Trabajo de Fin de Grado

Sistema de Facturación: Aplicación Móvil Nativa para la Generación y Gestión de Facturas con Funcionalidades Avanzadas

TITULACIÓN:
Grado en Ingeniería Informática

AUTOR:
Juan Carlos Acosta Perabá

TUTORIZADO POR:
María Dolores Afonso Suárez

Junio de 2025

ULPGC Universidad de Las Palmas de Gran Canaria | EII ESCUELA DE INGENIERÍA INFORMÁTICA

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D./Dª Juan Carlos Acosta Perabá, autor del trabajo Sistema de Facturación: Aplicación Móvil Nativa para la Generación y Gestión de Facturas con Funcionalidades Avanzadas, correspondiente a la titulación Grado en Ingeniería Informática

SOLICITA

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

[x] se autoriza / [ ] no se autoriza la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).

[x] No está previsto.

Y para que así conste firma la presente. (fecha en firma electrónica)

El/La estudiante

Fdo.————————

A rellenar y firmar **obligatoriamente** por el/la/los/las tutores

En relación con la presente solicitud, se informa:(firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)	Negativamente (justificación en TFT05)
Fdo.————————	Fdo.————————

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

En primer lugar, deseo expresar mi más profundo agradecimiento a mis padres, a mi hermana, a mi pareja y a mis amigos por su apoyo incondicional a lo largo de este arduo camino, por acompañarme y levantarme cada vez que sentí que no podía más.

Agradezco asimismo a la persona que me guió en este proyecto por su paciencia inquebrantable y la confianza que depositó en mí.

Finalmente, gracias a todas las personas que, de una u otra forma, hicieron posible que llegara hasta aquí.

Muchas gracias.

Resumen

Este Trabajo de Fin de Grado presenta una aplicación móvil nativa, complementada por un portal web, para la gestión integral de la facturación de empresas y autónomos. La plataforma centraliza facturas, empleados y clientes; permite registrar negocios, crear cuentas para el personal y ofrecer a los clientes acceso directo a sus documentos. Incluye generación de facturas mediante formularios dinámicos, almacenamiento en la nube, exportación PDF, trabajo offline con sincronización posterior y notificaciones automáticas. La herramienta mejora la eficiencia y accesibilidad del proceso, reduce la dependencia del papel y aporta una solución moderna, segura y alineada con las necesidades actuales de la gestión empresarial.

Abstract

This Final Degree Project presents a native mobile application, complemented by a web portal, for comprehensive invoicing management for businesses and freelancers. The platform centralises invoices, employees and clients; it lets owners register their company, create staff accounts and grant customers direct access to their documents. Features include invoice generation through dynamic forms, cloud storage, PDF export, offline work with later synchronisation and automatic notifications. The tool enhances efficiency and accessibility in invoicing, reduces reliance on paper and provides a modern, secure solution aligned with current business management needs.

Índice general

Bloques de código

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

> Si enciendes una lámpara para alguien,
> iluminas también tu propio camino.
>
> Buda

La facturación es uno de los cimientos de la actividad económica: registra las operaciones comerciales, permite el control fiscal y protege los derechos de compradores y vendedores. A lo largo de la historia, desde las tablillas mesopotámicas hasta las facturas digitales que circulan hoy día, la tecnología ha perfeccionado este proceso para garantizar disponibilidad, autenticidad e integridad de la información [1].

En España, la **Ley 18/2022 de Creación y Crecimiento Empresarial** ha acelerado esta transición al imponer la factura electrónica en todas las operaciones B2B. Las grandes empresas ya la aplican de forma obligatoria y, para pymes y autónomos, la exigencia será plena a lo largo de 2025 [2]. Al mismo tiempo, la expansión del trabajo remoto y el uso extendido de dispositivos móviles han mostrado claramente la diferencia que existe entre los programas de escritorio tradicionales y las necesidades actuales de movilidad.

En este trabajo se aborda ese desafío mediante un sistema de facturación centrado en aplicaciones móviles nativas (iOS y Android) junto con un panel

administrativo web.

1.1. Motivación y antecedentes

La gestión de facturas en la pequeña y mediana empresa sigue apoyándose, en gran medida, en un conjunto de soluciones aisladas: sistemas ERP de sobremesa, hojas de cálculo improvisadas y repositorios documentales que raramente se comunican entre sí. Esta fragmentación causa que los datos se dupliquen y dificulta la colaboración directa entre empresario, personal interno y cliente final.

Al mismo tiempo, el auge de los servicios SaaS ha desplazado el trabajo contable a la nube; sin embargo, muchos de estos portales exigen conectividad permanente para funcionar. En zonas con cobertura deficiente, la emisión o actualización de facturas se retrasa, lo que genera cuellos de botella y pone en riesgo la puntualidad de la información.

La obligatoriedad próxima de la factura electrónica, recogida en la Ley 18/2022, obliga a las pymes y autónomos a actualizar sus procesos administrativos para adaptarse plenamente antes de 2025.

Por último, la movilidad se ha consolidado como requisito operativo. Según el «2023 Mid-market Technology Trends Report» de Deloitte, la mayoría de las empresas de tamaño medio ofrece hoy acceso a sus plataformas de gestión desde dispositivos móviles, aunque con frecuencia esas aplicaciones replican solo las funciones básicas de escritorio [3].

1.2. Objetivos

1.2.1. Objetivo general

El objetivo general es desarrollar una plataforma integral de facturación con aplicaciones móviles nativas para iOS y Android, complementadas por un portal web administrativo. La solución debe operar sin interrupciones, incluso en ausencia de conexión, y ofrecer una experiencia de uso clara, accesible y satisfactoria.

1.2.2. Objetivos específicos

Los objetivos específicos que guían el desarrollo del proyecto son:

En primer lugar se plantea el diseño de una arquitectura de datos sólida que represente con precisión las entidades del dominio, es decir, empresas, usuarios y facturas, y permita la evolución y el mantenimiento del sistema sin inconsistencias.

El segundo objetivo establece la implantación de un mecanismo de autenticación y autorización que gestione roles y permisos con rigor, de manera que cada perfil acceda únicamente a las funciones que le correspondan.

El tercer objetivo se centra en desarrollar la lógica de generación automática de facturas, incorporando el cálculo exacto de impuestos y la exportación de los documentos a PDF con calidad profesional y compatibilidad con los estándares de facturación electrónica.

El cuarto objetivo persigue proporcionar a la plataforma un modo de funcionamiento sin conexión que posibilite crear o modificar facturas en entornos offline y sincronizar los cambios en cuanto se restablezca la conectividad.

El quinto objetivo aborda la construcción de un portal web administrativo con cuadros de mando y métricas financieras detalladas, cuyo propósito es facilitar el seguimiento de ingresos, gastos y morosidad y respaldar la toma de decisiones.

Por último, el sexto objetivo apunta a proteger la confidencialidad e integridad de la información mediante cifrado tanto en tránsito como en reposo, garantizando el cumplimiento de los requisitos establecidos por el RGPD.

1.3. Competencias específicas

El desarrollo de este proyecto ha requerido poner en práctica un conjunto de competencias técnicas directamente relacionadas con el diseño, implementación y evaluación de sistemas de información avanzados. A continuación se detallan las competencias específicas que se han puesto en práctica a lo largo de la realización de este trabajo:

- **CI1.** Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad y calidad.

- **CI2.** Aptitud para planificar, concebir y dirigir proyectos y servicios informáticos a lo largo de todo su ciclo de vida.

- **CI3.** Conocimiento de las métricas de calidad y los procesos de mejora continua aplicados al software.

- **CI5.** Capacidad para comprender, aplicar y mantener los principios de ingeniería de requisitos.

- **CI6.** Habilidad para diseñar y gestionar bases de datos y servicios en la nube, garantizando integridad y disponibilidad de la información.

- **CI8.** Competencia para analizar, construir y mantener aplicaciones robustas y eficientes, eligiendo los lenguajes y paradigmas más adecuados.

- **CI12.** Conocimiento y aplicación de las estructuras de datos y su impacto en el rendimiento del sistema.

- **CI16.** Aplicación de los principios y metodologías de la ingeniería del software para asegurar procesos ordenados de desarrollo y mantenimiento.

- **CI17.** Capacidad para diseñar y evaluar interfaces persona–computador que garanticen accesibilidad y usabilidad.

- **TI1.** Comprensión del entorno de una organización y de sus necesidades en materia de tecnologías de la información.

- **TI2.** Capacidad para seleccionar, desplegar e integrar tecnologías de hardware, software y redes dentro de los parámetros de calidad adecuados.

- **TI3.** Uso de metodologías centradas en el usuario y la organización para el desarrollo y la evaluación de aplicaciones.

- **TI5.** Capacidad para seleccionar, integrar y gestionar sistemas de información que satisfagan los requisitos de la organización.

- **TI6.** Capacidad para concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluidos entornos web y móviles.

- **TI7.** Aptitud para aplicar y gestionar la garantía y la seguridad de los sistemas de información.

Estos conocimientos y habilidades, extraídos directamente del plan de es-

tudios oficial, han sido fundamentales para llevar a cabo el análisis, diseño e implementación de la plataforma de facturación propuesta [4].

1.4. Alineamiento con los Objetivos de Desarrollo Sostenible

Analizando la contribución de este TFG a los Objetivos de Desarrollo Sostenible (ODS) definidos por la Agenda 2030, se ha evaluado el impacto potencial de la propuesta en relación con cada uno de ellos. En el Cuadro 1.1 se muestra el grado de relación alcanzado.

Cuadro 1.1: Grado de relación del TFT con los objetivos de desarrollo sostenible.

ODS	Grado de relación			
	0 No procede	1 Bajo	2 Medio	3 Alto
1 Fin de la Pobreza	X			
2 Hambre cero	X			
3 Salud y Bienestar	X			
4 Educación de calidad	X			
5 Igualdad de género	X			
6 Agua limpia y saneamiento	X			
7 Energía asequible y no contaminante	X			
8 Trabajo decente y crecimiento económico	X			
9 Industria, innovación e infraestructuras				X
10 Reducción de las desigualdades		X		
11 Ciudades y comunidades sostenibles			X	
12 Producción y consumo responsables			X	
13 Acción por el clima		X		
14 Vida submarina	X			
15 Vida de ecosistemas terrestres	X			
16 Paz, justicia e instituciones sólidas	X			
17 Alianzas para lograr objetivos	X			

En primer lugar, en relación con el **ODS 9 – Industria, innovación e infraestructuras**, la aplicación propone una infraestructura digital accesible, capaz de operar sin conexión para garantizar la continuidad del negocio en entornos con conectividad limitada. Además, sus funciones de generación y

exportación de facturas en PDF modernizan los procesos administrativos.

Respecto al **ODS 10 – Reducción de las desigualdades**, el diseño centrado en la usabilidad del teléfono móvil facilita que microempresas y autónomos, con recursos más reducidos, puedan adoptar tecnología avanzada de facturación, reduciendo la brecha digital frente a grandes organizaciones.

En cuanto al **ODS 11 – Ciudades y comunidades sostenibles**, la eliminación del papel y la eliminación de desplazamientos para la entrega física de documentos contribuye a disminuir residuos urbanos y a apoyar la sostenibilidad económica y medioambiental de los comercios locales.

El **ODS 12 – Producción y consumo responsables** se ve favorecido al sustituir procesos basados en facturas impresas por archivos cifrados en la nube, promoviendo prácticas *paper-less* y un uso más eficiente de los recursos materiales y energéticos.

Finalmente, el **ODS 13 – Acción por el clima** recibe un impulso gracias a la reducción del consumo de papel y de envíos postales, lo que disminuye la huella de carbono administrativa. Aunque el uso de dispositivos móviles y servicios en la nube conlleva consumo energético, el balance de emisiones es muy inferior al de los métodos tradicionales, siempre que se adopten buenas prácticas de eficiencia energética.

Capítulo 2

Análisis

2.1. Problema

El día a día de muchas pymes y autónomos sigue un recorrido fragmentado, combinando plantillas en hojas de cálculo, correos electrónicos y archivado físico. Esta fragmentación obliga a reintroducir datos manualmente en varias aplicaciones, aumentando el riesgo de errores humanos e inconsistencias, dificultando la obtención de información financiera fiable y generando retrasos en los cobros y pagos. Además, la falta de conectividad estable limita gravemente la movilidad, impidiendo emitir o consultar facturas en tiempo real, provocando que la información quede rápidamente obsoleta.

A esto se suman las exigencias normativas recientes, como la Ley 18/2022, que obliga a las empresas a emitir factura electrónica en todas sus transacciones comerciales, mantener registros íntegros y garantizar la autenticidad de cada documento. Cumplir con estos requisitos con herramientas tradicionales implica una inversión considerable en nuevas soluciones, formación del personal y rediseño de procesos internos, afectando negativamente la productividad, la competitividad y el potencial de crecimiento de los negocios.

2.2. Solución

Se propone una plataforma de facturación basada en aplicaciones móviles nativas para iOS y Android y un portal web administrativo. Todas las versiones se conectan a Firestore, lo que garantiza la consistencia de los datos. Firebase actúa como backend sin servidor, reduciendo la carga de mantenimiento y permitiendo un escalado automático según el uso.

Las apps móviles siguen un modelo *offline-first*: cuando no hay conexión, los datos se almacenan localmente de forma segura y se sincronizan con Firestore tan pronto se restablece la red. Así, se puede seguir trabajando sin interrupciones y los posibles conflictos se resuelven automáticamente mediante marcas de tiempo.

Se plantea utilizar Cloud Functions para generar los PDF de las facturas, aplicar numeración secuencial y almacenar los archivos en Cloud Storage. La seguridad está cubiera con reglas de Firestore, autenticación y cifrado, cumpliendo con el RGPD y dejando registro de auditoría para cada operación sensible.

El portal web, construido con Astro y alojado en Vercel, muestra en tiempo real indicadores clave (ingresos cobrados, importes pendientes, ratio de morosidad). Gracias a la suscripción en tiempo real a Firestore y al enfoque de generar HTML estático en el servidor y habilitar la hidratación únicamente en los componentes que lo requieren, la interfaz permanece ligera, accesible y adaptable a distintos dispositivos. En conjunto, esta solución unifica el flujo de trabajo, elimina la fragmentación de herramientas y permite facturar desde cualquier lugar, reduciendo costes operativos y minimizando errores.

2.3. Requisitos del sistema

2.3.1. Requisitos funcionales (RF)

RF1. Gestión de usuarios

- Registro de empresarios.
- Inicio y cierre de sesión de usuarios.
- Creación y gestión de empleados por parte del empresario.
- Alta de clientes asociados a cada empresa.
- Separación y aislamiento de datos por empresa.

RF2. Facturación

- Creación, edición, consulta y eliminación de facturas por el empresario (CRUD).
- Creación, edición y consulta de facturas por empleados (CRU).
- Lectura de facturas filtradas por cliente.
- Descarga de cada factura en formato PDF.
- Envío y reenvío de facturas por correo electrónico.
- Gestión de estados de factura: *Pagada*, *Pendiente*, *Vencida*.

- Marcado manual de factura como pagada por el empresario.
- Notificación push y correo al empresario y al cliente antes del vencimiento.

RF3. Funcionalidades avanzadas

- Facturación en modo *offline* con posterior sincronización.
- Firma digital de facturas.
- Escaneo de recibos mediante OCR para precargar datos.

RF4. Gestión de clientes

- Listado y búsqueda de clientes.
- Historial de facturas por cliente.

RF5. Seguridad y control

- Autenticación segura con Firebase Auth y roles de usuario.
- Registro de actividad de los usuarios en las operaciones sensibles.

2.3.2. Requisitos no funcionales (RNF)

RNF1. Disponibilidad: la aplicación iOS debe usable tanto con conexión como offline.

RNF2. Usabilidad: emitir una factura completa en menos de un minuto.

RNF3. Seguridad: todas las operaciones deben pasar por reglas de Firestore y HTTPS. Datos cifrados en tránsito y reposo.

RNF4. Escalabilidad: el modelo en Firestore debe soportar al menos 10 000 facturas por negocio sin rediseño.

RNF5. Compatibilidad: portal web operativo en Chrome, Firefox, Safari y Edge con versiones de los últimos dos años.

RNF6. Mantenibilidad: el código se organizará en MVVM (iOS) y en componentes desacoplados (Astro) para facilitar refactorización.

2.4. Metodología

Para desarrollar la plataforma se adoptó un enfoque ágil iterativo incremental basado en Kanban. Cada tarea se crea y prioriza en GitHub Projects, limitando el trabajo en curso para enfocarnos en entregas parciales y ajustando el alcance según sea necesario.

El plan de trabajo se estructuró en siete fases bien diferenciadas: una **planificación inicial**; cinco **incrementos funcionales**; y una fase final de **documentación y presentación**. Cada incremento funcional se subdividió de forma repetible en las cuatro tareas que se describen a continuación:

- **Análisis:** Detallamos funcionalidades y criterios de aceptación para cada incremento.
- **Diseño:** Definimos la arquitectura de datos y creamos prototipos de interfaz en Figma.
- **Implementación:** Desarrollamos la app para iOS y Android, escribimos las funciones de backend y construimos el portal web.

- **Pruebas:** Realizamos pruebas manuales en dispositivos emulados para verificar que cada función cumple los requisitos.

Al finalizar los cinco incrementos funcionales se dedicó tiempo a la elaboración de la documentación completa: manuales de usuario, guía de despliegue y la memoria técnica. Durante esta etapa se generaron capturas y métricas que respaldan los resultados.

Con este esquema iterativo avanzamos de forma progresiva, mantuvimos la calidad en cada entrega y pudimos incorporar mejoras o ajustes cuando surgían nuevos requisitos.

2.5. Marco de trabajo

El marco de trabajo del proyecto se articula en dos grandes componentes: los clientes nativos y el portal web. Cada uno cumple una función específica dentro del flujo integral de facturación:

2.5.1. App iOS (*Swift*)

El código de la aplicación iOS sigue el patrón Model–View–ViewModel (MVVM) para separar lógicamente la interfaz de usuario de la capa de datos y negocio.

- **Modelo (Model):** Contiene las estructuras que representan entidades como Empresas, Clientes y Facturas. Estas estructuras definen los campos que se almacenan en Firestore (por ejemplo, fecha de emisión, importe, estado, etc.) y métodos para convertir documentos de Firestore a objetos Swift.

- **Vista (View):** Con SwiftUI se crearon vistas reutilizables para cada

pantalla principal: lista de facturas, detalle de factura, formulario de creación/edición y ajustes. Cada vista se conecta a su ViewModel mediante propiedades marcadas con *@ObservedObject* o *@StateObject*.

- **ViewModel:** Cada pantalla tiene un ViewModel que agrupa la lógica de negocio y la interacción con Firebase. Los ViewModels se encargan de:
  - Recuperar y observar los cambios en documentos de Firestore.
  - Validar y formatear datos antes de mostrarlos en pantalla.
  - Enviar nuevas facturas a Firestore, gestionar el estado (Paid/Pending) y calcular totales.
  - Generar el PDF de la factura y guardarlo localmente, empleando el ID único de Firestore para numeración.
- **Servicios de Firebase:**
  - *Firestore:* Actúa como base de datos de documentos. Cada colección almacena objetos con la información necesaria.
  - *Authentication:* Gestiona el inicio de sesión con correo/contraseña, manteniendo sesiones y seguridad.

2.5.2. App Android (Kotlin)

La versión para Android estaba prevista en Kotlin siguiendo una arquitectura similar (MVVM), pero debido a limitaciones de tiempo no llegó a completarse. La carpeta *android/* contiene el esqueleto del proyecto (configuración de Gradle, dependencias de Firebase, diseño de pantallas básicas), pero

las funciones principales (CRUD de facturas, generación de PDF, lógica de sincronización) quedaron pendientes y se documentan como trabajo futuro.

2.5.3. Portal Web (Astro)

El portal web se desarrolló con el framework Astro y estilos en TailwindCSS. La estructura principal del código es:

■ **Páginas estáticas (‘/src/pages‘):**

- *index.astro*: Página de inicio que contiene la gráfica donde se muestra el total de los pagado y de lo que queda pendiente de pago mensualmente.

- *login.astro*: Página para que los usuarios, previamente registrados desde la aplicación móvil, puedan acceder con sus credenciales.

■ **Componentes (‘/src/components‘):**

- *ButtonToScrollToTop.astro*: Componente que renderiza un botón en la esquina inferior derecha de la página para volver al inicio de la página.

- *Header.astro*: Componente que renderiza la cabecera de la página con un enlace de la web, un menú de navegación entre las diferentes secciones y el botón de cerrar la sesión.

- *MoneyChart.astro*: Componente que renderiza la gráfica de barras usando *lightweight-charts*.

- *OverdueList.astro*: Componente que renderiza el listado de los clientes que no han pagado en su tiempo (morosos), haciendo el cálculo

de la cantidad total que deben.

■ **Servicios de Firebase (en '/src/firebase'):**

- *config.ts*: Fichero con la configuración para establecer la conexión con firebase.

- *firebase.ts*: Inicializa la instancia de Firebase y exporta funciones para autenticar y leer cualquier dato de Firestore.

■ **Despliegue:**

- El portal web se publica en Vercel, con integración continua vía GitHub Actions. Cada merge a 'main' dispara un despliegue automático.

2.6. Herramientas

En este proyecto hemos elegido herramientas que permitan un flujo de trabajo ágil y tecnológico acorde al alcance de la plataforma:

2.6.1. iOS

Para la app de iPhone se utiliza **Swift**, un lenguaje moderno que destaca por su seguridad (tipado fuerte y gestión automática de memoria) y buen rendimiento gracias a sus compiladores optimizados [5]. El desarrollo se lleva a cabo en **Xcode** 16.3, que combina editor, depurador y simulador de dispositivos. Además, Instruments facilita perfilar CPU, memoria y consumo energético, lo que ayuda a mejorar la fluidez de la interfaz. Con SwiftUI, se construyen vistas reutilizables que interactúan con los ViewModels, separando claramente la lógica de negocio de la presentación.

2.6.2. Android

La versión para Android se prediseñó en **Kotlin** con una arquitectura MVVM similar a la de iOS, aprovechando su sintaxis concisa, el manejo seguro de nulabilidad y las corrutinas para tareas asíncronas [6]. El entorno de trabajo es **Android Studio** 2024.3.2 (Meerkat Feature Drop), que incluye emuladores configurables, herramientas para inspeccionar bases de datos y analizadores de rendimiento en tiempo real [7]. En esta primera iteración el esqueleto del proyecto está preparado, aunque las funcionalidades principales se documentan para futuras versiones.

2.6.3. Web

El portal administrativo se desarrolla con **Astro**, que genera HTML estático en tiempo de compilación y aplica hidratación sólo donde se necesita, lo que reduce el tamaño inicial y mejora el rendimiento [8]. Los estilos se gestionan con **TailwindCSS**, facilitando una personalización rápida y coherente sin escribir CSS extenso. Para instalar dependencias se usa **pnpm**, que acelera el proceso al compartir una caché global y evitar duplicados [9]. El sitio se despliega en **Vercel**, integrando despliegues automáticos en cada fusión a la rama principal.

2.6.4. Servicios en la nube

Como backend se optó por **Firebase**, por su conjunto completo de servicios gestionados. *Firestore* ofrece una base de datos NoSQL en tiempo real y escalable, mientras que *Authentication* gestiona usuarios con correo y contraseña de forma sencilla. Aunque se planteó usar *Cloud Functions* para tareas como generación de PDFs y almacenamiento, algunas de esas funciones quedaron para versiones posteriores debido a restricciones de coste [10]. Las reglas de Firestore garantizan seguridad tanto en lectura como en escritura, cumpliendo con el RGPD.

Para el despliegue del portal web se eligió **Vercel**, una plataforma de hosting enfocada en sitios estáticos y Jamstack. Vercel ofrece despliegues automáticos a partir del repositorio de GitHub, gestión de dominios y certificados TLS integrados, así como funciones serverless ligeras si fueran necesarias [11]. Gracias a su integración continua y facilidad de configuración, el portal se publica con cada push a la rama principal, garantizando que la versión en producción esté siempre actualizada y optimizada.

2.6.5. Control de versiones

El código se gestiona con **Git**, un sistema distribuido que conserva el historial completo y permite revertir cambios con precisión [12]. El repositorio principal se aloja en **GitHub**, donde las *pull requests* facilitan las revisiones entre pares. Así, cada confirmación se valida y despliega de forma controlada, asegurando una integración continua estable.

Capítulo 3

Desarrollo

3.1. Base de datos

Para esta plataforma, la base de datos se implementó íntegramente en Firestore, aprovechando su modelo NoSQL en tiempo real y su escalabilidad automática. A continuación se describen los pasos clave:

3.1.1. Configuración inicial en Firebase

En primer lugar, se creó un proyecto en la consola de Firebase y se habilitaron los servicios de Firestore y Authentication. Dentro de Firestore, se generó la colección raíz *businesses*, donde cada documento tiene un identificador único (*businessId*). De forma automática, Firestore asigna una referencia a cada colección y subcolección cuando se insertan documentos por primera vez desde los clientes.

Para conectar la aplicación iOS, se descargó el archivo *GoogleService-Info.plist* desde la consola de Firebase y se añadió al directorio principal del proyecto Xcode. En el caso del portal web, se obtuvo el objeto de configura-

ción (API key, project ID, etc.), se guardó en un archivo *.env* y se cargó desde un archivo *config.ts* bajo */src/firebase*.

3.1.2. Modelo de datos en Firestore

```
/businesses/{businessId}
    |-- name
    |-- tagline (nullable)
    |-- address
    |-- city
    |-- country
    |-- postcode
    |-- taxId (CIF in Spain, VAT in the UK, EIN in the USA, etc.)
    |-- email
    |-- phone
    |-- createdAt
    |-- subscriptionPlan: "free" | "premium"
    |-- storageLimit: "1 month" | "unlimited"
    |
    |-- employees/{employeeId}
    |        |-- userId (linked to Firebase Auth)
    |        |-- name
    |        |-- email (linked to Firebase Auth)
    |        |-- phone
    |        |-- role: "admin" | "employee" (linked to Firebase Auth)
    |        |-- joinedAt
    |
    |-- clients/{clientId}
    |        |-- userClientId (linked to Firebase Auth) ??
    |        |-- companyName
    |        |-- clientName
    |        |-- address
    |        |-- city
    |        |-- country
    |        |-- postcode
    |        |-- email (linked to Firebase Auth)
    |        |-- phone
    |        |-- createdAt
    |
    |-- invoices/{invoiceId}
    |        |-- issuedAt
    |        |-- dueDate
    |        |-- status: "Paid" | "Pending" | "Overdue"
    |        |-- subtotal
    |        |-- taxTotal
```

```
|         |-- discounts
|         |-- totalAmount
|         |-- currency
|         |-- clientId
|         |-- employeeId
|         |-- pdfURL (Cloudflare)
|         |-- deleteAfter (Free plan: issuedAt + 1 month,
                           Premium plan: unlimited)
|         |-- productsStack/{productStackId}
|                          |-- productId
|                          |-- supplyDate
|                          |-- quantity
|                          |-- amount
|                          |-- taxRate
|                          |-- taxNet
|
|-- products/{productId}
|         |-- description
|         |-- unitPrice
```

Algoritmo 3.1: Modelo de datos en Firestore

businesses/{businessId}

Documento principal que contiene datos generales de la empresa (nombre, contactos, identificación fiscal, suscripción y límites de almacenamiento) y agrupa subcolecciones.

businesses/{businessId}/employees/{employeeId}

Empleados de la empresa, con referencia a Firebase Auth, datos de contacto y rol.

businesses/{businessId}/clients/{clientId}

Clientes asociados a la empresa, con datos de contacto y posible vinculación a Firebase Auth.

businesses/{businessId}/invoices/{invoiceId}

Facturas emitidas por la empresa, con fechas, estado, importes y subcolección *productsStack* para desglosar cada línea de producto (ID de producto, cantidad, impuesto, etc.).

businesses/{businessId}/products/{productId}
: Productos que la empresa ofrece, con descripción y precio unitario.

Esta organización centraliza la lógica de datos y asegura coherencia en todos los clientes, facilitando la mantenibilidad y escalabilidad del sistema.

3.1.3. Reglas de seguridad de Firestore

Para garantizar que solo los usuarios autorizados accedieran a la información, se definieron reglas en el panel de Firestore. Un extracto de las reglas más relevantes es:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // Raiz de negocios: controlamos creacion/lectura/actualizacion/borrado a nivel de negocio
    match /businesses/{businessId} {
      // Cualquiera autenticado puede leer la informacion de un negocio
      allow read: if request.auth != null;
      // Cualquiera autenticado puede crear un nuevo negocio
      allow create: if request.auth != null;
      // Solo un admin (claim role == "admin") puede modificar o borrar un negocio
      allow update, delete: if request.auth.token.role == 'admin';

      // Subcoleccion de empleados: solo admins pueden gestionar empleados
      match /employees/{employeeId} {
        allow read: if request.auth != null;
        allow create, update, delete: if request.auth.token.role == 'admin';
      }

      // Subcoleccion de clientes:
      //   - cualquiera autenticado ve datos de clientes
      //   - un admin o el propio cliente (uid == userClientId) puede crear/editar/borrar
      match /clients/{clientId} {
        allow read: if request.auth != null;
        allow create: if request.auth.token.role == 'admin';
        allow update, delete: if request.auth.token.role == 'admin'
                              || request.auth.uid == resource.data.userClientId;
      }

      // Subcoleccion de facturas:
      //   - cualquiera autenticado puede leer
      //   - crear: admin o empleado que firma la factura (employeeId en request.resource.data)
      //   - actualizar: admin o empleado que creo la factura (employeeId en el dato existente)
      //   - borrar: solo admin
      match /invoices/{invoiceId} {
        allow read: if request.auth != null;
```

```
        allow create: if request.auth.token.role == 'admin'
                      || request.auth.uid == request.resource.data.employeeId;
        allow update: if request.auth.token.role == 'admin'
                      || request.auth.uid == resource.data.employeeId;
        allow delete: if request.auth.token.role == 'admin';

        // Dentro de cada factura, la subcoleccion de lineas de producto:
        //   - cualquiera autenticado puede leer
        //   - solo el admin puede escribir/borrar
        match /productsStack/{productStackId} {
          allow read: if request.auth != null;
          allow create, update, delete: if request.auth.token.role == 'admin';
        }
      }

      // Subcoleccion de productos:
      //   - cualquiera autenticado puede leer
      //   - solo admin puede escribir/borrar
      match /products/{productId} {
        allow read: if request.auth != null;
        allow create, update, delete: if request.auth.token.role == 'admin';
      }
    }

  }
}
```

Algoritmo 3.2: Reglas de seguridad de Firestore

Con estas reglas, solo el administrador (rol *admin*) puede crear o eliminar facturas y productos, y cada empleado (rol *employee*) puede crear o actualizar facturas que él mismo genere. Los clientes solo acceden a sus propios documentos, y todas las operaciones requieren un usuario autenticado (*request.auth != null*), lo que cumple con los requisitos de seguridad y protección de datos.

3.2. Implementación iOS

En esta sección profundizamos en la aplicación iOS, donde se dedicaron la mayoría de las horas de desarrollo. Partimos desde cero, aprendiendo Swift y SwiftUI, y construimos la app siguiendo el patrón MVVM para separar responsabilidades y lograr un código más mantenible.

3.2.1. Configuración del proyecto y conexión a Firebase

Para iniciar el proyecto, se utilizó Xcode 16.3 con la plantilla de SwiftUI. Los pasos fueron:

1. En Xcode, crear un nuevo proyecto seleccionando *App* y, como lenguaje, **Swift** con interfaz **SwiftUI**.

2. Descargar *GoogleService-Info.plist* desde la consola de Firebase y arrastrarlo al grupo raíz del proyecto en Xcode. Asegurarse de marcar la opción de agregarlo a todos los targets.

3. Instalar el paquete oficial de Firebase mediante Swift Package Manager (*File → Add Packages...*), usando la URL https://github.com/firebase/firebase-ios-sdk.git.

4. En el archivo *QuickBillApp.swift*, inicializar Firebase antes de cargar la escena principal:

```
import SwiftUI
import FirebaseCore

@main
struct QuickBillApp: App {
    @StateObject private var auth = AuthViewModel()
    @AppStorage("appLanguage") private var appLanguage: String = AppLanguage.english.rawValue

    init() {
        FirebaseApp.configure()
    }

    var body: some Scene {
        WindowGroup {
            if auth.isSignedIn {
                MainTabView()
            } else {
                StartView()
            }
        }
        .environmentObject(auth)
        .environment(\.locale, Locale(identifier: appLanguage))
    }
}
```

Algoritmo 3.3: Inicialización de Firebase en QuickBillApp.swift

5. Configurar en la consola de Firebase el dominio de la app, activando Authentication con correo/contraseña.

Con estos pasos, la aplicación iOS ya está conectada a Firebase y lista para consumir Firestore y Authentication.

3.2.2. Arquitectura MVVM

La aplicación iOS se organiza siguiendo el patrón *Model–View–ViewModel* (MVVM). En la Figura 3.1 se muestra la distribución real de carpetas dentro de Xcode, enfocándose en la carpeta de los modelos:

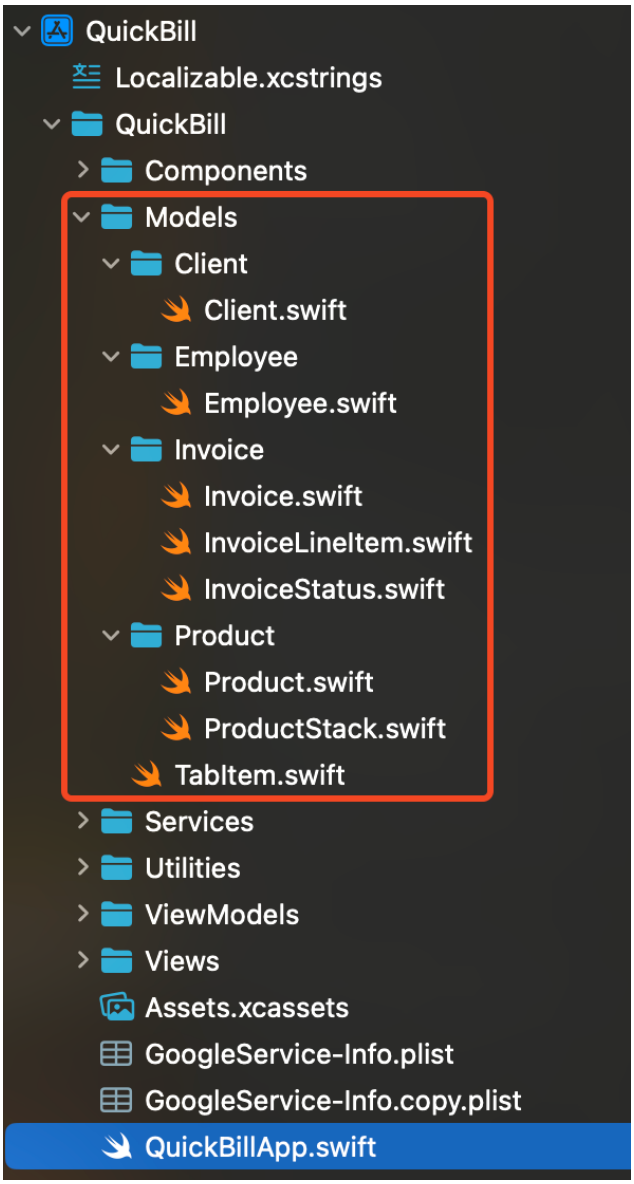

Ilustración 3.1: Estructura de la carpeta *Models* en Xcode

A continuación se describe brevemente el propósito de cada grupo.

Models agrupa todas las entidades que reflejan los documentos de Firestore. Cada subcarpeta contiene un único fichero con la estructura correspondiente:

- **Client/Client.swift**: define la estructura *Client* con campos básicos de la empresa o persona a facturar.

- **Employee/Employee.swift**: representa a cada empleado que accede a la app. Incluye su *userId* (Firebase Auth), nombre, email, teléfono y rol.

- **Invoice**:

  - *Invoice.swift*: entidad principal con fechas, importes y referencias a cliente y empleado.

  - *InvoiceLineItem.swift*: describe cada línea de producto/servicio dentro de una factura.

  - *InvoiceStatus.swift*: *enum* con los estados *paid*, *pending* y *overdue*.

- **Product**:

  - *Product.swift*: catálogo de productos con descripción y precio unitario.

  - *ProductStack.swift*: colección de productos que se añaden a una factura.

- *TabItem.swift*: enumeración con los tabs de navegación de la app.

Services agrupa la lógica que no depende de la UI pero tampoco encaja como modelo puro:

- *AuthService.swift*: envoltorio sencillo sobre Firebase Auth con la finalidad de persistir la sesión de los usuarios en la aplicación.

- *InvoicePDFBuilder.swift*: genera el PDF de una factura usando UIGraphicsPDFRenderer; recibe una instancia *Invoice* y devuelve la URL local del fichero.

- *InvoiceStatusService.swift*: actualiza el estado de una factura (*paid*, *pending*, *overdue*) según *issuedAt*, *dueDate* y el *status* almacenado.

Utilities contiene código de soporte reutilizable. Actualmente sólo incluye *AppLanguage.swift*, una enumeración para cambiar el idioma de la interfaz mediante la propiedad *@AppStorage('appLanguage')*.

ViewModels se divide en subcarpetas por dominio de negocio:

- **Authentication**: gestiona el inicio de sesión, registro y la opción de 'ovidé mi contraseña'.

- **Clients**: gestión de listado, creación, borrado y edición de clientes.

- **Invoice**: *InvoiceListViewModel.swift*, *InvoiceDetailViewModel.swift* y *AddInvoiceViewModel.swift* para listar, mostrar los detalles y crear nuevas faturas.

- **Products**: ViewModel para el catálogo de productos.

- **Settings**: controla las preferencias de la app.

- Además, en la raíz de la carpeta está *AuthViewModel.swift*, que mantiene el estado global de autenticación y se inyecta como *EnvironmentObject*.

Views organiza las pantallas SwiftUI con la misma lógica de dominios:

- **Authentication**: vistas de inicio de sesión y registro.

- **Clients**: lista de clientes y formulario de creación y edición.

- **Home**: vista inicial tras iniciar sesión (dashboard con todas las facturas).

- **Invoice**: formulario de creación y detalles de las facturas.
- **Products**: listado y formulario de productos.
- **Settings**: ajustes de idioma y cuenta.
- *MainTabView.swift*: barra de pestañas inferior que enlaza *Home*, *Products*, *Invoice*, *Clients*, y *Settings*.
- *StartView.swift*: pantalla que decide si mostrar la vista de autenticación o el *MainTabView* según el estado de *AuthViewModel*.

Esta distribución favorece la separación de responsabilidades, donde los ViewModels actúan como enlace entre modelos y vistas, y cada conjunto de vistas se mantiene cohesionado dentro de su propio dominio y los modelos definen la estructura de los datos que se manejan.

3.2.3. Generación de PDF

En lugar de recurrir a Cloud Functions, la app genera el PDF de la factura localmente a través del servicio *InvoicePDFBuilder*. El proceso se resume a continuación; el código completo puede consultarse en el repositorio público.

1. **Crear el lienzo A4**
   Se instancia un *UIGraphicsPDFRenderer* con tamaño A4 y metadatos del documento:

```
let bounds = CGRect(x: 0, y: 0, width: 595, height: 842) // A4 @72 dpi
let format = UIGraphicsPDFRendererFormat()
format.documentInfo = [
  kCGPDFContextCreator: "QuickBill",
  kCGPDFContextAuthor : "QuickBill"
]
let renderer = UIGraphicsPDFRenderer(bounds: bounds, format: format)
```

Algoritmo 3.4: Creación del lienzo A4

2. **Dibujar encabezados**
   Dentro del bloque *renderer.pdfData* { *ctx in ...* }, se escribe el título y los datos de empresa y cliente:

```
ctx.beginPage()
"INVOICE".draw(at: CGPoint(x: bounds.midX-40, y:36),
               withAttributes: titleAttr)

drawBlock(label: "Your␣company", lines: [businessName])
drawBlock(label: "Bill␣to", lines: [clientName])
```

Algoritmo 3.5: Dibujar encabezados

3. **Renderizar la tabla de productos**
   Se imprime la cabecera y, para cada elemento de *products*, se dibujan descripción, cantidad, precio unitario y total.

4. **Calcular importes**
   Al final se muestran subtotal, impuestos, descuentos y el TOTAL resaltado:

```
amountRow("Subtotal",  invoice.subtotal)
amountRow("Tax",       invoice.taxTotal)
amountRow("Discounts", invoice.discounts)
amountRow("TOTAL",     invoice.totalAmount, bold: true)
```

Algoritmo 3.6: Calcular importes

5. **Guardar en *Documents***
   El PDF resultante se escribe como *Invoice-¡id¿.pdf* en el sandbox de la app y se devuelve la URL para compartir o previsualizar.

Este enfoque permite al usuario generar la factura al instante, sin incurrir en costes adicionales. Para la numeración se reutiliza el *invoice.id* otorgado por Firestore, garantizando unicidad sin contadores globales.

Código completo: https://github.com/JuanCarlosAcostaPeraba/QuickBill-App/blob/main/iOS/QuickBill/Services/InvoicePDFBuilder.swift

3.2.4. Autenticación y permisos

La autenticación se resuelve con Firebase Auth usando correo y contraseña. La lógica de inicio de sesión se encapsula en el *SignInViewModel*, mientras que la interfaz se construye en *SignInView*. A grandes rasgos:

■ **SignInViewModel.swift**

- Propiedades *@Published* para *email*, *password*, estado de alerta y bandera *didSignIn*.
- Método *signIn()* que llama a *Auth.auth().signIn(withEmail:password:)* dentro de una *Task* / *await* y gestiona los posibles errores.

■ **SignInView.swift**

- Campos *TextField* y *SecureField* enlazados al ViewModel.
- Botón "Sign in" que ejecuta *await viewModel.signIn()*; si la autenticación tiene éxito se activa *auth.isSignedIn = true* (estado global mediante *AuthViewModel*).
- Enlace a la vista de registro y a la de "Forgot password".

Fragmento clave de autenticación:

```
@MainActor
func signIn() async {
    do {
        _ = try await Auth.auth()
            .signIn(withEmail: email, password: password)
        didSignIn = true
    } catch {
        alertMessage = error.localizedDescription
        showAlert = true
    }
```

```
}
```

Algoritmo 3.7: SignInViewModel.signIn()

Una vez autenticado, el UID se usa para filtrar datos, por ejemplo:

```
let uid = Auth.auth().currentUser?.uid ?? ""
Firestore.firestore()
  .collection("businesses")
  .document(businessId)
  .collection("invoices")
  .whereField("employeeId", isEqualTo: uid)
  .getDocuments { snapshot, error in ... }
```

Algoritmo 3.8: Filtrado de facturas por empleado

El código completo de la pantalla de inicio de sesión, así como del *SignInViewModel*, puede consultarse en GitHub: https://github.com/JuanCarlosAcostaPeraba/QuickBill-App/tree/main/iOS/QuickBill/Views/Authentication

3.2.5. Filtrado avanzado

Para mejorar la usabilidad, se implementó un sistema de filtrado avanzado en la lista de facturas. El usuario puede filtrar por cliente, estado, rango de fechas y rango de importe. Esta funcionalidad se encuentra en *HomeView* y se gestiona a través del *InvoiceListViewModel*.

```
var filteredInvoices: [Invoice] {
    invoices.filter { inv in
        (selectedStatus == .all || inv.status == selectedStatus) &&
        (
            searchText.isEmpty ||
            inv.companyName.lowercased().contains(searchText.lowercased()) ||
            dateFormatter.string(from: inv.issuedAt).contains(searchText) ||
            String(format: "%.2f", inv.totalAmount).contains(searchText)
        ) && (
            !enableDateFilter ||
            (
                (dateFrom == nil || inv.issuedAt >= dateFrom!) &&
                (dateTo   == nil || inv.issuedAt <= dateTo!)
            )
        ) && (
```

```
            !enableAmountFilter ||
            (
                (minTotal == nil || inv.totalAmount >= minTotal!) &&
                (maxTotal == nil || inv.totalAmount <= maxTotal!)
            )
        ) &&
        (selectedClientId == nil || inv.clientId == selectedClientId)
    }
}
```

Algoritmo 3.9: HomeView.swift - Filtrado avanzado

El usuario puede seleccionar el estado de la factura, introducir texto para buscar por cliente o fecha, y activar los filtros de fecha e importe. El resultado se actualiza dinámicamente en la vista.

3.2.6. UI/UX y pruebas en simulador

La interfaz de usuario se diseñó siguiendo las pautas de Apple para SwiftUI, priorizando la simplicidad y la usabilidad. Se utilizaron componentes estándar como *List*, *Form*, *NavigationView* y *Button* para construir una experiencia coherente.

Se priorizó la accesibilidad y la claridad visual mediante la asignación de colores distintivos a los diferentes estados de las facturas (verde para *paid*, lila para *pending* y rojo para *overdue*), así como la incorporación de animaciones suaves en las transiciones de pantalla. Para la visualización de productos, se optó por *LazyVGrid*, lo que permite una disposición adaptable y eficiente en distintos tamaños de pantalla.

Para verificar que todo funciona, se realizaron pruebas manuales en el simulador de iOS:

- Crear una nueva factura: abrir *AddInvoiceView*, rellenar campos, pulsar "Guardar" y comprobar que aparece en *HomeView*.

- Editar estado de factura: marcar una factura pendiente como “Paid” y verificar que cambia su color (verde) en la lista.

- Generar PDF: pulsar “Generar PDF” en *InvoiceDetailView* y verificar que se guarda en la carpeta *Documents* del simulador.

- Inicio de sesión: probar con credenciales correctas e incorrectas para validar la lógica de *Auth*.

A continuación se muestran algunas capturas de pantalla que ilustran estos flujos.

La Figura 3.2 muestra la pantalla de inicio de sesión de la aplicación iOS, mientras que la Figura 3.3 presenta la vista principal con el listado de facturas.

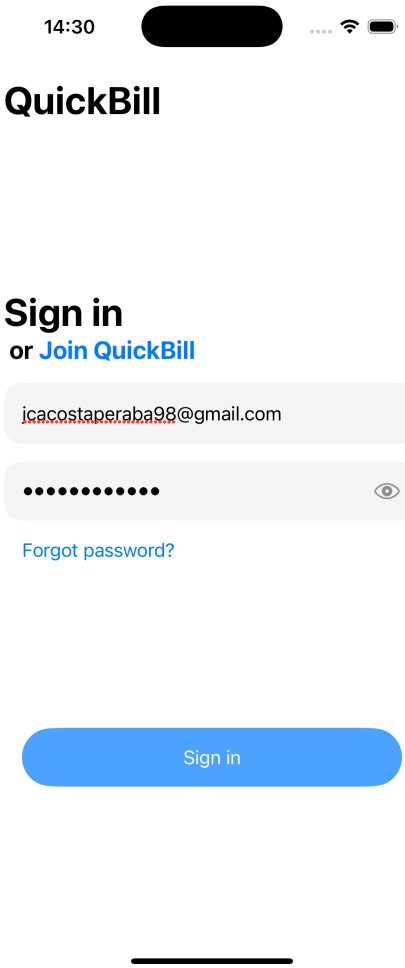

Ilustración 3.2: Pantalla de inicio de sesión

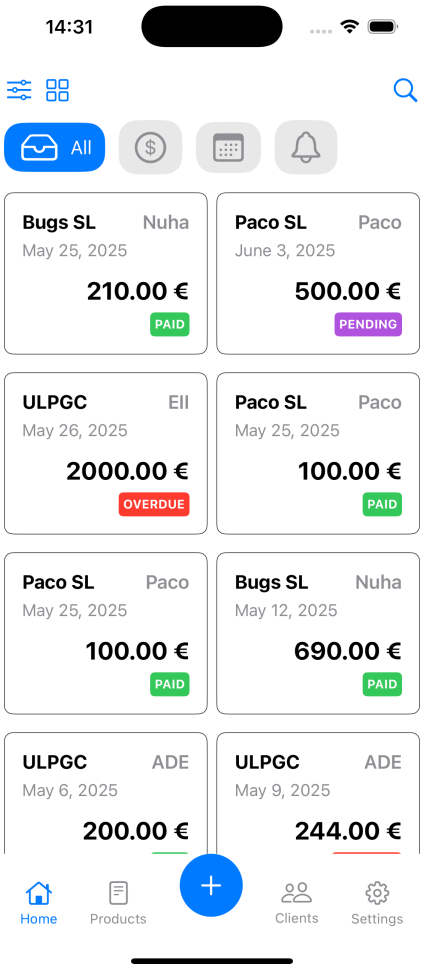

Ilustración 3.3: Vista principal con lista de facturas

La Figura 3.4 ilustra el formulario de creación de una nueva factura y la Figura 3.5 detalla la información de una factura ya registrada.

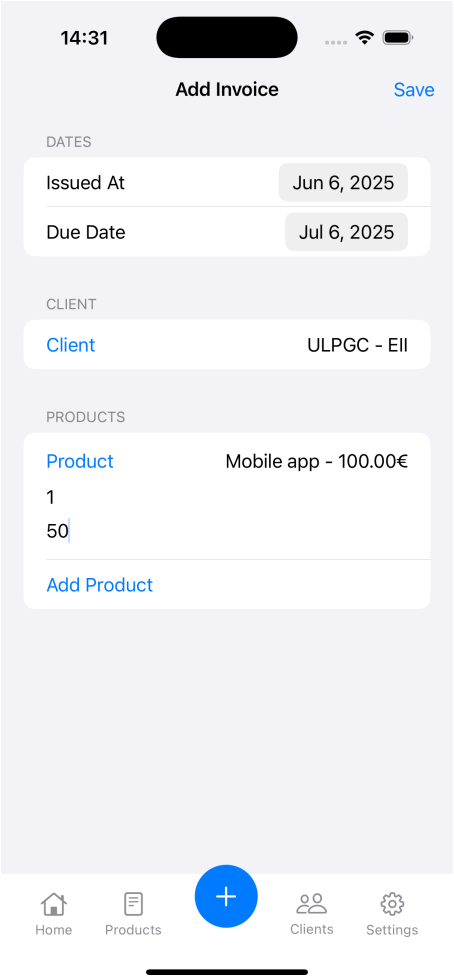

Ilustración 3.4: Formulario para añadir una nueva factura

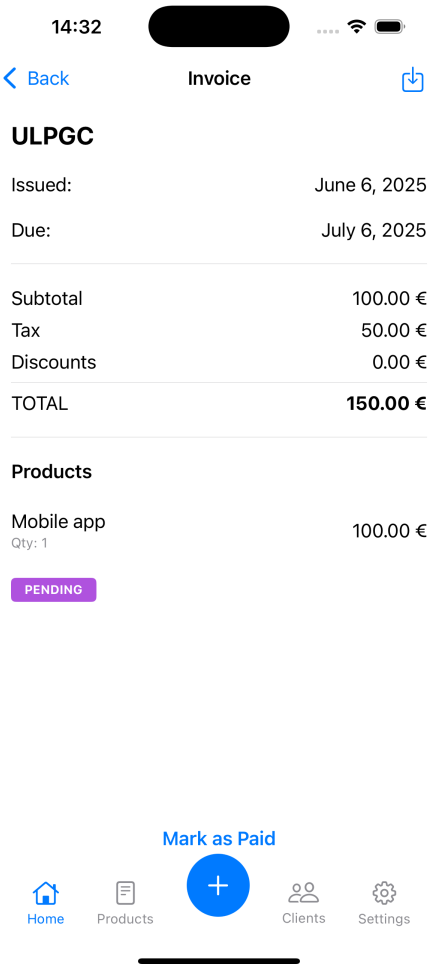

Ilustración 3.5: Detalles de una factura con opción de generar PDF

La Figura 3.6 muestra la interfaz de generación y guardado del PDF resultante.

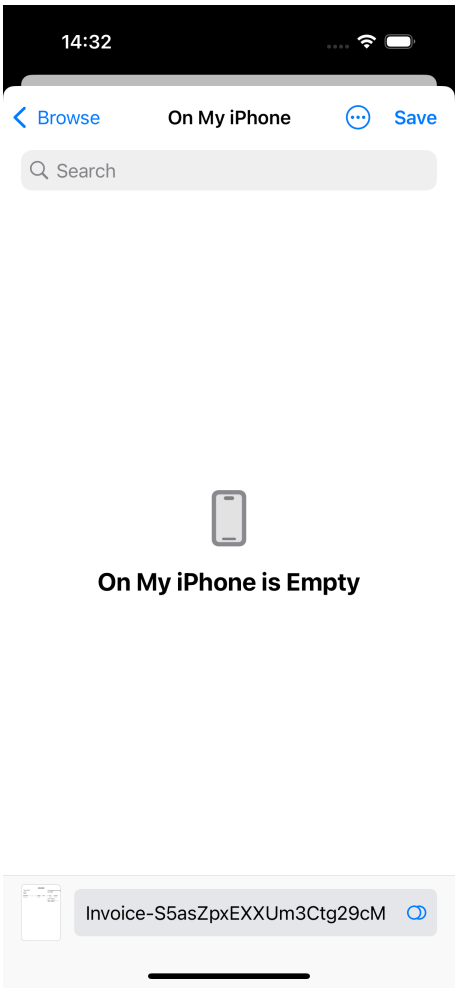

Ilustración 3.6: Generación del PDF de la factura

3.3. Implementación web

La parte web, aunque más ligera, ofrece tres funcionalidades clave: gráfica de pagos mensuales, listado de morosos y tabla con todas las facturas. A continuación se describe su estructura y lógica:

3.3.1. Estructura general del proyecto

El repositorio para el portal web sigue la convención estándar de Astro (Figura 3.7):

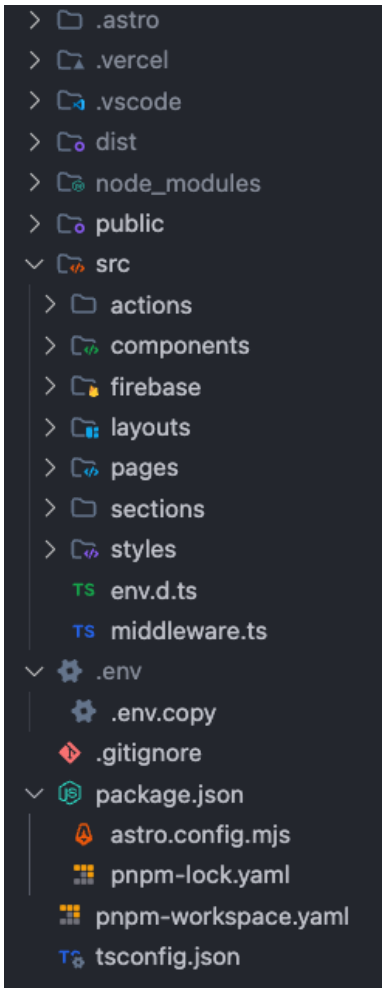

Ilustración 3.7: Estructura del proyecto Astro

- */src/pages*:
  - *index.astro*: página principal que importa los tres componentes.
  - *login.astro*: página de inicio de sesión para acceder al portal.
- */src/components*:

- *MoneyChart.astro*: renderiza la gráfica de barras usando *lightweight-charts*.

- *OverdueList.astro*: muestra tarjetas con clientes morosos y sus importes totales.

- *InvoiceTable.astro*: tabla con todas las facturas (cliente, fecha, importe, estado) con ZingGrid.

- *Header.astro*: barra de navegación con enlaces a las diferentes secciones y un botón de cierre de sesión.

- *ButtonToScrollTop.astro*: botón que permite volver al inicio de la página.

■ */src/firebase*:

- *firebase.ts*: inicializa Firebase con la configuración y exporta funciones e instancias de *firestore* y *auth*.

- *config.ts*: carga la configuración desde el archivo *.env* y exporta las variables necesarias.

3.3.2. Carga de datos y renderizado

Para cargar los datos de Firestore, se implementaron funciones asíncronas que se invocan en cada componente. A continuación se muestra cómo se obtienen todos los datos relevantes de un usuario a partir de su ID, que es el punto de partida para poblar las tablas y gráficas.

```
async function getAllDataForUser(userId: string) {
        const allData: any = {}

        // 1. get all businesses
        const businessesQuery = await query(collection(firestore, 'businesses'))
        const businessesSnapshot = await getDocs(businessesQuery)
```

```
        if (businessesSnapshot.empty) {
                console.log('No businesses found.')
                return []
        }

        // 2. filter all businesses where the user is in the employees collection
        const businesData = businessesSnapshot.docs.map(async (doc) => {
                const businessData = doc.data()
                const businesId = doc.id
                const employees = await getEmployeesBy(businesId)
                const isEmployee =
                        employees.filter((employee: any) => employee.userId === userId) || null
                if (!isEmployee || isEmployee.length === 0) {
                        return null
                }
                return {
                        id: businesId,
                        ...businessData,
                }
        })
        const businesses = (await Promise.all(businesData)).filter((b) => b !== null)
        if (businesses.length === 0) {
                console.log('No businesses found for this user.')
                return []
        }
        const businessId = businesses[0].id
        allData.businesses = businesses

        // 3. get all invoices that are inside the business collection
        const invoices = getInvoicesBy(businessId)
        allData.invoices = await invoices

        // 4. get all clients that are inside the business collection
        const clients = getClientsBy(businessId)
        allData.clients = await clients

        // 5. get all products that are inside the business collection
        const products = getProductsBy(businessId)
        allData.products = await products

        // 6. get all employees that are inside the business collection
        const employees = getEmployeesBy(businessId)
        allData.employees = await employees

        return allData
}
```

Algoritmo 3.10: Función para obtener todos los datos de un usuario a partir de su ID

Una vez se retorna el objeto *allData*, cada componente puede acceder a los datos necesarios del objeto, el cual se actualizará automáticamente cuando cambien los datos en Firestore.

```
function listAllInvoices(allData: any) {
    const invoiceGrid = document.querySelector('#invoiceGrid') as any
    if (!invoiceGrid) return
```

```
    const invoiceGridData = allData.invoices.map((invoice: any) => {
        const client = allData.clients.find(
            (client: any) => client.id === invoice.clientId
        )
        return {
            client: client.companyName,
            issued: new Date(invoice.issuedAt.seconds * 1000)
              .toLocaleDateString(
                'es',
                {
                    day: '2-digit',
                    month: '2-digit',
                    year: 'numeric',
                }
            ),
            total: `${invoice.totalAmount}${invoice.currency}`,
            status: invoice.status,
        }
    })
    invoiceGrid.data = invoiceGridData
}
```

Algoritmo 3.11: Función para listar todas las facturas en la tabla

Para el listado de morosos, *OverdueList.astro* filtra facturas cuyo estado sea "Overdue" y las agrupa por cliente.

```
function listOverdueClients(allData: any) {
        const overdueCards = document.querySelector('#overdueClientsCards')

        let dictOverdueClients: any = {}

        allData.clients.forEach((client: any) => {
                dictOverdueClients[client.id] = {
                        name: client.companyName,
                        amount: 0,
                        currency: '',
                }
        })

        allData.invoices.forEach((invoice: any) => {
                if (invoice.status === 'Overdue') {
                        const client = dictOverdueClients[invoice.clientId]
                        if (client) {
                                client.amount += invoice.totalAmount
                                client.currency = invoice.currency
                        }
```

```
                }
        })

        /* Inyectar HTML */
}
```

Algoritmo 3.12: Función para listar clientes morosos

3.3.3. Librerías externas

Para el portal web se utilizaron varias librerías externas que facilitan la creación de gráficos, tablas y la experiencia de usuario:

- **lightweight-charts**: Permite renderizar gráficas de barras y líneas de forma eficiente y responsiva [13]. Se emplea en el componente *MoneyChart.astro* para mostrar la evolución mensual de los pagos.

- **ZingGrid**: Framework para crear tablas interactivas y personalizables [14]. Se utiliza en *InvoiceTable.astro* para listar todas las facturas con ordenación, filtrado y paginación.

Estas librerías permiten construir una interfaz moderna y dinámica, reduciendo el esfuerzo de desarrollo y mejorando la experiencia del usuario final.

3.3.4. Despliegue del portal en Vercel

El portal web se despliega en Vercel siguiendo el flujo de integración continua:

1. Cada vez que se hace *push* a la rama *main*, Vercel detecta el cambio y ejecuta el comando *pnpm run build*.

2. Una vez compilado, Vercel optimiza los activos estáticos y publica el sitio

en una URL de producción.

La Figura 3.8 muestra el panel de Vercel con el estado del último despliegue, mientras que la Figura 3.9 presenta el portal ya publicado.

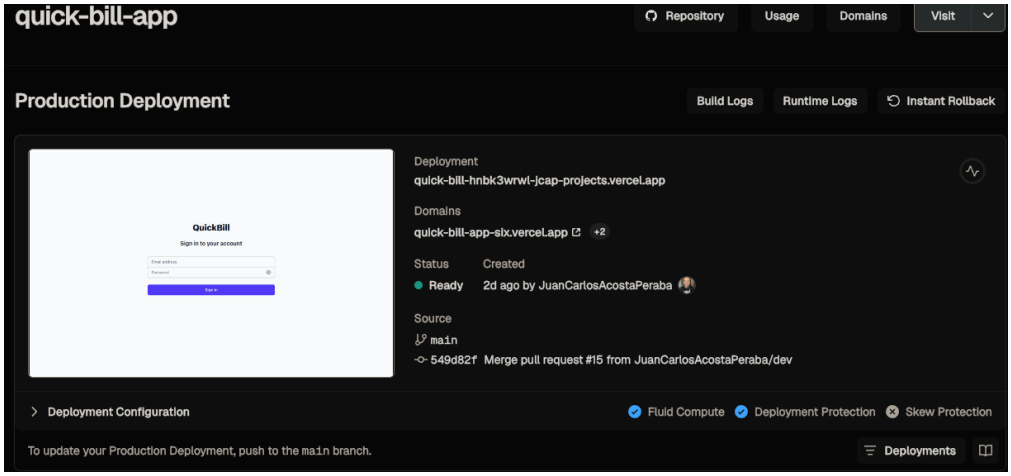

Ilustración 3.8: Panel de Vercel con el estado del último despliegue

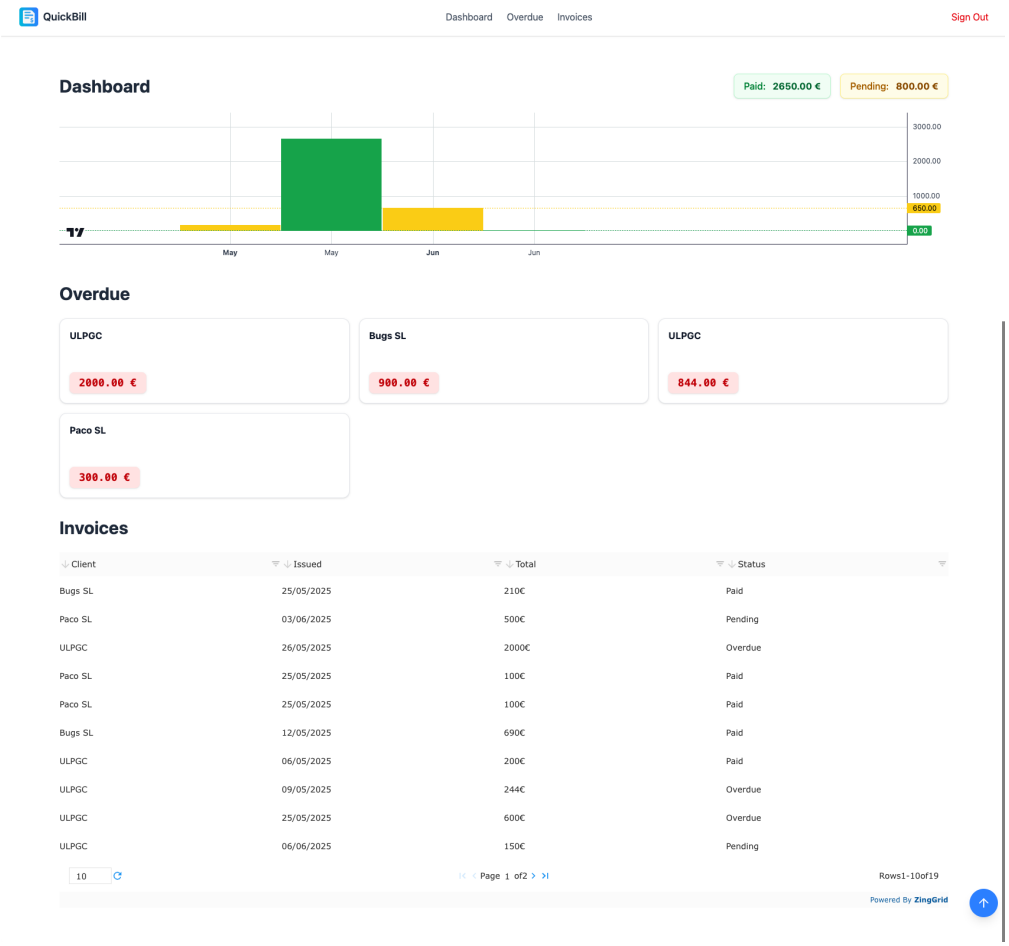

Ilustración 3.9: Portal web desplegado en Vercel

El portal web está disponible en la URL https://quick-bill-app-six.vercel.app/, donde los usuarios pueden iniciar sesión y acceder a las funcionalidades descritas. Algunas cuentas de prueba que se pueden usar son:

- `jcacostaperaba98@gmail.com` / `qwerty123456`
- `juancarlos@jcap.com` / `estoEsUnT3st`

Capítulo 4

Resultados y trabajo futuro

4.1. Resultado

La versión actual de la aplicación cumple con los objetivos esenciales planteados en el Capítulo 1:

- Gestión completa de facturas: creación, edición, filtrado por estado (*Paid* / *Pending* / *Overdue*) y eliminación.
- Catálogo de clientes y productos totalmente operativos.
- Generación de PDF directamente en el dispositivo iOS, con numeración única basada en el `invoice.id` de Firestore.
- Portal web accesible con tres vistas clave: gráfica de pagos mensuales, listado de morosos y tabla completa de facturas.
- Despliegue continuo en Vercel, con certificados TLS automáticos.

4.1.1. Desviaciones respecto al plan inicial

- **Modo offline**: no llegó a implementarse por limitación de tiempo.
- **App Android**: sólo se completó el esqueleto de proyecto.
- **Cloud Functions y Cloud Storage**: sustituidos por generación y almacenamiento local de PDFs para evitar costes.

Aun así, el núcleo funcional se encuentra implementado y operativo, permitiendo emitir facturas y visualizar su estado desde cualquier lugar con conexión.

4.2. Conclusión

El desarrollo de este proyecto ha puesto de manifiesto el esfuerzo que implica diseñar una plataforma coherente en todas sus capas. Desde la definición del modelo de datos en Firestore hasta la disposición final de cada botón en la pantalla, cada decisión ha requerido un equilibrio entre simplicidad de uso y robustez interna.

Arquitectura de la base de datos

El esquema jerárquico (*businesses/invoices/clients/employees/products*) resultó sencillo de consultar y asegurar, pero elegir una estructura que escalara sin duplicar datos supuso múltiples iteraciones. Hubo que considerar, por ejemplo, qué nivel de anidación permitía aplicar reglas de seguridad sin penalizar el rendimiento; o cómo traducir relaciones $n:1$ (una factura pertenece a un único cliente) a un modelo sin uniones nativas. El resultado es una base de datos que mantiene la coherencia y aísla correctamente la información de cada empresa.

Diseño minimalista de la app

Mostrar solo lo imprescindible en cada pantalla exigió agrupar operaciones y reducir opciones visibles. Se descartaron menús redundantes y se reorganizaron flujos para que el usuario pudiese emitir una factura en menos de un minuto: seleccionar cliente, añadir líneas de producto y pulsar "Guardar". La navegación basada en pestañas (*Home*, *Invoices*, *Clients*, *Products*, *Settings*) demostró ser la forma más intuitiva de acceso rápido, y el uso de colores (verde para *Paid*, lila para *Pending*, rojo para *Overdue*) evita distracciones a la vez que señala el estado crítico de cada pago.

Lecciones aprendidas

- Separar la lógica de negocio en ViewModels y servicios facilita testear y añadir nuevas funciones sin efectos colaterales sobre la UI.

- Definir las reglas de Firestore antes de implementar llamadas previene fugas de seguridad y reduce tiempo de depuración.

- El minimalismo en interfaz no implica menos funcionalidad; implica esconder la complejidad hasta que sea necesaria.

En conjunto, la plataforma cumple su objetivo principal: ofrecer una solución de facturación ágil y comprensible. El trabajo invertido en la arquitectura de datos y la experiencia de usuario sienta una base sólida para continuar evolucionando la aplicación.

4.3. Trabajos futuros

Como líneas de evolución se plantean diversas mejoras que consolidarían la plataforma y la harían competitiva frente a soluciones consolidadas:

1. **Modo offline completo**. Implementar caché local persistente (Core Data + Cloud Kit o SQLite) y un mecanismo de sincronización diferencial con Firestore. De esta forma, el usuario podría emitir facturas sin conexión y la reconciliación de cambios se ejecutaría en segundo plano cuando vuelva la red.

2. **OCR y lectura de códigos QR**. Integrar Vision Kit para capturar facturas de proveedor o tickets y extraer automáticamente importes, fechas e IVA. Además, permitir la lectura de códigos QR europeos (Facturae o Factur-X) para precargar datos de cliente y líneas de producto.

3. **Exportación avanzada**. Añadir exportación a CSV / Excel y conexión directa con herramientas de contabilidad (A3, Contasol, Holded) mediante APIs públicas. Esto facilitará auditorías y conciliaciones bancarias.

4. **Firma digital**. Incluir un flujo opcional para firmar digitalmente la factura, asegurando integridad y no repudio.

5. **Automatización de envío**. Configurar plantillas de correo y adjuntar el PDF firmado directamente desde la app, con opción de recordatorios automáticos a los 7 días y el día de vencimiento.

6. **Notificaciones proactivas**. Enviar notificaciones push o correos cuando una factura vence, cuando el cliente visualiza el PDF o cuando se acerca el umbral de almacenamiento del plan gratuito.

7. **Refactorización Android**. Retomar el esqueleto en Kotlin y completar la paridad funcional con iOS, usando Jetpack Compose y el mismo

esquema de Firestore.

8. **Dashboard avanzado**. Añadir nuevos gráficos comparativos y funciones de contabilidad.

Estas mejoras ampliarían el alcance funcional y aportarían valor añadido a autónomos y pymes.

Bibliografía

[1] Ana Petrovic. From clay to cloud: A deep dive into the history of invoicing, 2024. URL https://clockify.me/blog/tracking-time/history-of-invoicing/. Consultado el 17–05–2025.

[2] Ley 18/2022, de 28 de septiembre, de creación y crecimiento de empresas, 2022. URL https://www.boe.es/buscar/act.php?id=BOE-A-2022-15818. Consultado el 17–05–2025.

[3] Deloitte. 2023 mid-market technology trends report. Technical report, Deloitte, 2023. URL https://www2.deloitte.com/content/dam/Deloitte/us/Documents/us-Deloitte-MMTS-report.pdf. Consultado el 17–05–2025.

[4] Escuela de Ingeniería Informática, ULPGC. Grado en ingeniería informática – rev03, 2024. URL https://eii.ulpgc.es/sites/default/files/2024-05/20240229%20Grado%20en%20Ingenier%C3%ADa%20Inform%C3%A1tica%20-%20rev03.pdf. Consultado el 17–05–2025.

[5] Apple Inc. Swift language guide, 2025. URL https://docs.swift.org/swift-book/documentation/the-swift-programming-language. Consultado el 22–05–2025.

[6] JetBrains. Kotlin documentation, 2025. URL https://kotlinlang.org/docs/home.html. Consultado el 22–05–2025.

[7] Google. Android studio 2024.3.2 release notes, 2024. URL https://developer.android.com/studio/releases. Consultado el 22–05–2025.

[8] Astro Technology Company. Astro documentation, 2025. URL https://docs.astro.build. Consultado el 22–05–2025.

[9] pnpm Contributors. pnpm documentation, 2025. URL https://pnpm.io. Consultado el 22–05–2025.

[10] Google Firebase. Firebase documentation, 2025. URL https://firebase.google.com/docs. Consultado el 22–05–2025.

[11] Vercel Documentation. Vercel documentation. https://vercel.com/docs, 2025. Consultado el 06–06–2025.

[12] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2 edition, 2025. URL https://git-scm.com/book/en/v2.

[13] TradingView. Lightweight charts. https://www.tradingview.com/lightweight-charts/, 2025. Consultado en junio de 2025.

[14] ZingGrid. Zinggrid documentation. https://www.zinggrid.com/, 2025. Consultado en junio de 2025.

Glosario

Astro Framework web basado en la arquitectura de islas. Genera HTML estático en el servidor y añade JavaScript solo en los componentes que lo necesitan, lo que mejora el rendimiento y la accesibilidad.

B2B Siglas de *Business to Business*. Transacciones comerciales de empresa a empresa, a diferencia del modelo B2C dirigido al consumidor final. Incluye cadenas de suministro, distribuidores y servicios profesionales..

ERP *Enterprise Resource Planning*. Sistema que integra y automatiza procesos empresariales (finanzas, compras, producción, inventario, RR. HH.) en una base de datos central para ofrecer trazabilidad y visión en tiempo real..

RGPD Reglamento (UE) 2016/679 de protección de datos personales que establece principios de tratamiento, derechos de los interesados y obligaciones para responsables y encargados..

SaaS Modelo *Software as a Service*: la aplicación se aloja en la nube del proveedor y se consume vía Internet mediante suscripción. Evita instalaciones locales, actualiza de forma automática y escala según la demanda..