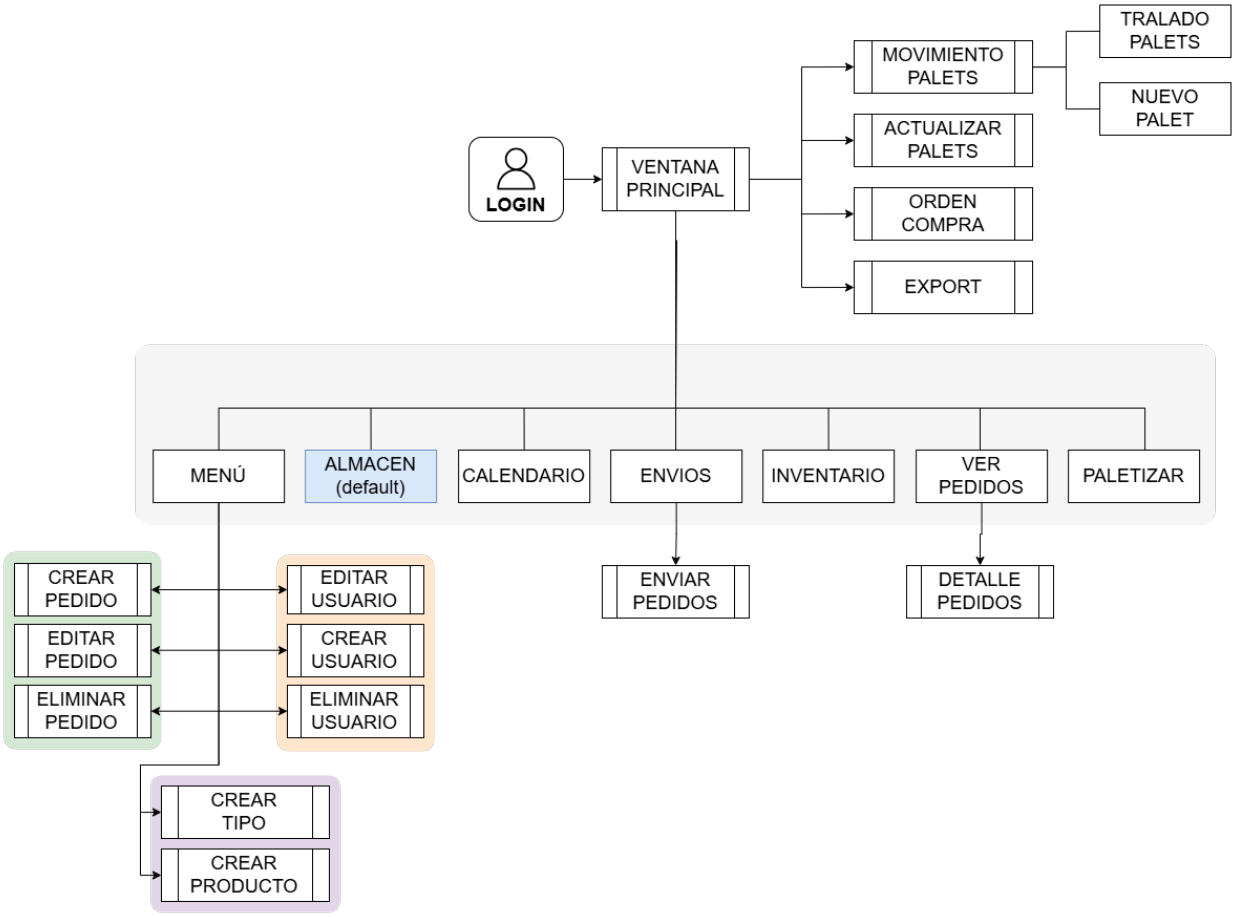


Figura 6: Diagrama de navegación entre ventanas de la aplicación Desktop

A continuación, en la Figura **??** se muestra un ejemplo de navegación entre ventanas de la aplicación Desktop, desde la ventana de *Login* hasta la ventana de *Orden Compras* pasando por la ventana principal *Almacén*.

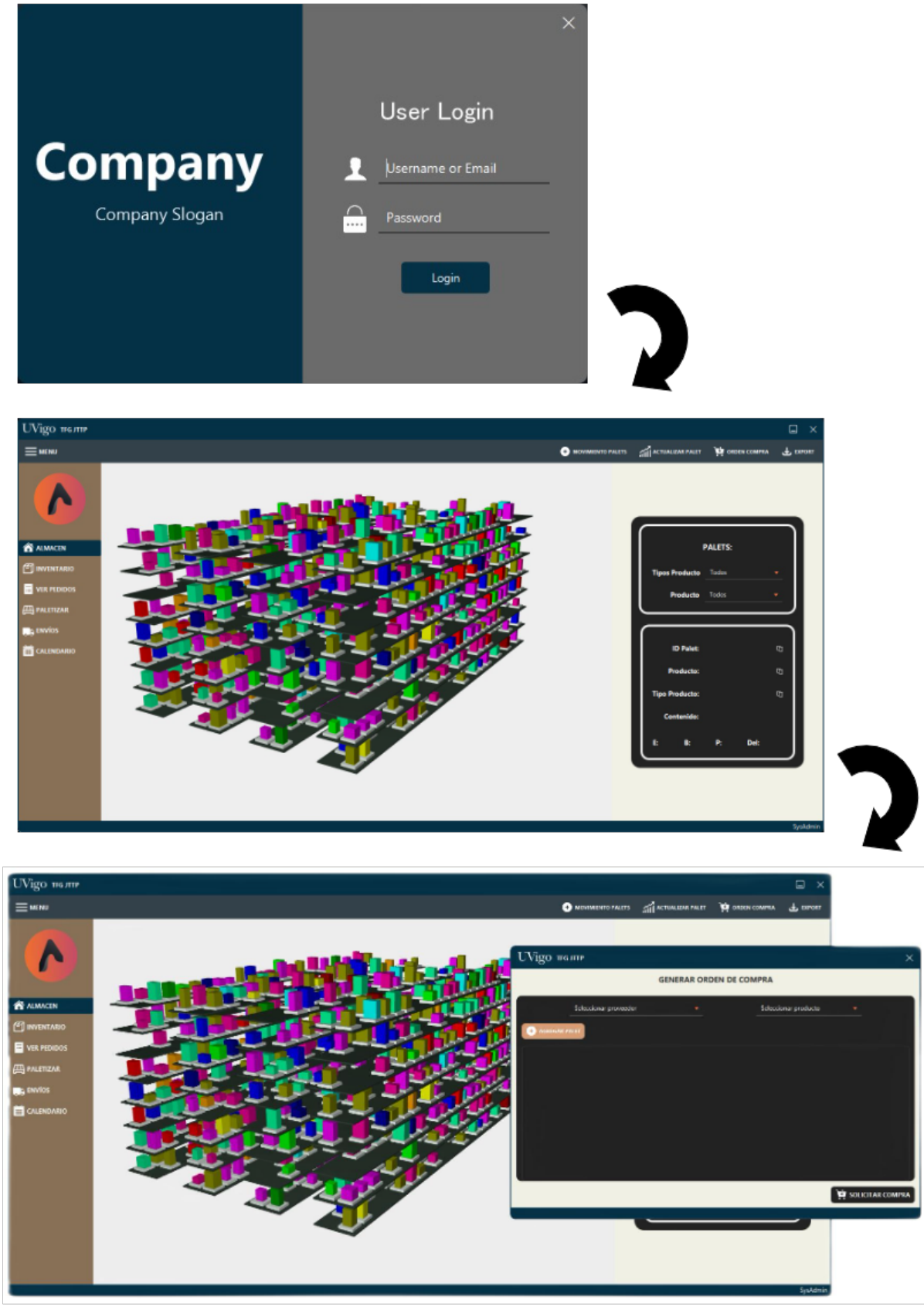


Figura 7: Ejemplo de navegación entre ventanas de la aplicación Desktop (desde *Login* hasta *Orden Compras*)

5.3 Diseño de la aplicación Android

La aplicación móvil Android ha sido diseñada como una herramienta de apoyo a la operativa en planta, orientada principalmente a los operarios del almacén. Su diseño prioriza la simplicidad de uso y el acceso rápido a la información relevante durante la ejecución de los pedidos por parte de los operarios.

La aplicación actúa como un cliente que delega la lógica de negocio en el servidor, comunicándose con este mediante una API REST. De este modo, la aplicación móvil se limita a gestionar la interfaz de usuario y el envío y recepción de datos.

La función de esta aplicación es la de permitir a los operarios ejecutar movimientos de palets y apoyar en la ejecución de pedidos, todo ello desde dispositivos móviles que facilitan su uso en el entorno del almacén. En particular, esta aplicación es un subapartado de la aplicación de escritorio combinando las funcionalidades de las pestañas de *Paletizar* y *Actualizar de palets*, permitiendo asi la correcta ejecución de los pedidos.

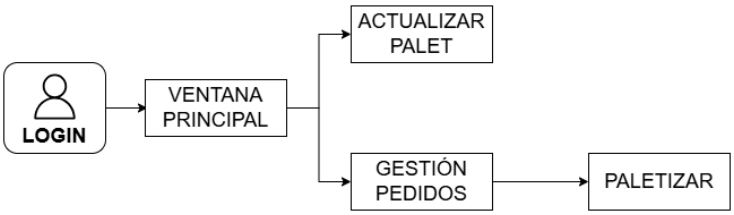


Figura 8: Diagrama de navegación entre ventanas de la aplicación Android

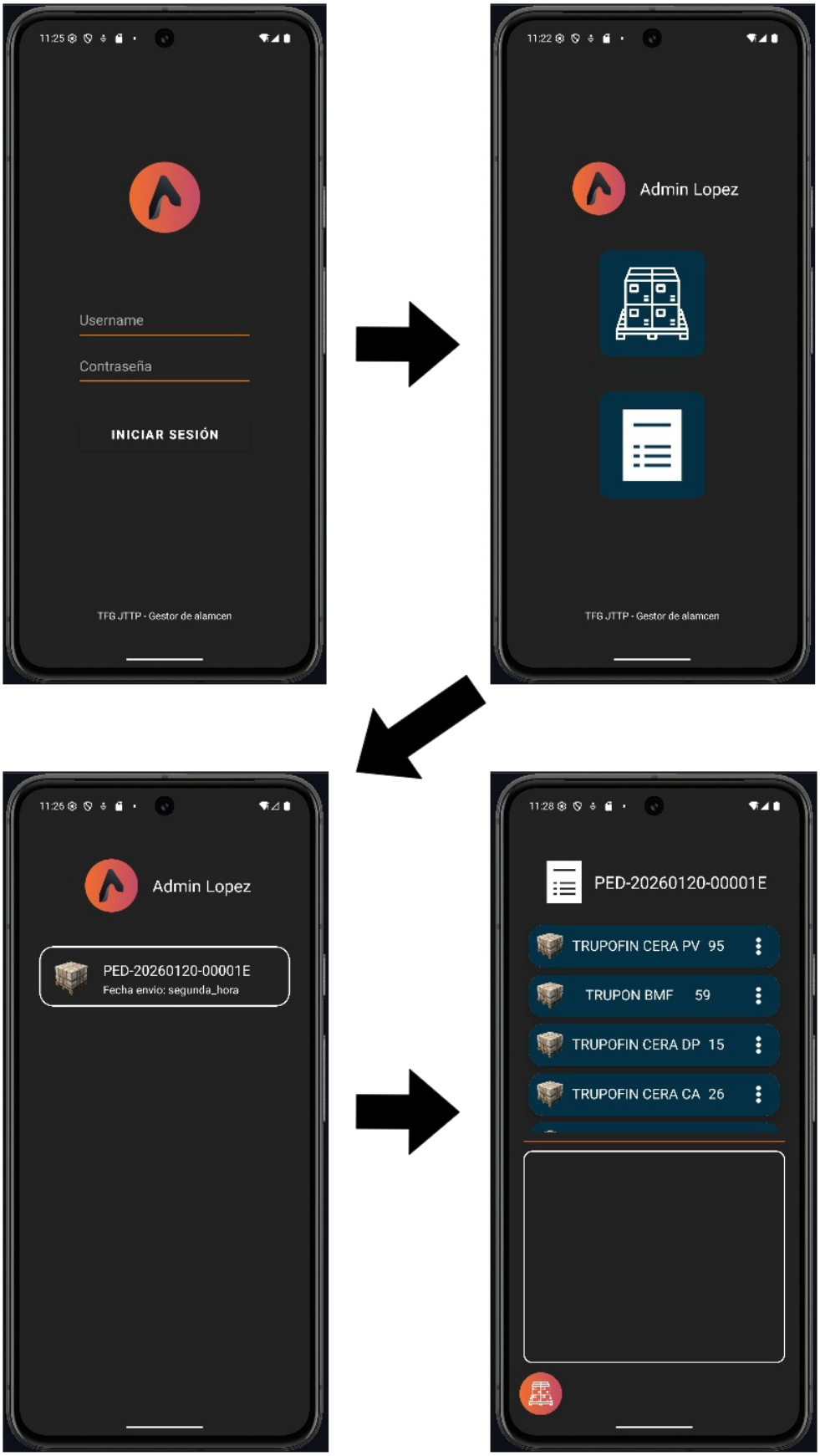


Figura 9: Ejemplo de navegación entre ventanas de la aplicación Android (desde *Login* hasta *Paletizar*)

5.4 Diseño de la base de datos

La base de datos constituye el núcleo de persistencia del sistema y ha sido diseñada para dar soporte directo a las operaciones propias de un almacén de consolidación. El modelo de datos representa de forma explícita los principales elementos del dominio, incluyendo productos, tipos de producto, palets, ubicaciones, pedidos, detalles del pedido, usuarios y roles, entre otros.

El diseño se ha basado en un modelo relacional normalizado (Base de datos relacional). Cada entidad se implementa mediante una tabla independiente identificada por una clave primaria, lo que permite un acceso unívoco a los registros y facilita su referenciación desde otras tablas. Las relaciones entre entidades se modelan mediante claves foráneas, reflejando dependencias como la asignación de palets a ubicaciones concretas, la composición de los pedidos mediante líneas asociadas a productos o el registro de movimientos vinculados a palets específicos. Este enfoque garantiza la integridad referencial y evita la duplicación innecesaria de información.

A nivel de esquema se han definido restricciones explícitas para reforzar la coherencia de los datos almacenados. En primer lugar, se han establecido campos obligatorios (`NOT NULL`) en atributos críticos del sistema, como el código SKU y la descripción de los productos, la ubicación asociada a cada palet, el estado de los pedidos o el identificador del usuario responsable de determinadas operaciones. Estas restricciones impiden la inserción de registros incompletos que podrían comprometer la operativa del almacén.

Asimismo, se han aplicado restricciones de unicidad (`UNIQUE`) en aquellos atributos que deben identificar de forma inequívoca una entidad dentro del sistema. Entre estos se incluyen el nombre de usuario utilizado para la autenticación, el código SKU de los productos y determinados identificadores internos de pedidos y palets. De este modo se evitan duplicidades que podrían generar inconsistencias, errores en los procesos de gestión o ambigüedad en la identificación de los elementos del almacén.

Las relaciones de dependencia entre entidades se controlan mediante claves foráneas (`FOREIGN KEY`), asegurando, por ejemplo, que un palet solo pueda estar asociado a una ubicación existente, que las líneas de pedido referencien productos previamente registrados o que los movimientos de almacén correspondan a palets válidos. Estas restricciones impiden la eliminación o modificación de registros que estén siendo utilizados por otras tablas, evitando la aparición de estados inconsistentes dentro de la base de datos.

En lo relativo a la seguridad, el diseño de la base de datos contempla específicamente la protección de las credenciales de los usuarios. Las contraseñas no se almacenan en texto claro, sino como valores derivados mediante un algoritmo de hash criptográfico. Para ello se ha empleado Argon2, un algoritmo de derivación de claves diseñado específicamente para el almacenamiento seguro de contraseñas. Este algoritmo permite configurar parámetros de coste en memoria y tiempo de cómputo, proporcionando resistencia frente a ataques de fuerza bruta y ataques basados en hardware especializado, y alineando el sistema con las buenas prácticas de seguridad en aplicaciones de gestión.

El script completo de creación del esquema se incluye en el Anexo 9.1. Para entender bien los distintos símbolos que aparecen en los campos de las tablas y las lineas que representan las relaciones entre las tablas del diagrama EER se debe consultar el Anexo 9.2.

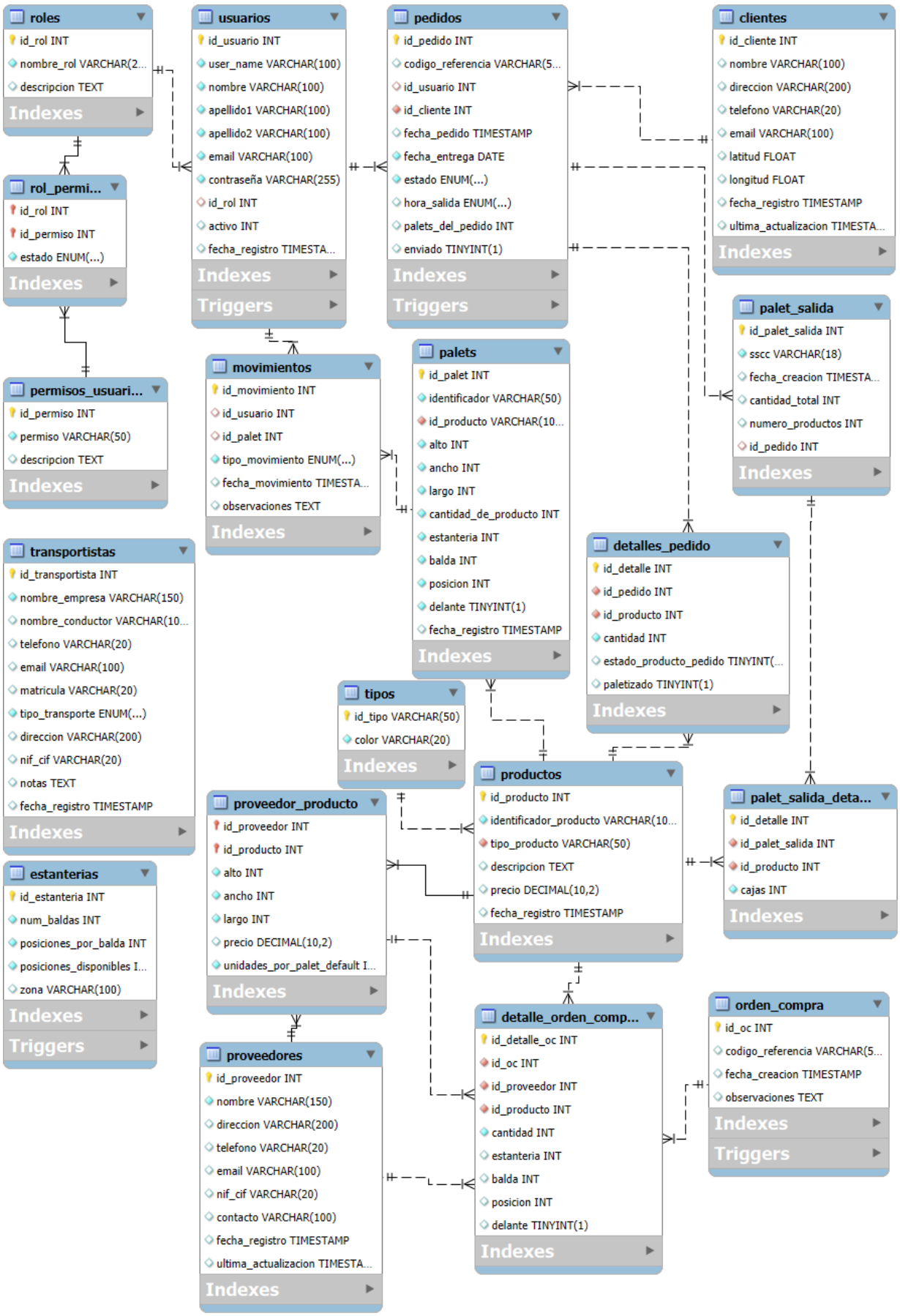


Figura 10: Diagrama Entidad-Relación (EER) del sistema de gestión de almacén

5.4.1 Gestión de usuarios, roles y permisos

Tabla `roles`. Define los perfiles de acceso del sistema. Su clave primaria es `id_rol`. Se referencia desde `usuarios` mediante `id_rol`.

Tabla `permisos_usuarios`. Catálogo de permisos/funcionalidades controladas por rol. La clave primaria es `id_permiso`. El campo `nombre_permiso` es `UNIQUE`, evitando duplicidades en el identificador lógico del permiso.

Tabla `rol_permiso`. Tabla de asociación rol–permiso. Implementa una relación N:M entre `roles` y `permisos_usuarios`. Su clave primaria es compuesta: `(id_rol, id_permiso)`. Incluye el campo `estado` (p. ej. `activo`, `inactivo`, `ver`) para modelar el nivel de acceso concedido por cada rol a cada permiso. Define dos claves foráneas: `id_rol` → `roles(id_rol)` y `id_permiso` → `permisos_usuarios(id_permiso)`.

Tabla `usuarios`. Almacena la identidad y credenciales de acceso. La clave primaria es `id_usuario`. Los campos `nombre`, `apellido1`, `apellido2` y `email` están definidos como `NOT NULL`; `email` es `UNIQUE`. El campo `contraseña` (`VARCHAR(255)`) almacena el *hash* de contraseña (Argon2), evitando el almacenamiento en texto plano. La relación con roles se implementa mediante `id_rol` como clave foránea a `roles(id_rol)` con `ON DELETE CASCADE` y `ON UPDATE CASCADE`. Incluye `fecha_registro` con `DEFAULT CURRENT_TIMESTAMP`.

Trigger `generar_user_name`. Trigger `BEFORE INSERT` sobre `usuarios`. Genera automáticamente `user_name` concatenando la inicial del nombre y fragmentos de apellidos, y añade un sufijo hexadecimal en función del número de coincidencias existentes; también inicializa `activo=1`. Este trigger evita dependencias del cliente para construir el identificador de usuario.

5.4.2 Catálogo de productos y proveedores

Tabla `tipos`. Catálogo de tipos de producto. Clave primaria `id_tipo`. El campo `nombre` es `UNIQUE`, imponiendo unicidad del tipo a nivel de esquema.

Tabla `proveedores`. Almacena proveedores. Clave primaria `id_proveedor`. El campo `nombre_proveedor` es `UNIQUE` para evitar duplicidades de proveedor en el catálogo.

Tabla `productos`. Define el maestro de producto. Clave primaria `id_producto`. Incluye `identificador_producto` como clave lógica `UNIQUE` (referencia interna/externa estable para integraciones y trazabilidad). `nombre` y `cantidad_por_palet` están marcados como `NOT NULL`. Relaciona el producto con su tipo mediante la clave foránea `id_tipo` → `tipos(id_tipo)`.

Tabla `proveedor_producto`. Materializa la relación N:M entre `proveedores` y `productos`, permitiendo registrar la asociación de un proveedor con un producto concreto. Clave primaria compuesta `(id_proveedor, id_producto)`. Define dos claves foráneas: `id_proveedor` → `proveedores(id_proveedor)` y `id_producto` → `productos(id_producto)`.

5.4.3 Estructura física del almacén, ubicaciones y palets

Tabla `estanterias`. Representa una estantería física. Clave primaria `id_estanteria`. Incluye `nombre_estanteria` y parámetros geométricos/capacidad: `num_baldas`, `num_posiciones_balda`, `posiciones_total`, `posiciones_disponibles`. El campo `nombre_estanteria` es `UNIQUE`.

Trigger `calcular_posiciones_disponibles`. Trigger `BEFORE INSERT` sobre `estanterias`. Calcula `posiciones_total` y `posiciones_disponibles` como producto de `num_baldas` por `num_posiciones_balda`. Con ello se evita inconsistencia entre capacidad declarada y capacidad derivada.

Tabla `ubicaciones`. Modela la ubicación física concreta dentro de una estantería. Clave primaria `id_ubicacion`. Incluye `balda`, `posicion` y `delante` (profundidad/posición frontal-trasera). Se asocia a una estantería mediante la clave foránea `id_estanteria` → `estanterias(id_estanteria)` con `ON DELETE CASCADE` y `ON UPDATE CASCADE`. Impone unicidad lógica de la coordenada física mediante `UNIQUE(id_estanteria, balda, posicion, delante)`, evitando crear dos ubicaciones idénticas dentro de la misma estantería.

Tabla `palets`. Entidad central de inventario: cada registro representa un palet identificado por `id_palet`. Contiene datos dimensionales (`alto`, `ancho`, `profundo`), `peso` y `cantidad_producto`. Se asocia a producto mediante `identificador_producto` como clave foránea a `productos(identificador_producto)` y a ubicación mediante `id_ubicacion` como clave foránea a `ubicaciones(id_ubicacion)`. La restricción `UNIQUE(id_ubicacion)` garantiza ocupación 1:1 (una ubicación no puede contener más de un palet simultáneamente en el modelo).

5.4.4 Pedidos, detalle y expedición

Tabla `clientes`. Datos del cliente destinatario de pedidos. Clave primaria `id_cliente`. Los campos `nif` y `nombre_cliente` están definidos como `NOT NULL`.

Tabla `transportistas`. Catálogo de transportistas. Clave primaria `id_transportista`. El campo `nombre_transportista` es `NOT NULL`.

Tabla `pedidos`. Cabecera de pedido. Clave primaria `id_pedido`. Incluye `fecha_creacion` con `DEFAULT CURRENT_TIMESTAMP` y el campo `estado` como `ENUM('pendiente','en_proceso','completado','enviado','cancelado')` para restringir estados válidos. Relaciona el pedido con: `usuarios(id_usuario)` (usuario responsable/asignado), `clientes(id_cliente)` (destinatario), `transportistas(id_transportista)` (expedición; admite `NULL` hasta asignación). Incluye `codigo_referencia` como identificador operativo `UNIQUE`.

Trigger `generar_codigo_referencia`. Trigger `BEFORE INSERT` sobre `pedidos`. Genera `codigo_referencia` con formato `YYYYMMDD-XXX`, donde `XXX` es un contador diario en hexadecimal (`LPAD(HEX( ... ),3,'0')`). Asegura unicidad por día sin depender del cliente.

Tabla `detalles_pedido`. Líneas de pedido. Clave primaria `id_detalle_pedido`. Define `cantidad` como `NOT NULL`. Implementa la composición del pedido mediante dos claves foráneas: `id_pedido` → `pedidos(id_pedido)` y `id_producto` → `productos(id_producto)`.

Tabla `palet_salida`. Tabla de enlace entre `palets` y `pedidos` para registrar qué palets se asocian a la expedición de un pedido. Clave primaria `id_palet_salida`. Define claves foráneas: `id_palet` → `palets(id_palet)` y `id_pedido` → `pedidos(id_pedido)`.

5.4.5 Órdenes de compra

Tabla `orden_compra`. Cabecera de orden de compra. Clave primaria `id_oc`. Incluye `fecha_creacion` con `DEFAULT CURRENT_TIMESTAMP`, `estado` como `ENUM (`pendiente',`tramitada',`cancelada')` y `codigo_oc` como identificador `UNIQUE`. Cabecera de orden de compra. Clave primaria `id_oc`. Incluye `fecha_creacion` con `DEFAULT CURRENT_TIMESTAMP`, `estado` como `ENUM('pendiente','tramitada','cancelada')` y `codigo_oc` como identificador `UNIQUE`. Asocia la orden con un proveedor mediante `id_proveedor` como clave foránea a `proveedores(id_proveedor)`.

Trigger `generar_codigo_orden_compra`. Trigger `BEFORE INSERT` sobre `orden_compra`. Genera `codigo_oc` con formato `OC-YYYYMMDD-XXX`, usando un contador diario en hexadecimal para garantizar unicidad operativa.

Tabla `detalle_orden_compra`. Líneas de una orden de compra. Clave primaria `id_detalle`. Contiene `cantidad` como `NOT NULL`. Define claves foráneas: `id_oc` → `orden_compra(id_oc)` y `id_producto` → `productos(id_producto)`.

5.5 Diseño de la API

La API del sistema se ha diseñado como una aplicación REST, actuando como capa intermedia entre la aplicación móvil Android y el sistema de persistencia. Su objetivo principal es exponer de forma controlada el modelo de datos del almacén y encapsular la lógica de negocio asociada a la gestión de palets, ubicaciones y demás elementos operativos.

Desde el punto de vista arquitectónico, la API sigue una estructura en capas claramente diferenciadas, compuesta por una capa de controladores, una capa de servicios y una capa de acceso a datos. Esta organización permite separar responsabilidades, reducir el acoplamiento entre componentes y facilitar tanto el mantenimiento como la evolución del sistema.

La **capa de presentación** está formada por controladores REST que exponen los distintos endpoints HTTP y gestionan la entrada y salida de datos en formato JSON. Estos controladores se encargan de validar las peticiones recibidas, delegar el procesamiento en la capa de servicios y devolver respuestas con códigos HTTP adecuados según el resultado de la operación, como respuestas de éxito, errores de validación, recursos no encontrados o conflictos de estado.

La **capa de servicio** concentra la lógica de negocio del sistema. En ella se implementan las reglas que gobiernan el funcionamiento del almacén, como la validación de ubicaciones disponibles, la prevención de colisiones entre palets o la coherencia de los movimientos registrados. Los servicios orquestan el acceso a los repositorios y gestionan operaciones que pueden implicar varias acciones sobre la base de datos, utilizando transacciones para garantizar la consistencia de los datos.

La **capa de acceso a datos** se apoya en Spring Data JPA y está compuesta por repositorios que extienden interfaces del tipo `JpaRepository`. Esta capa proporciona las operaciones básicas de persistencia y permite definir consultas adicionales mediante el uso de métodos derivados por nombre. Por ejemplo, se emplean métodos específicos para comprobar si una ubicación ya está ocupada por otro palet antes de permitir su creación o movimiento, evitando estados inconsistentes en la base de datos.

El modelo de datos expuesto por la API se basa en entidades JPA que representan los elementos principales del almacén, como palets, estanterías, baldas y ubicaciones. Para desacoplar la representación interna del sistema de la información intercambiada con los clientes, se utilizan objetos de transferencia de datos (DTOs). Estos DTOs permiten controlar los campos expuestos en cada operación y facilitan la validación de los datos recibidos.

La API hace uso de validaciones declarativas mediante anotaciones estándar, garantizando que los datos recibidos cumplan los requisitos definidos antes de ser procesados. Los errores y excepciones generados durante la ejecución se gestionan de forma centralizada, transformándolos en respuestas estructuradas que incluyen información clara sobre la causa del fallo.

En cuanto a la interacción con los telefonos Android, la API define endpoints REST semánticos que siguen las convenciones habituales del protocolo HTTP, incluyendo operaciones de consulta, creación, actualización y eliminación de recursos. Se emplean códigos de estado HTTP para reflejar correctamente el resultado de cada operación y, cuando es necesario, se admite el uso de paginación y filtrado mediante parámetros de consulta.

Este diseño permite que la API actúe como un componente independiente, reutilizable y fácilmente ampliable, proporcionando un punto de acceso controlado a la información del almacén y garantizando la coherencia de las operaciones realizadas desde la aplicación móvil Android.

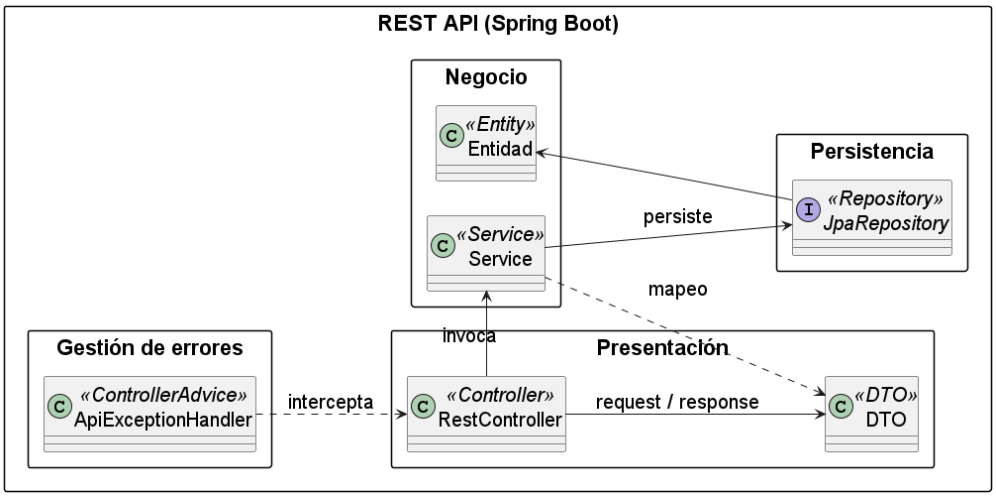


Figura 11: Arquitectura en capas de la API REST

5.6 Diagramas UML

Para representar de forma gráfica la estructura y el comportamiento del sistema, se han empleado distintos diagramas UML.

5.6.1 Diagrama de clases

El diagrama de clases representa las principales entidades del sistema, sus atributos y las relaciones existentes entre ellas. Este diagrama sirve como referencia para el diseño del modelo de datos y para la implementación de la lógica del sistema.

En la Figura 12 se representa el diagrama de clases correspondiente a la aplicación de escritorio. Este diagrama se centra en las entidades del dominio del almacén, como productos, palets, pedidos, movimientos y usuarios, así como en las relaciones que reflejan los procesos operativos del sistema.

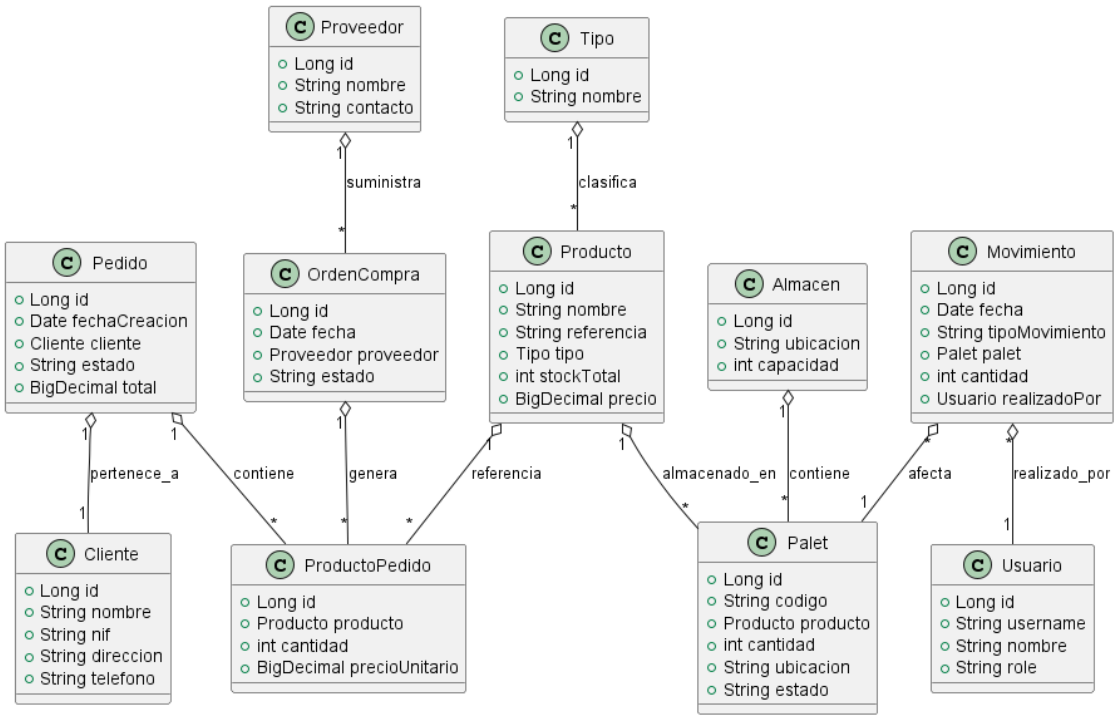


Figura 12: Diagrama de clases del sistema desktop

Por otra parte, la Figura 13 muestra una vista del diagrama de clases de la aplicación Android. Se muestra la estructura de la capa cliente, destacando las actividades principales de la interfaz, los clientes de comunicación con la API REST y los objetos de transferencia de datos (DTOs).

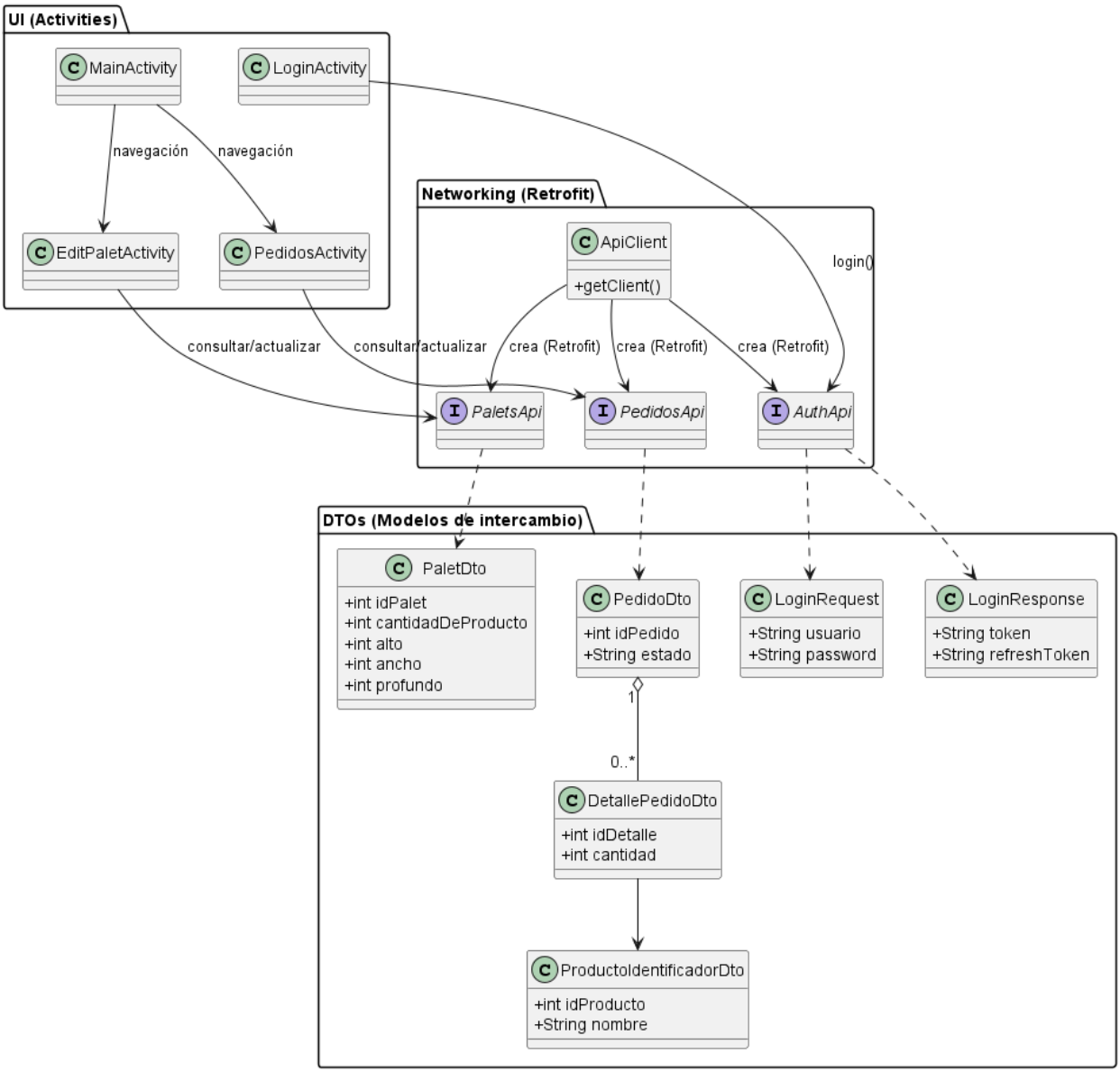


Figura 13: Diagrama de clases del sistema android

5.6.2 Diagrama de secuencia

Los diagramas de secuencia permiten representar de forma temporal y ordenada las interacciones entre los distintos componentes del sistema durante la ejecución de determinados casos de uso.

En este trabajo se han seleccionado algunos diagramas de secuencia representativos, que ilustran procesos clave del sistema, se reprsentan un porceso interno de la aplicación (login), el proceso principal de gestión de pedidos y un proceso interno del almacen de modificación de cantidad de palet.

El primer diagrama de secuencia describe el proceso de autenticación de usuarios, haciendo especial énfasis en el tratamiento seguro de las credenciales. En este flujo se representa cómo la aplicación cliente envía los datos de acceso, cómo el sistema verifica la contraseña mediante su

comparación con el valor almacenado en forma de hash utilizando el algoritmo Argon2, y cómo se gestiona la respuesta de autenticación en función del resultado de la validación.

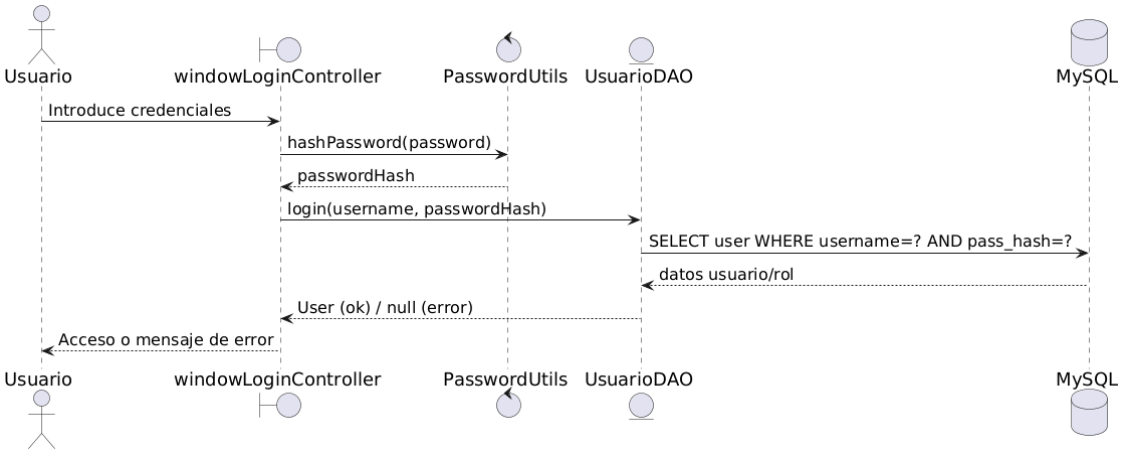


Figura 14: Diagrama de secuencia de inicio de sesión

El segundo diagrama de secuencia muestra el flujo completo asociado a la gestión de un pedido, desde su asignación inicial hasta su envío. Este diagrama refleja las interacciones entre los distintos componentes del sistema durante la consulta del pedido, su preparación, la actualización de su estado y la confirmación final del envío, proporcionando una visión clara del ciclo de vida de un pedido dentro del sistema.

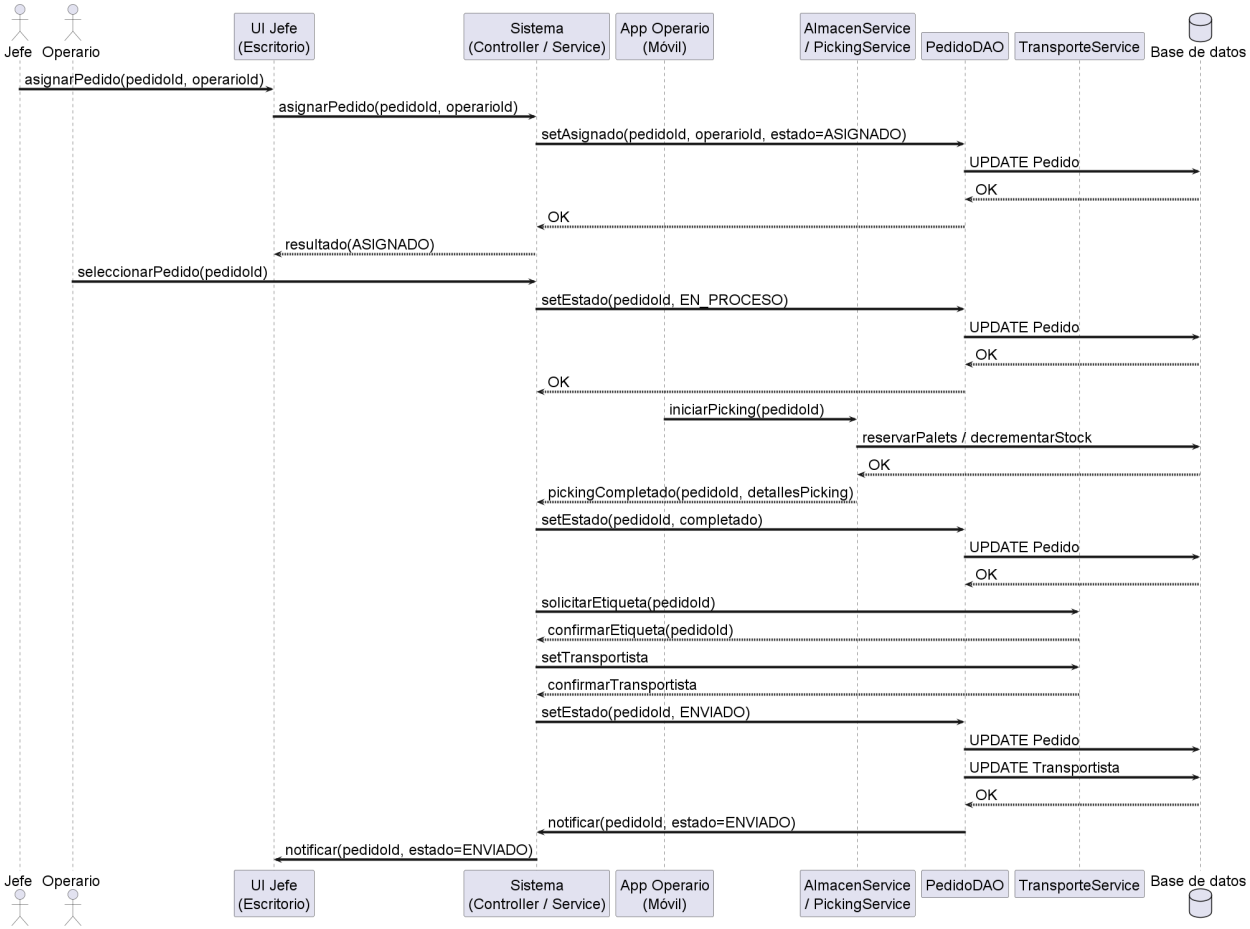


Figura 15: Diagrama de secuencia de gestión de pedidos

A continuación se presenta un diagrama de secuencia que describe el flujo de interacción necesario para modificar la cantidad de producto asociada a un palet existente.

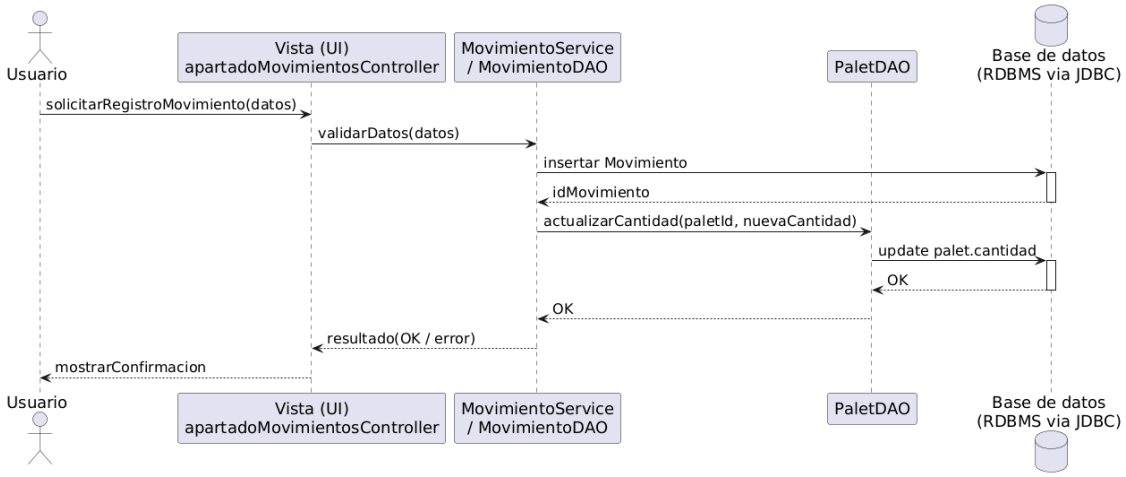


Figura 16: Diagrama de secuencia de modificación de cantidad de palet

5.6.3 Diagrama de despliegue

El diagrama de despliegue describe la distribución física de los componentes del sistema en tiempo de ejecución y su asignación a los distintos nodos de la infraestructura. En él se representan las estaciones de trabajo que ejecutan la aplicación de escritorio, los dispositivos móviles que ejecutan la aplicación Android, el servidor que aloja el backend y el servidor de base de datos. Asimismo, se especifican los enlaces de comunicación entre estos nodos, indicando el protocolo utilizado y el formato de los datos intercambiados.

En el sistema desarrollado se distinguen dos tipos de acceso a la información en función del tipo de cliente. Por un lado, la aplicación de escritorio, instalada en equipos ubicados dentro de la red interna del almacén, establece una conexión directa con el servidor de base de datos MySQL mediante JDBC. Por otro lado, la aplicación Android se comunica exclusivamente con el backend implementado con Spring Boot a través de una API REST sobre HTTPS, intercambiando mensajes en formato JSON. El backend actúa como intermediario entre la aplicación móvil y la base de datos, accediendo a esta última también mediante JDBC.

Este diagrama permite justificar decisiones arquitectónicas relevantes, como la separación del backend para los clientes móviles, el uso de distintos canales de comunicación según el entorno de ejecución y la limitación del acceso directo a la base de datos a un conjunto reducido de aplicaciones en un entorno controlado. Además, se reflejan elementos auxiliares del sistema, como la generación de ficheros de exportación y el envío de documentos que serán consumidos por dispositivos exrternos como impresoras..

Supuestos de red El sistema está diseñado para operar dentro de la red interna del almacén, sin necesidad de acceso desde redes externas.

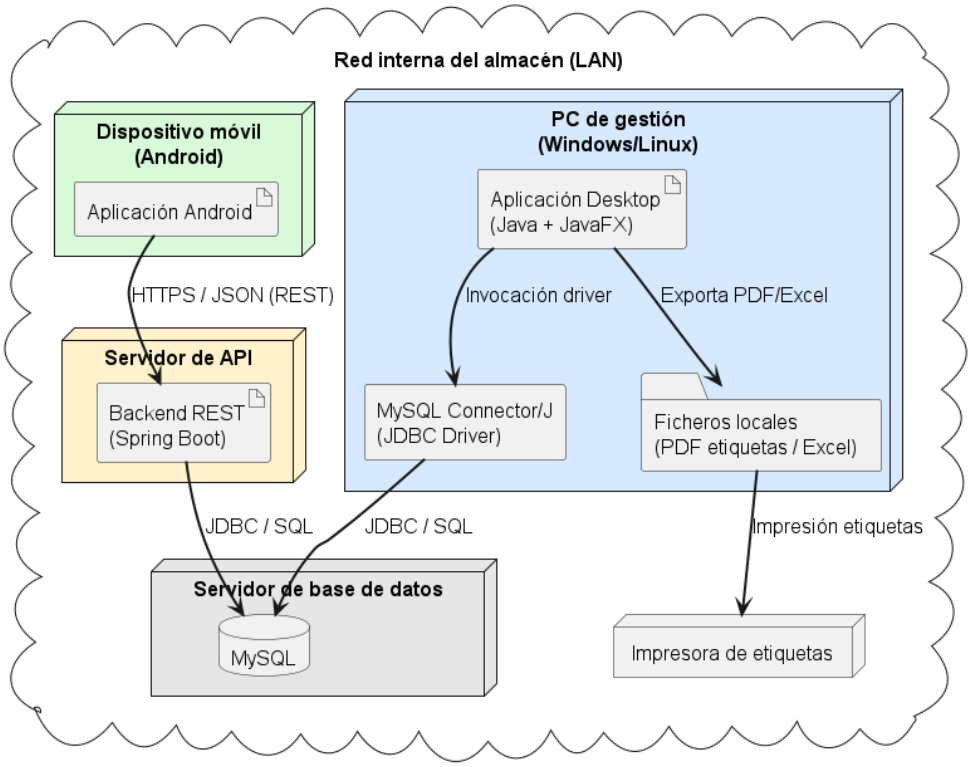


Figura 17: Diagrama de despliegue del sistema

5.7 Diseño de la interfaz de usuario

El diseño de la interfaz de usuario se ha definido a partir de los requisitos funcionales del sistema y de los distintos perfiles de usuario identificados en el análisis. El objetivo principal ha sido garantizar una interacción clara, consistente y alineada con el contexto operativo de un almacén de consolidación.

En la aplicación de escritorio, orientada a usuarios con funciones de supervisión y administración, la interfaz se estructura en torno a una ventana principal que centraliza el acceso a las funcionalidades del sistema. La disposición de menús y botones responde a una organización funcional, agrupando las operaciones relacionadas con inventario, pedidos, movimientos y administración. Se prioriza la visualización estructurada de grandes volúmenes de información mediante tablas, vistas detalladas y paneles específicos, permitiendo al usuario mantener una visión global del estado del almacén y acceder a operaciones avanzadas.

Las ventanas secundarias se utilizan para acciones que requieren un flujo de trabajo independiente, como la creación y edición de pedidos, la gestión de usuarios o la actualización de información de palets. Esta separación evita la sobrecarga de la vista principal y reduce el riesgo de errores derivados de la ejecución simultánea de tareas no relacionadas.

A continuación se muestra la ventana principal de la aplicación de escritorio en la Figura 18.

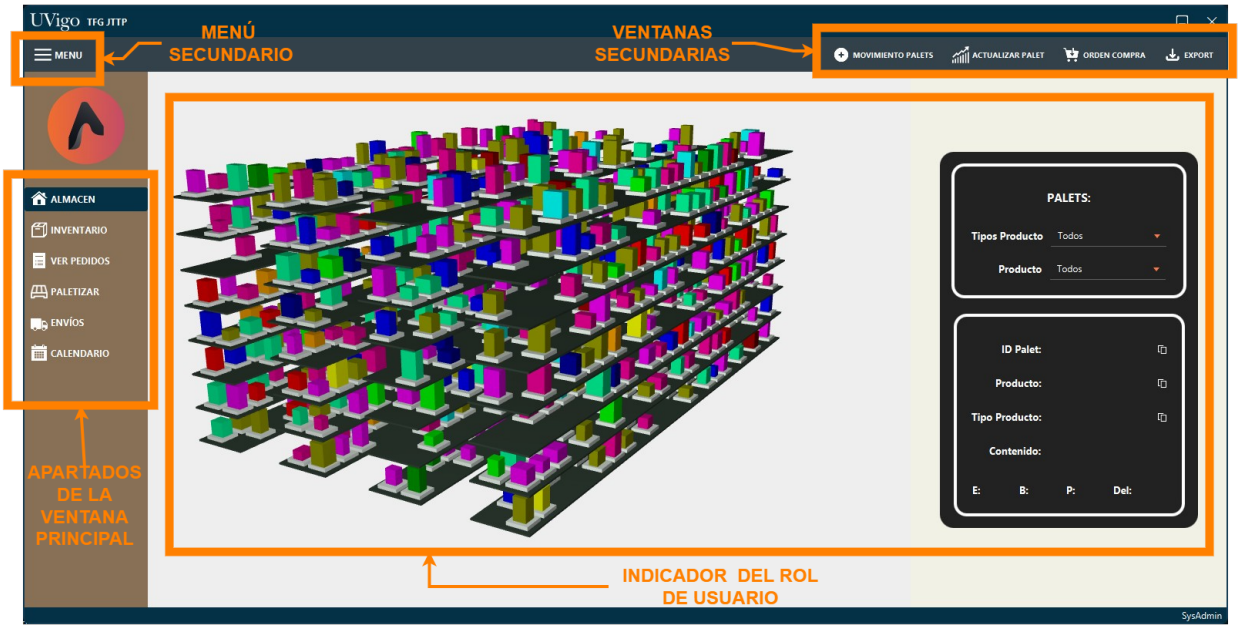


Figura 18: Ventana principal de la aplicación de escritorio

las Figura 19 y 20 muestran algunas de las ventanas secundarias de la aplicación de escritorio, mostrando las diversas de tareas que pueden gestionarse desde estas ventanas especializadas.

(a) Movimiento de palets

(b) Exportación de datos

Figura 19: Ventanas secundarias de la aplicación de escritorio

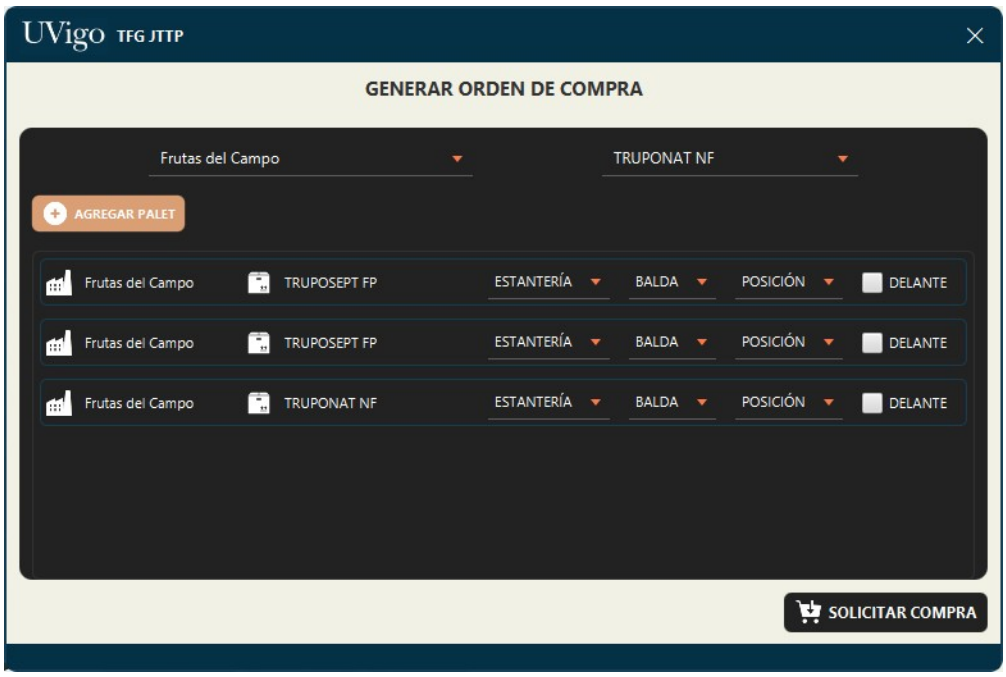
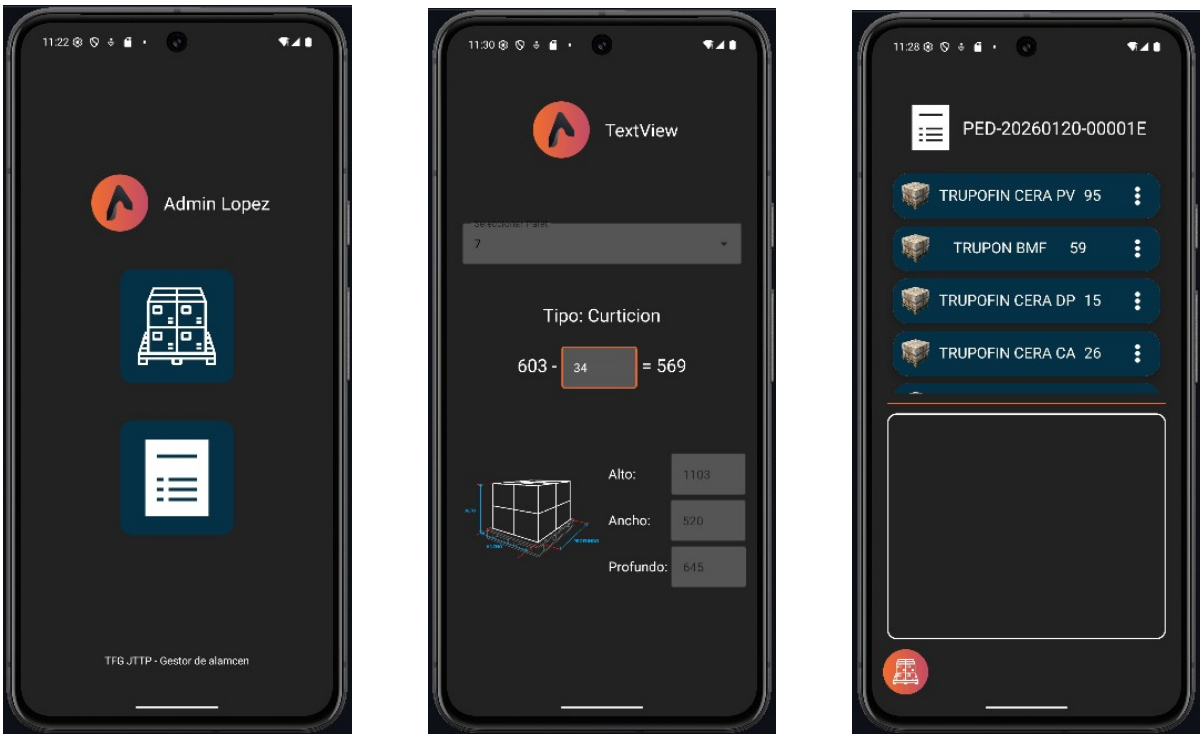


Figura 20: Ventana de orden de compra

En la aplicación móvil Android, destinada principalmente a operarios, el diseño de la interfaz se ha orientado a la ejecución rápida y directa de acciones en el entorno del almacén. La navegación se basa en pantallas específicas para cada tarea, con un número reducido de elementos visibles y controles intuitivos. Las operaciones más frecuentes, como la actualización de las cantidades de los palets que se están manipulando, la consulta de pedidos asignados o el registro del contenido de los palets generados para el envío, se han diseñado para completarse con el mínimo número de interacciones posibles.



(a) Ventana principal android (b) Actualizar palet (c) Paletización

Figura 21: Ventanas principales de la aplicación Android

6 Implementación

6.1 Herramientas y tecnologías utilizadas

El sistema desarrollado se ha implementado íntegramente utilizando el lenguaje de programación Java. Este lenguaje se ha empleado tanto en el desarrollo de la aplicación de escritorio como en la aplicación móvil Android y en el backend del sistema, implementado mediante el framework Spring Boot.

La utilización de un lenguaje común en todos los componentes de la solución facilita la integración entre las distintas aplicaciones, simplifica el mantenimiento del sistema y permite reutilizar conocimientos y conceptos a lo largo de todo el desarrollo. Además, Java es un lenguaje ampliamente utilizado en entornos industriales y empresariales, caracterizado por su portabilidad, robustez y amplio ecosistema de herramientas y librerías **oracle_java**.

- **Control de versiones**: Git.
- **Backend**: Java junto con el framework Spring Boot.
- **Base de datos**: MySQL.
- **Gestión y modelado de datos**: MySQL Workbench y Draw.io.
- **Desarrollo de la aplicación de escritorio**: IntelliJ IDEA Ultimate.
- **Desarrollo de la aplicación móvil**: Android Studio.
- **Pruebas de la API**: Postman.
- **Entorno de despliegue**: VMware Workstation con un servidor Ubuntu.
- **Gestión del proyecto**: Git.

6.2 Implementación de la aplicación Desktop

La aplicación de escritorio se ha implementado en Java utilizando JavaFX para la construcción de la interfaz gráfica y JDBC, mediante el conector oficial *MySQL Connector*, para la comunicación directa con la base de datos relacional.

La estructura del proyecto se organiza en paquetes diferenciados según su función. La capa de interfaz de usuario agrupa los controladores JavaFX asociados a cada ventana o diálogo, encargados de gestionar los eventos de usuario y la actualización de la vista. La capa de acceso a datos encapsula las consultas Lenguaje de Consulta Estructurado (SQL) mediante clases DAO, responsables de ejecutar consultas parametrizadas y gestionar los resultados obtenidos a través de ResultSet. Esta separación permite aislar la lógica de persistencia de la interfaz gráfica y facilita el mantenimiento del código.

La ventana principal constituye el punto de acceso a las funcionalidades del sistema y centraliza la navegación entre los distintos módulos de la aplicación, tales como la gestión del almacén, el inventario, los pedidos, la paletización, los envíos y el calendario de operaciones. Cada módulo se estructura a nivel de interfaz mediante ficheros FXML específicos.

Para evitar la sobrecarga de visual de la ventana principal, se utilizan ventanas secundarias para determinadas operaciones (orden de compra, exportacion de datos, entre otras), cargadas dinámicamente a partir de ficheros FXML independientes, así podemos realizar una interfaz gráfica más clara e intuitiva.

El control de acceso a funcionalidades se implementa en el cliente mediante comprobaciones del rol del usuario autenticado, habilitando o deshabilitando botones y opciones de menú en función de los permisos asignados. Las operaciones críticas se validan adicionalmente a nivel de base de datos, garantizando coherencia ante usos incorrectos de la interfaz, por ejemplo, la base de datos imposibilita introducir un palet en una posicion ya acupada o inexistente.

Finalmente, la aplicación incorpora mecanismos de exportación de datos a ficheros Excel y generación de documentos PDF para etiquetas y listados operativos, integrados directamente en el flujo de trabajo del usuario.

Las tecnologías empleadas en la implementación de la aplicación de escritorio, la Tabla 2 recoge las dependencias principales utilizadas, junto con su versión y su función dentro del proyecto.

Componente	Versión	Uso en el proyecto
JavaFX (controls, fxml, graphics)	23.0.1	Interfaz gráfica de la aplicación de escritorio (escenas, controles y carga de vistas FXML).
MySQL Connector	9.5.0	Conectividad JDBC con MySQL desde la aplicación desktop.
Argon2 JVM	2.11	Hash/verificación de contraseñas (almacenamiento seguro de credenciales).
Apache POI (poi, poi-ooxml)	5.5.0	Exportación de datos a ficheros Excel (XLSX).
Apache PDFBox	2.0.30	Generación/gestión de documentos PDF (p. ej., etiquetas o informes).
iText (itextpdf)	5.5.13.3	Generación de PDF (si se emplea en alguna parte del flujo documental).
ControlsFX	11.1.0	Controles UI adicionales para JavaFX (diálogos, validaciones visuales, etc.).
Ikonli (core, javafx, fontawesome-pack)	12.2.0	Iconografía en la interfaz JavaFX.
CalendarFX (view)	11.10.1	Vista de calendario para planificación/consulta de operaciones.

Tabla 2: Dependencias principales empleadas en la aplicación de escritorio.

La Figura 22 muestra el árbol de directorios correspondiente a la lógica principal de la aplicación situado en el paquete “main”. En él se pueden observar los distintos paquetes que organizan las clases según su funcionalidad, tales como controladores de la intefaz gráfica *controller*, modelos, acceso a datos y utilidades.

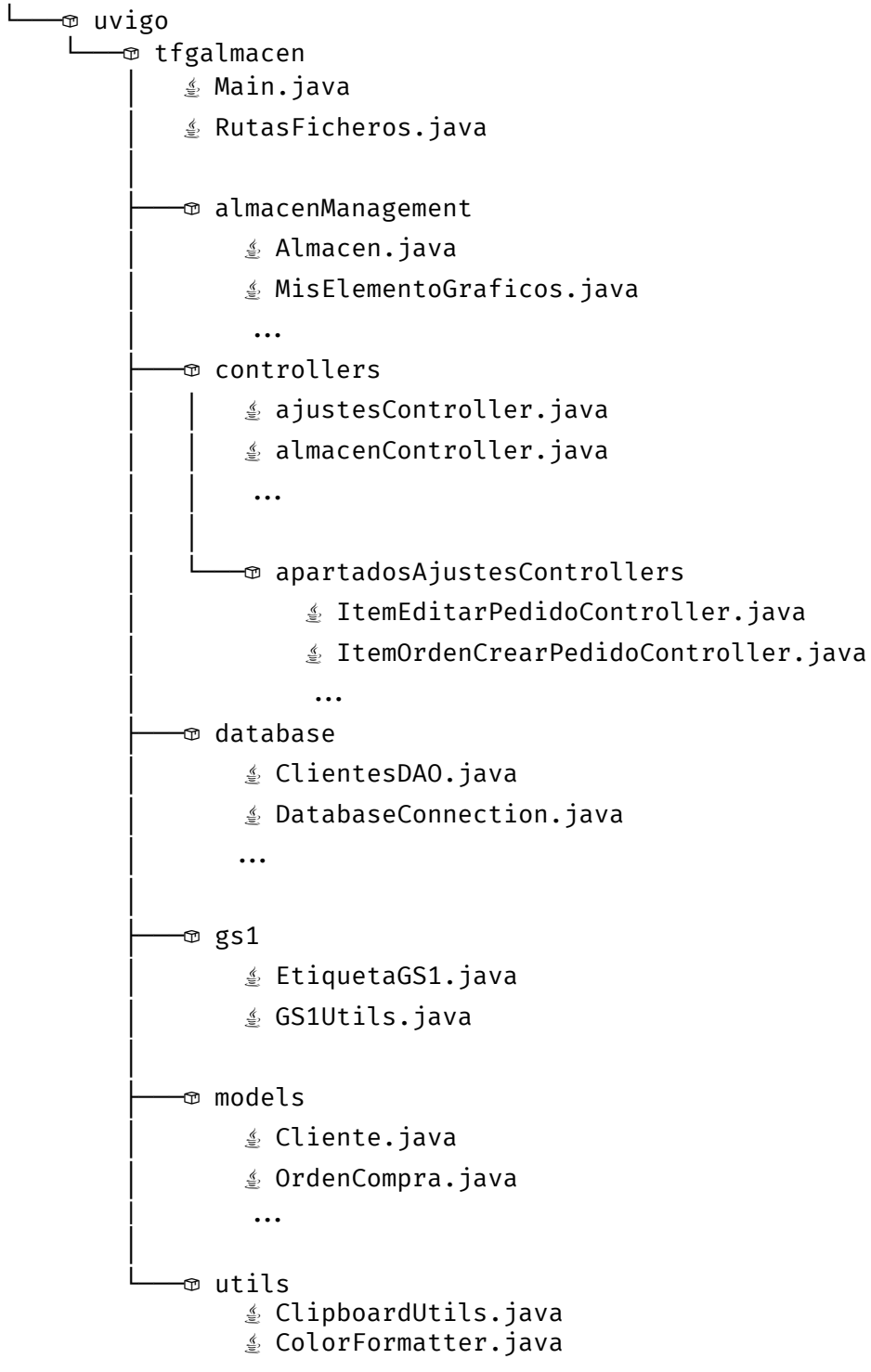


Figura 22: Árbol de directorios de la lógica de la app desktop

Por otra parte, los recursos de interfaz se organizan en un paquete específico “resources” que agrupa los ficheros FXML, junto con subdirectorios destinados a iconos e imágenes utilizados por las distintas vistas.

La Figura 23 muestra el árbol de directorios correspondiente a la interfaz gráfica de la aplicación, aquí tenemos todos los recursos gráficos como iconos, imagenes, que se emplearon en el diseño de la interfaz gráfica.

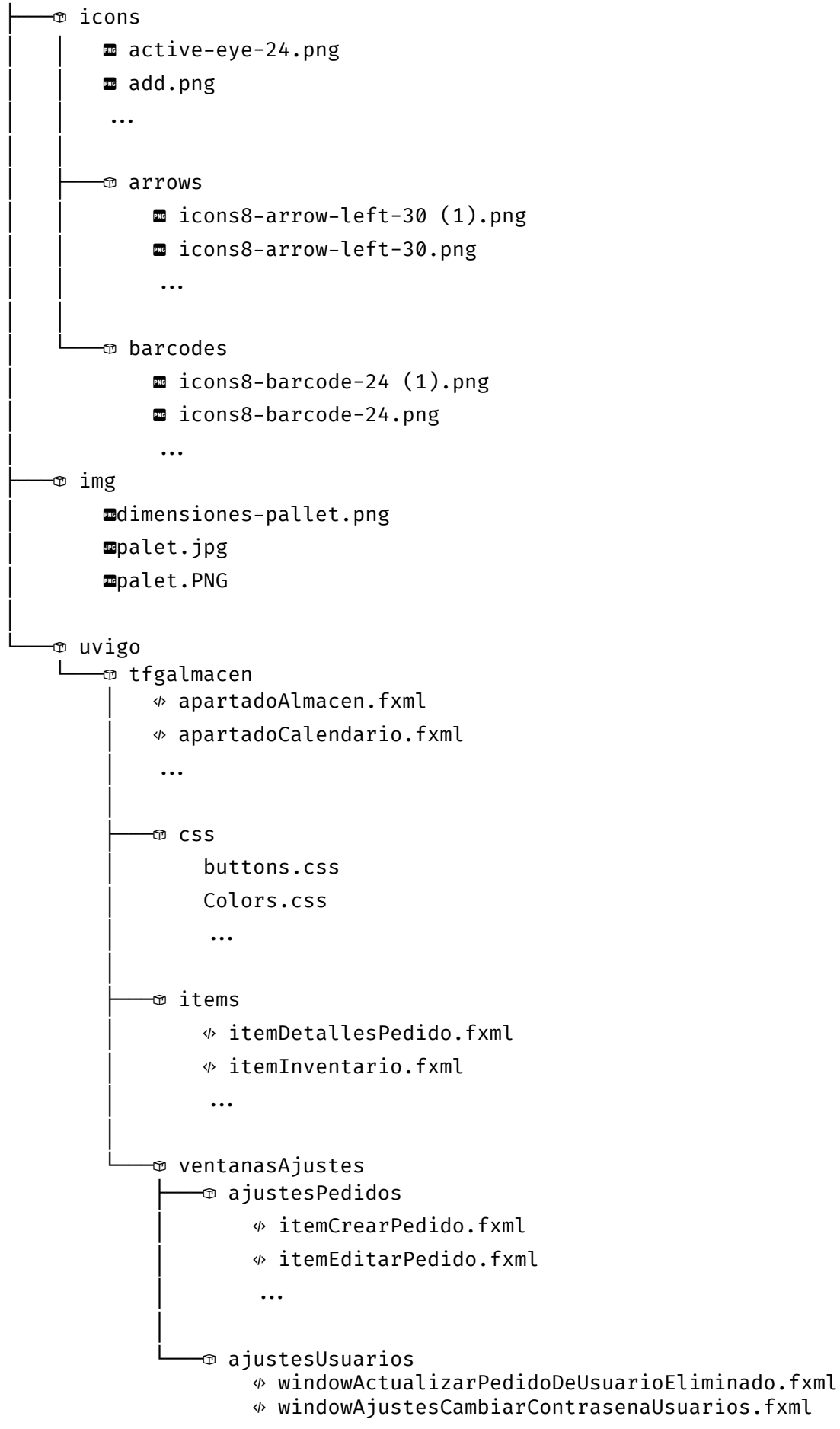


Figura 23: Árbol de directorios de la interfaz gráfica

6.3 Implementación de la aplicación Android

La aplicación Android se ha desarrollado siguiendo el modelo de cliente ligero, delegando la lógica de negocio y la persistencia de datos en el backend del sistema. La implementación se basa en el uso de actividades para la gestión de las distintas pantallas y en el consumo de servicios REST mediante la librería Retrofit.

Las actividades principales gestionan flujos bien definidos, como la autenticación del usuario, la consulta de pedidos asignados y la modificación de información asociada a palets. El intercambio de información con el backend se realiza exclusivamente mediante objetos DTO, que representan contratos de datos desacoplados del modelo interno del servidor.

La autenticación se implementa mediante el envío de credenciales al backend, que valida el acceso y devuelve un token de sesión utilizado en las peticiones posteriores. Este token se gestiona en memoria durante la ejecución de la aplicación, evitando el almacenamiento persistente de información sensible en el dispositivo.

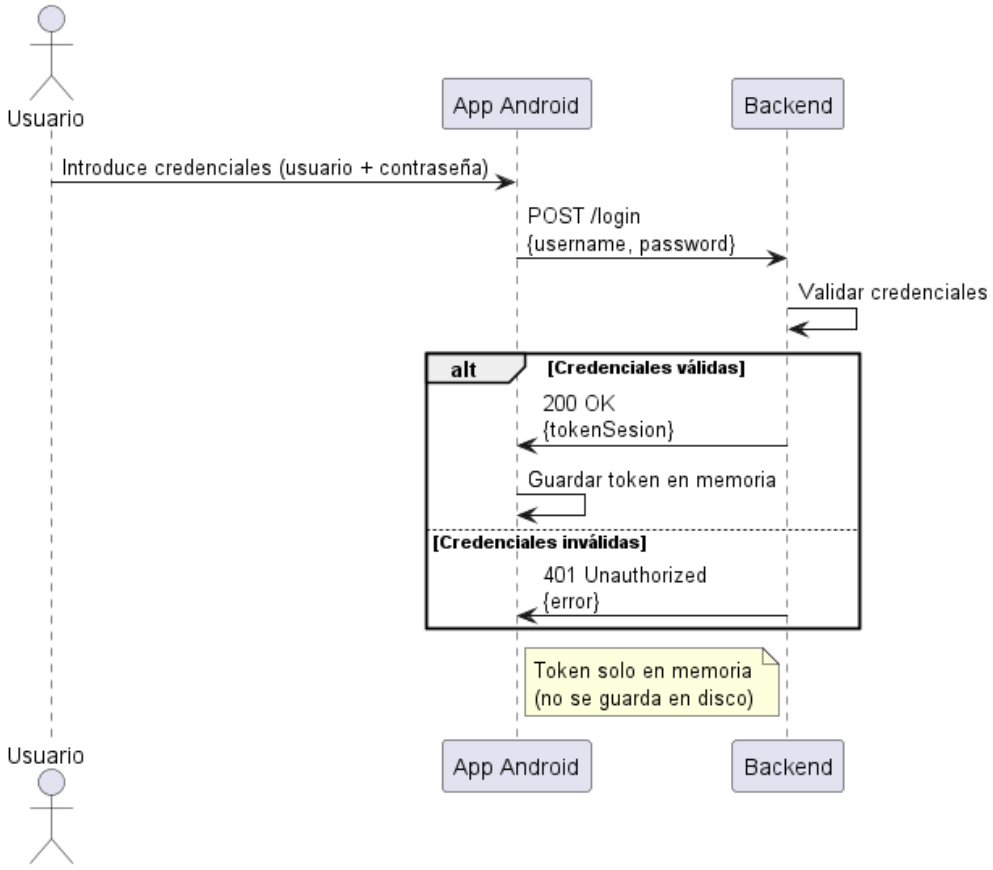


Figura 24: Secuencia de autenticación en la aplicación Android

El diseño de la interfaz prioriza la reducción de interacciones necesarias para completar una tarea. Las acciones más frecuentes, como actualización de cantidades, se implementan mediante formularios simplificados y selección asistida de datos, adaptados al uso en entornos operativos.

La aplicación Android restringe su funcionalidad al rol de operario, garantizando que únicamente se expongan operaciones relacionadas con la ejecución de pedidos y la gestión física de palets.

Las tecnologías empleadas en la implementación de la aplicación Android, la Tabla 3 recoge las dependencias principales utilizadas, junto con su versión y su función dentro del proyecto.

Componente	Versión	Uso en el proyecto
AndroidX AppCompat	1.7.1	Compatibilidad con versiones modernas de Android y soporte de componentes base de interfaz.
Material Components	1.13.0	Implementación de elementos de interfaz siguiendo las guías de diseño Material Design.
AndroidX Activity	1.12.0	Gestión del ciclo de vida de las *activities* y comunicación con la interfaz.
ConstraintLayout	2.2.1	Diseño flexible y eficiente de las pantallas mediante restricciones entre componentes.
Retrofit	3.0.0	Cliente HTTP para el consumo de la API REST desde la aplicación Android.
Retrofit Gson Converter	3.0.0	Serialización y deserialización automática de mensajes JSON intercambiados con la API.

Tabla 3: Dependencias principales y versiones de la aplicación Android.

El código se organiza en cuatro bloques: activities" controla la interfaz y el flujo inicial, incluyendo validaciones y acciones; api" gestiona la comunicación externa con Retrofit para enviar y recibir datos de palets y pedidos; models" define los DTO para estructurar la información; y adapters" convierte esos modelos en elementos de las listas dinámicas de la interfaz, cerrando el ciclo de visualización y respuesta a eventos. La Figura 25 muestra el árbol de directorios de la aplicación.

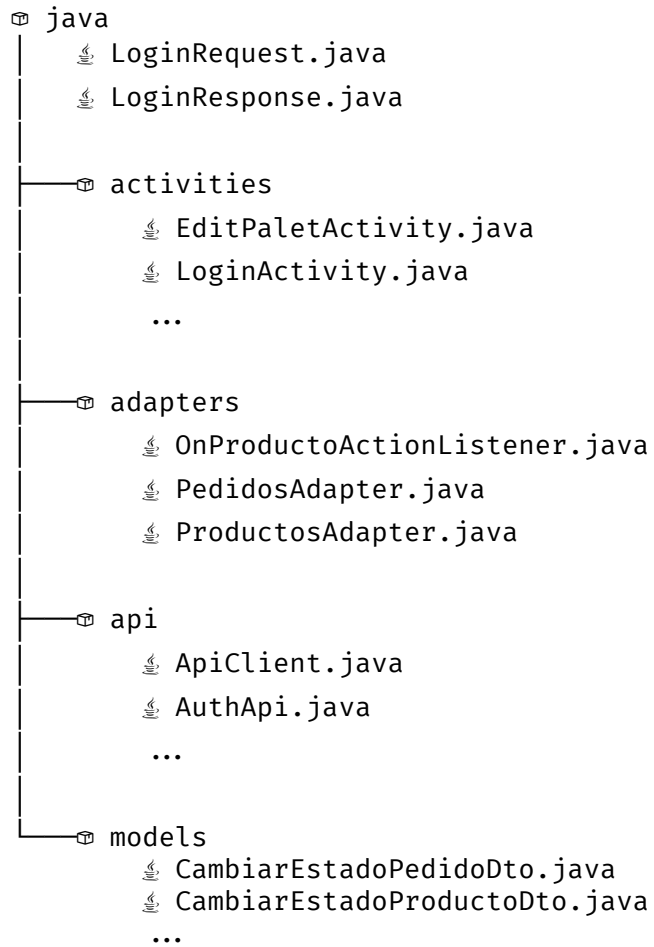


Figura 25: Árbol de directorios del paquete de la app Android

Para la intefaz gráfica, se utilizan ficheros XML que definen la estructura visual de cada pantalla, así como los diferentes iconos e imagenes implementadas en la aplicación. La Figura 26 muestra el árbol de directorios correspondiente a los recursos gráficos de la aplicación Android.

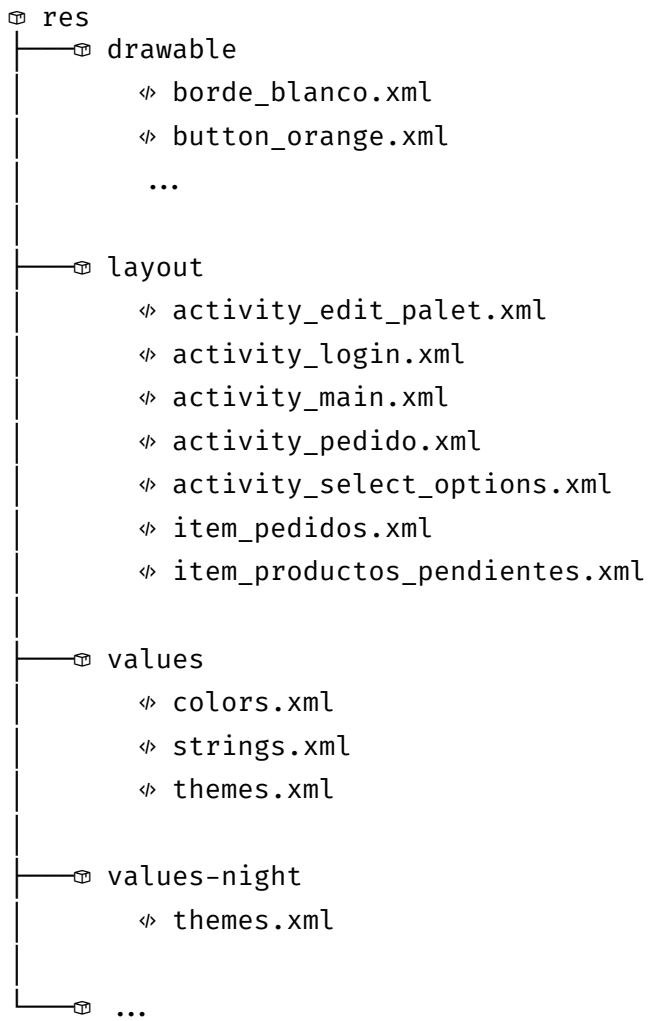


Figura 26: Árbol de directorios de los recursos gráficos de la app Android

6.4 Implementación del backend / servidor

El backend del sistema se ha desarrollado como una API REST basada en Spring Boot. Esta tecnologia ha sido seleccionada por ser el framework más popular en el ecosistema Java, ofreciendo una amplia gama de funcionalidades integradas y una comunidad activa que facilita la resolución de problemas y la incorporación de buenas prácticas.

La API Spring Boot desarrollada está estructurada en capas bien definidas que separan la exposición de servicios, la lógica de negocio y el acceso a datos. Esta organización facilita el mantenimiento del código, la evolución funcional y la validación de reglas de negocio de forma centralizada.

La capa de controladores, define los endpoints HTTP de la API y actúa como punto de entrada para las peticiones de los clientes. En esta capa se realiza la validación inicial de los datos recibidos y la conversión entre los objetos de transferencia (DTOs) y el modelo interno del sistema. Asimismo, los controladores construyen las respuestas HTTP, devolviendo códigos de estado coherentes con el resultado de cada operación (por ejemplo, éxito, error de validación o recurso no encontrado).

La lógica de negocio se concentra en la capa de servicios, donde se implementan las reglas

funcionales del sistema. Esta capa coordina las operaciones sobre los repositorios y garantiza la coherencia del dominio, aplicando comprobaciones como la validez del estado de un pedido antes de su modificación o la detección de conflictos en la asignación de ubicaciones de palets. Las operaciones que implican cambios persistentes se ejecutan dentro de transacciones para asegurar la consistencia de los datos ante fallos o accesos concurrentes.

El acceso a la base de datos se realiza mediante Spring Data JPA. Este enfoque permite disponer de operaciones CRUD estándar y de consultas derivadas por nombre o personalizadas, empleadas para validar condiciones específicas, como la existencia previa de registros o la verificación de restricciones operativas antes de una actualización.

El backend incorpora también mecanismos de seguridad y control de errores. La autenticación de usuarios se apoya en credenciales almacenadas de forma segura y en la validación centralizada de accesos a los endpoints. El tratamiento de excepciones se unifica mediante un componente global que traduce errores de dominio y validación en respuestas HTTP normalizadas, garantizando un comportamiento consistente frente a situaciones anómalas para la aplicación móvil.

En la implementación del backend se emplearon diversas tecnologías; la Tabla 4 recoge las dependencias principales utilizadas, junto con su versión y su función dentro del proyecto.

Componente	Versión	Uso en el proyecto
Framework backend	4.0.0	Estructura base del backend y gestión del ciclo de vida de la aplicación
API REST (Web MVC)	4.0.0	Exposición de endpoints HTTP y controladores REST
Persistencia (JPA)	4.0.0	Abstracción del acceso a datos mediante repositorios
ORM	7.1.8	Mapeo objeto-relacional entre entidades Java y tablas MySQL
Pool de conexiones JDBC	7.0.2	Gestión eficiente de conexiones a la base de datos
Driver MySQL	9.5.0	Comunicación JDBC entre la aplicación y MySQL
Hash de contraseñas (Argon2)	2.11	Almacenamiento seguro de credenciales de usuario
Servidor embebido	4.0.0	Ejecución autónoma de la API sin servidor externo
Servlet container (Tomcat)	11.0.14	Gestión de peticiones HTTP y ciclo de servlets

Tabla 4: Dependencias principales, versiones y uso en el backend (API REST).

6.5 Implementación de la base de datos

La base de datos del sistema se ha implementado sobre un sistema gestor relacional MySQL, seleccionado por su madurez, estabilidad y compatibilidad con entornos de producción y herramientas de desarrollo ampliamente utilizadas como *MySQL Workbench*. El esquema se define mediante scripts SQL que crean las tablas, claves primarias, claves foráneas y restricciones necesarias para garantizar la integridad de los datos. Ver el Anexo 9.1 para consultar el script completo de creación de la base de datos.

Las relaciones entre entidades se refuerzan mediante restricciones `FOREIGN KEY`, que controlan dependencias como la asociación entre palets y ubicaciones o entre pedidos y productos. Se han definido índices sobre campos de búsqueda frecuente, como identificadores de usuario, códigos de palet y estados de pedido, con el objetivo de optimizar las consultas más habituales.

También se han automatizado ciertas operaciones mediante `TRIGGERS` que garantizan la co-

herencia de datos y su trazabilidad. Uno de estos TRIGGERS lo implemntamos para la generación automática de códigos para nuevos pedidos. El siguiente código (Figura 27) muestra un trigger que asigna un código único a cada nuevo pedido insertado en la base de datos:

```
DELIMITER / / CREATE TRIGGER generar_codigo_referencia BEFORE INSERT ON
    pedidos FOR EACH ROW BEGIN DECLARE secuencia_diaria INT;

DECLARE secuencia_hex VARCHAR(10);

DECLARE nuevo_codigo VARCHAR(50);

SELECT
   COUNT(*) + 1 INTO secuencia_diaria
FROM
   pedidos
WHERE
   DATE (fecha_pedido) = DATE (NEW.fecha_pedido);

SET
   secuencia_hex = LPAD (HEX (secuencia_diaria), 6, '0');

SET
   nuevo_codigo = CONCAT (
      'PED-',
      DATE_FORMAT (NEW.fecha_pedido, '%Y%m%d'),
      '-',
      secuencia_hex
   );

SET
   NEW.codigo_referencia = nuevo_codigo;

END;

/ / DELIMITER;
```

Figura 27: Trigger para asignar código único a nuevos pedidos

La gestión de usuarios incluye el almacenamiento seguro de credenciales, la seguridad de las credenciales se detalla en la sección siguiente. Las contraseñas no se guardan en texto plano, sino como valores ilegibles. Esta implementación protege las credenciales frente a ataques de fuerza bruta y accesos no autorizados.

Para facilitar la puesta en marcha y las pruebas del sistema, el esquema incluye datos iniciales, como roles, permisos y configuraciones básicas del almacén (información obtenida del cliente), que permiten validar el funcionamiento completo de la aplicación desde las primeras fases de ejecución.

6.6 Seguridad y gestión de errores

El sistema incorpora mecanismos específicos para la protección de las credenciales de los usuarios y la gestión controlada de errores. Las contraseñas no se almacenan en texto claro en la base de datos, sino que se transforman mediante un algoritmo de derivación de claves diseñado para entornos de autenticación, evitando que la exposición de la base de datos implique la divulgación directa de credenciales.

Para este propósito se ha utilizado Argon2, un algoritmo moderno orientado al almacenamiento seguro de contraseñas. Este algoritmo permite incrementar el coste computacional del proceso de verificación, dificultando ataques de fuerza bruta. Su diseño prioriza el consumo controlado de recursos, lo que refuerza la seguridad sin afectar de forma significativa al uso normal del sistema.

La verificación de credenciales se realiza siempre comparando el valor derivado de la contraseña introducida con el hash almacenado, sin necesidad de revertir el proceso. De este modo, la lógica de autenticación mantiene una separación clara entre los datos sensibles y el resto de la aplicación, reduciendo la superficie de ataque y alineándose con buenas prácticas de seguridad en sistemas de información.

7 Resultados y discusión

7.1 Resultados del desarrollo

El desarrollo del sistema ha permitido materializar una solución funcional orientada a la gestión de un almacén de consolidación en un entorno industrial, dando respuesta directa a las necesidades operativas descritas en el capítulo de introducción. El sistema resultante integra una aplicación de escritorio, una aplicación móvil para dispositivos Android y un backend este último, todos ellos conectados mediante una base de datos relacional centralizada.

La aplicación de escritorio cubre las tareas de planificación, control y gestión estructurada del almacén. En ella se han implementado funcionalidades para la gestión de productos, palets, ubicaciones, pedidos y órdenes de compra. Esta aplicación actúa como herramienta principal del sistema, facilita una visión global y actualizada del estado del inventario y de los procesos de consolidación.

La aplicación móvil se orienta a la operativa en planta, permitiendo a los operarios consultar pedidos y actualizar cantidades de producto de forma inmediata durante la ejecución de tareas físicas. Este enfoque reduce el uso de registros manuales y minimiza el desfase entre la operación real y su reflejo en el sistema, uno de los problemas identificados en el análisis inicial.

El backend implementado en la aplicación móvil centraliza la lógica de negocio y el acceso a datos, garantizando la coherencia de la información compartida por ambas aplicaciones. Las validaciones implementadas, junto con las restricciones definidas a nivel de base de datos, evitan situaciones inconsistentes como la ocupación simultánea de una ubicación por varios palets o la modificación de pedidos en estados no permitidos.

En conjunto, el sistema desarrollado permite cubrir el flujo completo de trabajo de un almacén de consolidación, desde la entrada de mercancía y su ubicación física hasta la preparación y expedición de pedidos, manteniendo la trazabilidad de palets, ubicaciones, movimientos y usuarios.

7.2 Evaluación del cumplimiento de objetivos

Los resultados obtenidos permiten afirmar que el objetivo general del proyecto, consistente en desarrollar un sistema de gestión orientado a almacenes de consolidación en entornos industriales, ha sido alcanzado de forma satisfactoria.

En relación con los objetivos específicos definidos en el apartado 1.3, el sistema desarrollado responde a las necesidades operativas identificadas mediante la digitalización de los procesos clave del almacén. La arquitectura adoptada, basada en la comunicación entre aplicaciones cliente y un backend común, permite separar claramente la operativa en planta de las tareas de planificación y control, cumpliendo el objetivo de adaptar la solución a distintos perfiles de usuario.

La gestión de productos, palets, ubicaciones y pedidos se ha implementado de forma estructurada mediante un modelo de datos relacional coherente, reforzado por validaciones tanto en la lógica de negocio como en la base de datos. Este enfoque contribuye a mejorar la trazabilidad y el control del inventario, aspectos críticos señalados en la definición del problema.

Asimismo, el diseño de interfaces diferenciadas para escritorio y dispositivos móviles facilita la integración del sistema en la rutina diaria del almacén, reduciendo la complejidad percibida por el usuario final y favoreciendo su adopción en un entorno industrial real.

7.3 Limitaciones detectadas

A pesar de los resultados obtenidos, el sistema presenta limitaciones derivadas del alcance definido para el proyecto. En primer lugar, la solución se centra exclusivamente en el apoyo a la

gestión del almacén, sin intervenir en el control directo de maquinaria o sistemas automatizados, tal como se estableció en la definición del problema.

La operativa está pensada para un entorno de red interna controlada, por lo que aspectos como la alta disponibilidad, el acceso remoto o la escalabilidad a gran escala no se han abordado en profundidad. Del mismo modo, la aplicación móvil cubre las tareas operativas esenciales, pero no sustituye completamente a la aplicación de escritorio en labores de planificación o gestión avanzada.

8 Conclusiones y trabajos futuros

8.1 Conclusiones

El presente Trabajo Fin de Grado ha abordado el diseño y la implementación de un sistema software orientado a la gestión de almacenes de consolidación en entornos industriales. Partiendo del análisis de las necesidades operativas propias de este tipo de instalaciones, se ha desarrollado una solución técnica compuesta por una aplicación de escritorio, una aplicación móvil para dispositivos Android y un backend común encargado del tratamiento y persistencia de la información.

El sistema desarrollado permite gestionar de forma estructurada los elementos fundamentales del almacén —productos, palets, ubicaciones, movimientos y pedidos— proporcionando trazabilidad completa de las operaciones realizadas y reduciendo la dependencia de procedimientos manuales o herramientas genéricas no adaptadas a este contexto. La centralización de la información en una base de datos relacional, junto con las validaciones implementadas tanto a nivel de lógica de negocio como de esquema, contribuye a mejorar la coherencia y fiabilidad de los datos.

Desde el punto de vista arquitectónico, la separación entre planificación y operativa, materializada mediante la combinación de una aplicación de escritorio y una aplicación móvil, se ajusta a la realidad de los entornos industriales analizados. Este enfoque facilita que cada perfil de usuario disponga de una interfaz adaptada a su contexto de uso, mejorando la eficiencia durante la ejecución de tareas diarias en el almacén.

En el ámbito académico, el desarrollo del proyecto ha permitido aplicar de forma integrada conocimientos relacionados con arquitectura de software, diseño de bases de datos, desarrollo de interfaces gráficas y comunicación entre sistemas distribuidos. Asimismo, ha supuesto una aproximación práctica a problemas reales de gestión logística, reforzando la capacidad de análisis, diseño y toma de decisiones técnicas propias de la ingeniería industrial.

En conjunto, los objetivos planteados al inicio del proyecto han sido alcanzados, obteniendo un sistema funcional y coherente que puede servir como base para su aplicación o ampliación en entornos reales de almacenes de consolidación.

8.2 Líneas de mejora y trabajos futuros

A partir del trabajo realizado, se identifican diversas líneas de mejora y posibles extensiones que permitirían ampliar las capacidades del sistema y adaptarlo a escenarios más exigentes. Una primera línea de evolución consistiría en la integración con sistemas de automatización industrial o dispositivos de captura automática de datos, como lectores de códigos de barras, lo que permitiría reducir aún más la intervención manual, como la búsqueda por el codigo de identificación del palets, durante la operativa en planta.

Otra posible mejora se centra en la ampliación de la aplicación móvil, incorporando funcionalidades avanzadas de planificación o supervisión que actualmente se concentran en la aplicación de escritorio. Asimismo, podría evaluarse el desarrollo de una interfaz web adicional, dado que tenemos un backend implementado, que facilitase el acceso al sistema desde distintos dispositivos sin necesidad de instalación específica donde se puedan ejecutar operaciones de consulta u otras que convenga implementar.

En cuanto a la base de datos, futuras ampliaciones podrían incluir optimizaciones adicionales mediante particionado, replicación o análisis histórico de datos para la obtención de indicadores de rendimiento del almacén. Estas mejoras resultarían especialmente relevantes en instalaciones con un volumen elevado de movimientos o una rotación intensa de mercancía.

Finalmente, el sistema desarrollado puede considerarse una base sólida sobre la que continuar investigando y desarrollando soluciones software aplicadas a la gestión logística industrial, manteniendo la orientación práctica y el enfoque ingenieril que ha guiado este Trabajo Fin de Grado.

9 Anexos

9.1 Script SQL completo de la base de datos

```sql
create database tfg_almacenDB;

use tfg_almacenDB;

CREATE TABLE
   transportistas (
      id_transportista INT PRIMARY KEY AUTO_INCREMENT,
      nombre_empresa VARCHAR(150) NOT NULL,
      nombre_conductor VARCHAR(150) NOT NULL,
      telefono VARCHAR(20),
      email VARCHAR(120),
      matricula VARCHAR(15),
      tipo_transporte VARCHAR(50),
      direccion VARCHAR(200),
      nif_cif VARCHAR(20),
      notas TEXT,
      fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   );


CREATE TABLE
   proveedores (
      id_proveedor INT PRIMARY KEY AUTO_INCREMENT,
      nombre VARCHAR(150) NOT NULL,
      direccion VARCHAR(200),
      telefono VARCHAR(20),
      email VARCHAR(100) UNIQUE,
      nif_cif VARCHAR(20) UNIQUE,
      contacto VARCHAR(100),
      fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      ultima_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
         CURRENT_TIMESTAMP
   );

CREATE TABLE
   permisos_usuarios (
      id_permiso INT PRIMARY KEY AUTO_INCREMENT,
      permiso VARCHAR(50) NOT NULL,
      descripcion TEXT
   );

INSERT INTO
   permisos_usuarios (permiso, descripcion)
VALUES
   (
      'Gestión de Usuarios',
      'Permiso para crear, editar o eliminar usuarios del sistema.'
   ),
   (
      'Gestión de Productos',
      'Permiso para gestionar la información de los productos, incluyendo
         la creación, edición y eliminación.'
   ),
   (
```

```
53          'Gestión de Inventario',
54          'Permiso para administrar el inventario, realizar ajustes y mantener
               registros actualizados.'
55      ),
56      (
57          'Registro de Entradas/Salidas de Productos',
58          'Permiso para registrar y gestionar el movimiento de productos
               dentro y fuera del sistema.'
59      ),
60      (
61          'Gestión de Pedidos',
62          'Permiso para realizar, modificar o cancelar pedidos dentro del
               sistema.'
63      ),
64      (
65          'Generación de Reportes',
66          'Permiso para generar reportes detallados sobre diversas actividades
               dentro del sistema.'
67      ),
68      (
69          'Asignación de Tareas',
70          'Permiso para asignar tareas a usuarios o grupos dentro del sistema.
               '
71      ),
72      (
73          'Acceso al Historial de Movimientos',
74          'Permiso para acceder al historial de movimientos y transacciones
               registradas en el sistema.'
75      ),
76      (
77          'Mantenimiento de Infraestructura',
78          'Permiso para realizar tareas de mantenimiento en la infraestructura
               tecnológica del sistema.'
79      ),
80      (
81          'Gestión Financiera',
82          'Permiso para gestionar aspectos financieros, como facturación y
               pagos dentro del sistema.'
83      ),
84      (
85          'Gestión de Incidencias',
86          'Permiso para gestionar y hacer seguimiento a incidencias o
               problemas reportados en el sistema.'
87      );
88
89  CREATE TABLE
90      roles (
91          id_rol INT PRIMARY KEY AUTO_INCREMENT,
92          nombre_rol VARCHAR(20) NOT NULL,
93          descripcion TEXT
94      );
95
96  INSERT INTO
97      roles (nombre_rol, descripcion)
98  VALUES
99      (
100         'SysAdmin',
101         'Administrador del sistema con todos los privilegios.'
```

```
    ),
    (
        'Gestor Almacén',
        'Responsable de la gestión del inventario y almacenes.'
    ),
    (
        'Operario',
        'Trabajador encargado de tareas específicas operativas.'
    ),
    (
        'Administración',
        'Gestión administrativa y documentación.'
    );

CREATE TABLE
    rol_permiso (
        id_rol INT,
        id_permiso INT,
        estado ENUM ('activo', 'inactivo', 'ver') NOT NULL,
        PRIMARY KEY (id_rol, id_permiso),
        FOREIGN KEY (id_rol) REFERENCES roles (id_rol),
        FOREIGN KEY (id_permiso) REFERENCES permisos_usuarios (id_permiso)
    );

INSERT INTO
    rol_permiso (id_rol, id_permiso, estado)
VALUES
    (1, 1, 'activo'),
    (2, 1, 'inactivo'),
    (3, 1, 'inactivo'),
    (4, 1, 'inactivo');

INSERT INTO
    rol_permiso (id_rol, id_permiso, estado)
VALUES
    (1, 2, 'activo'),
    (2, 2, 'activo'),
    (3, 2, 'inactivo'),
    (4, 2, 'inactivo');

INSERT INTO
    rol_permiso (id_rol, id_permiso, estado)
VALUES
    (1, 3, 'activo'),
    (2, 3, 'activo'),
    (3, 3, 'ver'),
    (4, 3, 'ver');

INSERT INTO
    rol_permiso (id_rol, id_permiso, estado)
VALUES
    (1, 4, 'activo'),
    (2, 4, 'activo'),
    (3, 4, 'activo'),
    (4, 4, 'inactivo');

INSERT INTO
    rol_permiso (id_rol, id_permiso, estado)
```

```
160 VALUES
161    (1, 5, 'activo'),
162    (2, 5, 'activo'),
163    (3, 5, 'activo'),
164    (4, 5, 'ver');
165
166 INSERT INTO
167    rol_permiso (id_rol, id_permiso, estado)
168 VALUES
169    (1, 6, 'activo'),
170    (2, 6, 'activo'),
171    (3, 6, 'inactivo'),
172    (4, 6, 'activo');
173
174 INSERT INTO
175    rol_permiso (id_rol, id_permiso, estado)
176 VALUES
177    (1, 7, 'activo'),
178    (2, 7, 'activo'),
179    (3, 7, 'inactivo'),
180    (4, 7, 'inactivo');
181
182 INSERT INTO
183    rol_permiso (id_rol, id_permiso, estado)
184 VALUES
185    (1, 8, 'activo'),
186    (2, 8, 'activo'),
187    (3, 8, 'ver'),
188    (4, 8, 'ver');
189
190 INSERT INTO
191    rol_permiso (id_rol, id_permiso, estado)
192 VALUES
193    (1, 9, 'activo'),
194    (2, 9, 'inactivo'),
195    (3, 9, 'inactivo'),
196    (4, 9, 'inactivo');
197
198 INSERT INTO
199    rol_permiso (id_rol, id_permiso, estado)
200 VALUES
201    (1, 10, 'activo'),
202    (2, 10, 'inactivo'),
203    (3, 10, 'inactivo'),
204    (4, 10, 'activo');
205
206 INSERT INTO
207    rol_permiso (id_rol, id_permiso, estado)
208 VALUES
209    (1, 11, 'activo'),
210    (2, 11, 'inactivo'),
211    (3, 11, 'ver'),
212    (4, 11, 'inactivo');
213
214 CREATE TABLE
215    usuarios (
216       id_usuario INT PRIMARY KEY AUTO_INCREMENT,
217       user_name VARCHAR(100) NOT NULL,
```

```
218         nombre VARCHAR(100) NOT NULL,
219         apellido1 VARCHAR(100) NOT NULL,
220         apellido2 VARCHAR(100) NOT NULL,
221         email VARCHAR(100) NOT NULL UNIQUE,
222         contraseña VARCHAR(255) NOT NULL,
223         id_rol INT,
224         activo INT,
225         fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
226         FOREIGN KEY (id_rol) REFERENCES roles (id_rol) ON DELETE CASCADE ON
                UPDATE CASCADE
227     );
228
229 CREATE TABLE
230     clientes (
231         id_cliente INT PRIMARY KEY AUTO_INCREMENT,
232         nombre VARCHAR(100),
233         direccion VARCHAR(200),
234         telefono VARCHAR(20),
235         email VARCHAR(100) UNIQUE,
236         latitud DECIMAL(9, 6),
237         longitud DECIMAL(9, 6),
238         fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
239         ultima_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
                CURRENT_TIMESTAMP
240     );
241
242 CREATE TABLE
243     tipos (
244         id_tipo VARCHAR(50) PRIMARY KEY,
245         color VARCHAR(20) NOT NULL
246     );
247
248 CREATE TABLE
249     productos (
250         id_producto INT PRIMARY KEY AUTO_INCREMENT,
251         identificador_producto VARCHAR(100) NOT NULL UNIQUE,
252         tipo_producto VARCHAR(50) NOT NULL,
253         descripcion TEXT,
254         precio DECIMAL(10, 2),
255         fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
256         FOREIGN KEY (tipo_producto) REFERENCES tipos (id_tipo)
257     );
258
259 CREATE TABLE
260     proveedor_producto (
261         id_proveedor INT NOT NULL,
262         id_producto INT NOT NULL,
263         alto INT NOT NULL,
264         ancho INT NOT NULL,
265         largo INT NOT NULL,
266         precio DECIMAL(10, 2) NULL,
267         unidades_por_palet_default INT NOT NULL DEFAULT 1,
268         PRIMARY KEY (id_proveedor, id_producto),
269         FOREIGN KEY (id_proveedor) REFERENCES proveedores (id_proveedor) ON
                UPDATE CASCADE ON DELETE CASCADE,
270         FOREIGN KEY (id_producto) REFERENCES productos (id_producto) ON
                UPDATE CASCADE ON DELETE CASCADE,
```

```
271         CONSTRAINT chk_unidades_por_palet CHECK (unidades_por_palet_default
                > 0)
272     );
273
274 CREATE TABLE
275     palets (
276         id_palet INT PRIMARY KEY AUTO_INCREMENT,
277         identificador VARCHAR(50) NOT NULL UNIQUE,
278         id_producto VARCHAR(100) NOT NULL,
279         alto INT NOT NULL,
280         ancho INT NOT NULL,
281         largo INT NOT NULL,
282         cantidad_de_producto INT NOT NULL,
283         estanteria INT NOT NULL,
284         balda INT NOT NULL,
285         posicion INT NOT NULL,
286         delante BOOLEAN NOT NULL,
287         fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
288         FOREIGN KEY (id_producto) REFERENCES productos (
                identificador_producto)
289     );
290
291
292 CREATE TABLE
293     pedidos (
294         id_pedido INT PRIMARY KEY AUTO_INCREMENT,
295         codigo_referencia VARCHAR(30) NOT NULL UNIQUE,
296         id_cliente INT NOT NULL,
297         id_usuario INT NULL,
298         id_transportista INT NULL,
299         estado VARCHAR(20) NOT NULL DEFAULT 'Pendiente',
300         fecha_pedido TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
301         fecha_entrega DATE NOT NULL,
302         hora_salida ENUM ('primera_hora', 'segunda_hora') NULL,
303         palets_del_pedido INT NOT NULL DEFAULT 0,
304         enviado TINYINT (1) NOT NULL DEFAULT 0,
305         CONSTRAINT fk_pedidos_clientes FOREIGN KEY (id_cliente) REFERENCES
                clientes (id_cliente) ON UPDATE CASCADE ON DELETE RESTRICT,
306         CONSTRAINT fk_pedidos_usuarios FOREIGN KEY (id_usuario) REFERENCES
                usuarios (id_usuario) ON UPDATE CASCADE ON DELETE SET NULL,
307         CONSTRAINT fk_pedidos_transportistas FOREIGN KEY (id_transportista)
                 REFERENCES transportistas (id_transportista) ON UPDATE CASCADE
                ON DELETE SET NULL
308     );
309
310 CREATE TABLE
311     detalles_pedido (
312         id_detalle INT AUTO_INCREMENT PRIMARY KEY,
313         id_pedido INT NOT NULL,
314         id_producto INT NOT NULL,
315         cantidad INT NOT NULL,
316         estado_producto_pedido BOOLEAN DEFAULT 0,
317         paletizado BOOLEAN DEFAULT 0,
318         FOREIGN KEY (id_pedido) REFERENCES pedidos (id_pedido),
319         FOREIGN KEY (id_producto) REFERENCES productos (id_producto),
320         KEY uq_pedido_producto (id_pedido, id_producto)
321     );
322
```

```
DELIMITER / / CREATE TRIGGER generar_codigo_referencia BEFORE INSERT ON pedidos FOR EACH ROW BEGIN DECLARE secuencia_diaria INT;

DECLARE secuencia_hex VARCHAR(10);

DECLARE nuevo_codigo VARCHAR(50);

SELECT
   COUNT(*) + 1 INTO secuencia_diaria
FROM
   pedidos
WHERE
   DATE (fecha_pedido) = DATE (NEW.fecha_pedido);

SET
   secuencia_hex = LPAD (HEX (secuencia_diaria), 6, '0');

SET
   nuevo_codigo = CONCAT (
      'PED-',
      DATE_FORMAT (NEW.fecha_pedido, '%Y%m%d'),
      '-',
      secuencia_hex
   );

SET
   NEW.codigo_referencia = nuevo_codigo;

END;

/ / DELIMITER;

DELIMITER / / CREATE TRIGGER calcular_posiciones_disponibles BEFORE INSERT ON estanterias FOR EACH ROW BEGIN
SET
   NEW.posiciones_disponibles = NEW.num_baldas * NEW.posiciones_por_balda ;

END;

/ / DELIMITER;

DELIMITER / / CREATE TRIGGER generar_user_name BEFORE INSERT ON usuarios FOR EACH ROW BEGIN IF NEW.user_name IS NULL
OR NEW.user_name = '' THEN
SET
   NEW.user_name = LOWER(CONCAT (LEFT (NEW.nombre, 1), NEW.apellido1));

END IF;

END;

/ / DELIMITER;

CREATE TABLE
   orden_compra (
      id_oc INT PRIMARY KEY AUTO_INCREMENT,
      codigo_referencia VARCHAR(50) UNIQUE,
```

```
377         fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
378         observaciones TEXT NULL
379     );
380 
381 DELIMITER / / CREATE TRIGGER generar_codigo_orden_compra BEFORE INSERT ON
        orden_compra FOR EACH ROW BEGIN DECLARE secuencia_diaria INT;
382 
383 DECLARE secuencia_hex VARCHAR(10);
384 
385 IF NEW.fecha_creacion IS NULL THEN
386 SET
387    NEW.fecha_creacion = CURRENT_TIMESTAMP;
388 
389 END IF;
390 
391 SELECT
392    COUNT(*) + 1 INTO secuencia_diaria
393 FROM
394    orden_compra
395 WHERE
396    DATE (fecha_creacion) = DATE (NEW.fecha_creacion);
397 
398 SET
399    secuencia_hex = LPAD (HEX (secuencia_diaria), 6, '0');
400 
401 SET
402    NEW.codigo_referencia = CONCAT (
403       'OC-',
404       DATE_FORMAT (NEW.fecha_creacion, '%Y%m%d'),
405       '-',
406       secuencia_hex
407    );
408 
409 END;
410 
411 / / DELIMITER;
412 
413 CREATE INDEX idx_oc_fecha ON orden_compra (fecha_creacion);
414 
415 CREATE INDEX idx_oc_codigo ON orden_compra (codigo_referencia);
416 
417 CREATE TABLE
418    detalle_orden_compra (
419       id_detalle_oc INT PRIMARY KEY AUTO_INCREMENT,
420       id_oc INT NOT NULL,
421       id_proveedor INT NOT NULL,
422       id_producto INT NOT NULL,
423       cantidad INT NOT NULL CHECK (cantidad > 0),
424       estanteria INT NULL,
425       balda INT NULL,
426       posicion INT NULL,
427       delante BOOLEAN NULL,
428       FOREIGN KEY (id_oc) REFERENCES orden_compra (id_oc),
429       FOREIGN KEY (id_proveedor) REFERENCES proveedores (id_proveedor),
430       FOREIGN KEY (id_producto) REFERENCES productos (id_producto),
431       FOREIGN KEY (id_proveedor, id_producto) REFERENCES
             proveedor_producto (id_proveedor, id_producto)
432    );
```

```
433
434 CREATE INDEX idx_doc_oc ON detalle_orden_compra (id_oc);
435
436 CREATE INDEX idx_doc_prov ON detalle_orden_compra (id_proveedor);
437
438 CREATE INDEX idx_doc_prod ON detalle_orden_compra (id_producto);
439
440 DELIMITER / / CREATE PROCEDURE crear_orden_compra (
441    IN p_observaciones TEXT,
442    OUT p_id_oc INT,
443    OUT p_codigo_referencia VARCHAR(50)
444 ) BEGIN
445 INSERT INTO
446    orden_compra (observaciones)
447 VALUES
448    (p_observaciones);
449
450 SET
451    p_id_oc = LAST_INSERT_ID ();
452
453 SELECT
454    codigo_referencia INTO p_codigo_referencia
455 FROM
456    orden_compra
457 WHERE
458    id_oc = p_id_oc;
459
460 END;
461
462 / / DELIMITER;
463
464 CREATE TABLE
465    palet_salida (
466       id_palet_salida INT PRIMARY KEY AUTO_INCREMENT,
467       sscc VARCHAR(18) UNIQUE NOT NULL,
468       fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
469       cantidad_total INT,
470       numero_productos INT,
471       id_pedido INT,
472       FOREIGN KEY (id_pedido) REFERENCES pedidos (id_pedido)
473    );
474
475 CREATE TABLE
476    palet_salida_detalle (
477       id_detalle INT PRIMARY KEY AUTO_INCREMENT,
478       id_palet_salida INT NOT NULL,
479       id_producto INT NOT NULL,
480       cajas INT NOT NULL,
481       FOREIGN KEY (id_palet_salida) REFERENCES palet_salida (
             id_palet_salida),
482       FOREIGN KEY (id_producto) REFERENCES productos (id_producto)
483    );
```

9.2 Guía de símbolos del diagrama Entidad-Relación

9.2.1 Tipos de Relaciones

Símbolos	Nombre	Explicación Corta
	1:1 No Identificativa	Tablas independientes. Ej: Usuario y su Perfil opcional.
	1:N No Identificativa	Un padre, muchos hijos independientes. Ej: Departamento y Empleado.
	1:1 Identificativa	Relación fuerte; el hijo no existe sin el padre. Ej: Venta y Factura.
	1:N Identificativa	Relación de dependencia total. Ej: Libro y sus Capítulos.
	N:M (Muchos a Muchos)	Crea una tabla intermedia. Ej: Estudiantes y Cursos.

Tabla 5: Simbología de relaciones en diagramas EER.

9.2.2 Símbolos de Campos (Columnas)

Convenciones de color y forma:

1. Forma de Llave: Siempre indica una Primary Key.
2. Color Rojo: Siempre indica una relación o Foreign Key.
3. Color Azul: Indica un Atributo Simple (datos normales).
4. Relleno Sólido: El campo es NOT NULL (obligatorio).
5. Relleno Blanco (Vacío): El campo es NULLABLE (opcional).

Icono	Nombre	Descripción Técnica
	Primary Key (PK)	Identificador único de la fila. Por defecto es Not Null y Unique.
	**PK + FK**	Indica que el campo es, al mismo tiempo, Llave Primaria y Llave Foránea. Común en relaciones identificativas.
	**Not Null (NN)**	El campo es obligatorio. No permite guardar registros con este valor vacío.
	**Foreign Key (FK)**	Llave Foránea ”pura”. Vincula la tabla con la Llave Primaria de otra entidad para mantener la integridad referencial.
	**Nullable FK**	Una Llave Foránea que permite valores nulos; es decir, la relación con la otra tabla es opcional.
	**Null / Optional**	Un campo estándar (no es llave) que es opcional y permite valores nulos.

Tabla 6: Significado de los iconos de columna en MySQL Workbench.

Bibliografía

[1] Universidade de Vigo. "Arquitectura Cliente/Servidor." Material docente de la asignatura Sistemas Cliente/Servidor. Último acceso: enero 2026. dirección: https://ccia.esei.uvigo.es/docencia/SCS/1011/transparencias/Tema1.pdf

[2] Comisión Nacional de los Mercados y la Competencia. "Panel de hogares: usos de Internet." Gráfica "Sistema operativo del smartphone". Último acceso: enero 2026. dirección: https://www.cnmc.es/prensa/panel-hogares-usos-internet-20231103

[3] Google. "Android Developers Documentation," visitado ene. de 2026. dirección: https://developer.android.com/guide/components/fundamentals

[4] Asociación Española de Banca. "APIs." Documento institucional. Último acceso: enero 2026. dirección: https://s1.aebanca.es/wp-content/uploads/2018/05/apis.pdf

[5] Iniciativa Aporta. "Buenas prácticas en el diseño de APIs y linked data." Gobierno de España. dirección: https://datos.gob.es/elearning/Unidades_Didacticas/Unidad_8/contenidos/descargas/unidad_imprimible.pdf

[6] Amazon Web Services. "¿Qué es un marco en programación e ingeniería?" Visitado ene. de 2026. dirección: https://aws.amazon.com/what-is/framework/

[7] Oracle. "What Are Backends?" Visitado ene. de 2026. dirección: https://docs.oracle.com/en/cloud/paas/app-builder-cloud/visual-builder-developer/what-are-backends.html

[8] VMware. "Spring Boot Reference Documentation," visitado ene. de 2026. dirección: https://docs.spring.io/spring-boot/docs/current/reference/html/

Índice de figuras

Índice de tablas