Sistema TPV para la gestión de un bar

TRABAJO DE FIN DE GRADO

CFGS MULWEB 2

Unai Pastor Martínez

ÍNDICE

Índice

Resumen

Este **Terminal de Punto de Venta (TPV)** es una aplicación de escritorio desarrollada con **C#, .NET Framework** y **WinForms**. Utiliza una arquitectura en tres capas para separar la presentación, la lógica de negocio y el acceso a datos.

El sistema cuenta con:

Un módulo de administración, donde los gestores podrán generar facturas, administrar usuarios y gestionar productos de forma centralizada.

Un módulo para empleados, donde podrán iniciar sesión con su ubicación y registrar cada venta de manera sencilla.

Para garantizar la disponibilidad de los datos, el sistema utilizará **PostgreSQL** como base de datos principal en la nube (base de datos alojada en **Neon.tech**), permitiendo que los cambios en precios y productos se apliquen instantáneamente en todos los dispositivos. Además, se implementará **SQLite** para almacenamiento local en caso de fallos en la conexión.

He elegido C# y .NET Framework por varias razones:

- **Entorno robusto y escalable:** .NET ofrece un ecosistema maduro con herramientas avanzadas para el desarrollo de aplicaciones empresariales.
- **Arquitectura en tres capas:** Permite una mejor organización del código y facilita el mantenimiento.
- **Seguridad y autenticación:** Incluye **bibliotecas especializadas para el hashing** de contraseñas y la gestión de accesos.
- **Comunidad y soporte:** .NET cuenta con una gran comunidad de desarrolladores y documentación extensa, lo que facilita la resolución de problemas y futuras mejoras.
- Este sistema busca **mejorar la gestión de TPV en establecimientos de hostelería**, ofreciendo una **solución accesible, segura y fácil de actualizar**.

Para la interfaz de usuario, se ha elegido WinForms por ser una tecnología madura, fácil de implementar y con un rendimiento óptimo para aplicaciones de escritorio. Su integración con .NET permite un desarrollo rápido y eficiente, además de contar con una amplia comunidad de soporte.

Palabras clave
C#

.NET

PostgreSQL

TPV

Ventas

WinForms

Neon.tech

SQLite

Abstract

This **Point Of Seal (POS)** system is a desktop application developed with **C#, .NET Framework, and WinForms**. It uses a **three-layer** architecture to separate the presentation, business logic, and data access.

The system includes:

An administration module, where managers can generate invoices, manage users, and centrally handle products.

An employee module, where users can log in with their location and easily register each sale.

To ensure data availability, the system will use **PostgreSQL** as the primary cloud database, allowing price and product changes to be instantly applied across all devices. Additionally, **SQLite** will be implemented for local storage in case of connection failures.

I have chosen C# and .NET Framework for several reasons:

- **Robust and scalable environment:** .NET provides a mature ecosystem with advanced tools for enterprise application development.
- **Three-layer architecture:** Enables better code organization and facilitates maintenance.
- **Security and authentication:** Includes specialized **libraries for password hashing** and access management.
- **Community and support:** .NET has a large developer community and extensive documentation, making problem-solving and future improvements easier.
- This system aims to improve POS management in hospitality establishments by offering an accessible, secure, and easy-to-update solution.

For the user interface, WinForms was chosen as it is a mature, easy-to-implement technology with optimal performance for desktop applications. Its integration with .NET allows for fast and efficient development, along with extensive community support.

KEY WORDS
C#

.NET

PostgreSQL

TPV

Sales

WinForms

Neon.tech

SQLite

1. Memoria del proyecto

1.1 Justificación del proyecto

Mi propia **TPV** es un proyecto que surge como resultado de mi **experiencia trabajando como camarero** en diversos establecimientos. A lo largo de estos años, he utilizado distintos tipos de TPV e incluso he trabajado sin ellos debido a las molestias que puede suponer su instalación o la dificultad de uso.

Uno de los mayores inconvenientes que he observado es la **falta de flexibilidad en la configuración** de precios. Cambiar el precio de un producto requiere acceder a un menú poco intuitivo que rara vez se explica a los empleados.

Otro inconveniente es la mala **interfaz** que suelen tener las TPV lo cual hace que sea más propenso que ocurran errores lo cual le acaba perjudicando al dueño del negocio.

Por ello he decidido hacer **mi propia TPV** con todas las mejoras posibles para que su uso sea intuitivo, rápido y eficiente.

1.2 Objetivos del proyecto

El objetivo de este proyecto es **poner en práctica** todos los conocimientos aprendidos durante este año y aprender a utilizar nuevas herramientas.

Otro objetivo es **mejorar las TPV que he utilizado** a lo largo de mi experiencia laboral haciendo una TPV rápida, intuitiva y conectada a la red haciendo que no con un simple click se puedan añadir productos, editar y eliminar productos.

También tener una **correcta gestión** de usuarios por zonas para que no sea necesario ir hasta el establecimiento para agregar al usuario.

2. Aspectos teóricos

2.1.1 ¿Qué es una TPV?

Un Terminal de Punto de Venta (TPV) es un sistema informático utilizado en comercios y establecimientos de hostelería para gestionar transacciones de venta, control de stock y administración de productos y usuarios. Su función principal es facilitar el registro y procesamiento de pagos, aunque en la actualidad muchas TPV incluyen funcionalidades avanzadas como:

- **Emisión de facturas y tickets:** Generación automática de comprobantes de compra.
- **Gestión de empleados:** Control de turnos y registro de ventas por usuario.
- **Informes y análisis:** Registro de ventas diarias
- **Integración con bases de datos:** Sincronización en la nube o almacenamiento local para garantizar la disponibilidad de los datos.
- El uso de un TPV **optimiza la gestión del negocio**, reduciendo errores manuales y mejorando la experiencia del cliente y la eficiencia operativa.

2.1.2 Roles de usuario en la TPV

Para garantizar un correcto funcionamiento y organización del sistema, la TPV implementa distintos roles de usuario con permisos específicos:

Administrador:

- Gestiona productos y precios.
- Administra los usuarios del sistema.
- Genera y revisa facturas.
- Consulta estadísticas y reportes de ventas.
- Consulta el acceso de los administradores

Empleado:

- Inicia sesión con su ubicación.
- Registra ventas y cobra a los clientes.
- Consulta productos y precios.
- Accede al historial de ventas.

2.2 ¿Qué es PostgreSQL?

PostgreSQL es un sistema de gestión de bases de datos relacional (**RDBMS**) de código abierto, reconocido por su potencia, flexibilidad y fiabilidad. Diseñado para manejar grandes volúmenes de datos de manera eficiente, PostgreSQL es ampliamente utilizado en entornos empresariales y aplicaciones críticas.

Principales características de PostgreSQL

- **Código abierto y gratuito:** No requiere licencias y cuenta con una comunidad activa de desarrolladores.
- **Alta fiabilidad y estabilidad**: Diseñado para garantizar la integridad de los datos con mecanismos como transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad).
- **Escalabilidad**: Soporta grandes volúmenes de datos y permite configuraciones distribuidas y replicación de bases de datos.
- **Seguridad avanzada**: Incluye autenticación, cifrado de datos y gestión de permisos de usuarios.
- **Compatibilidad con SQL estándar**: Permite realizar consultas complejas y optimizar el rendimiento mediante índices, vistas y procedimientos almacenados.

2.2.1 Uso de PostgreSQL en la TPV

En este proyecto**, PostgreSQL** se utiliza como la **base de datos principal en la nube**, permitiendo:

Almacenamiento **centralizado** de información sobre ventas, productos y usuarios.

Actualización de información, reflejándose en **todos los dispositivos conectados**.

2.3 ¿Qué es .Net?

.NET es un marco de trabajo **(framework**) de código abierto desarrollado por Microsoft, diseñado para construir aplicaciones multiplataforma. Permite a los desarrolladores crear aplicaciones para una amplia variedad de plataformas, como Windows, macOS, Linux, y dispositivos móviles. .NET proporciona un conjunto de herramientas, bibliotecas y lenguajes de programación que facilitan el desarrollo de aplicaciones de manera eficiente.

Principales características de .NET

- **Multiplataforma:** .NET permite desarrollar aplicaciones que pueden ejecutarse en diferentes sistemas operativos sin necesidad de modificaciones importantes en el código.
- **Lenguajes de programación compatibles:** Aunque .NET está optimizado para C#, también es compatible con otros lenguajes como F# y Visual Basic.
- **Entorno de ejecución común:** El Common Language Runtime (CLR) de .NET ejecuta las aplicaciones y maneja tareas como la gestión de memoria y el control de excepciones.

- **Bibliotecas y frameworks integrados:** Incluye bibliotecas estándar y frameworks como ASP.NET para aplicaciones web, WinForms para aplicaciones de escritorio, y Entity Framework para el acceso a bases de datos.
- **Seguridad y rendimiento:** Ofrece un alto nivel de seguridad y un entorno optimizado para aplicaciones de alto rendimiento, con capacidades de manejo de excepciones y concurrencia.

2.3.1 Uso de .NET en la TPV

En este proyecto, .NET Framework se ha utilizado para el desarrollo de la TPV debido a sus múltiples ventajas:

- **Estructura sólida y escalable:** Proporciona herramientas avanzadas que facilitan el desarrollo de aplicaciones empresariales de gran escala.
- **Arquitectura en tres capas:** La separación de la presentación, lógica de negocio y acceso a datos se implementa fácilmente en .NET, lo que mejora la organización del código y facilita el mantenimiento.
- **Seguridad y autenticación:** .NET cuenta con bibliotecas especializadas para implementar funciones de seguridad, como el hashing de contraseñas y la gestión de accesos, asegurando que los datos sean protegidos de manera eficiente.
- **Amplia comunidad de soporte:** .NET tiene una gran comunidad de desarrolladores y una amplia documentación, lo que facilita la resolución de problemas y futuras mejoras del sistema.

2.4 ¿Qué es SQLite?

SQLite es un sistema de gestión de bases de datos relacional (**RDBMS**) ligero, de código abierto y altamente **eficiente**, que se utiliza principalmente para aplicaciones que requieren **almacenamiento local** de datos. A diferencia de otros sistemas de bases de datos más complejos, SQLite no requiere un servidor de base de datos separado, ya que almacena la base de datos directamente en un archivo en el sistema de archivos local.

Principales características de SQLite:

- **Ligereza y simplicidad:** SQLite es muy ligero y fácil de integrar en aplicaciones sin necesidad de configuraciones complejas, lo que lo hace ideal para aplicaciones de escritorio y móviles.
- **Sin servidor:** No requiere un servidor o proceso separado, ya que las aplicaciones acceden a la base de datos directamente a través de una biblioteca de enlace dinámico (DLL).
- **Compatibilidad con SQL estándar:** Utiliza SQL para gestionar y manipular los datos, lo que facilita su integración y el uso de consultas estándar.
- **Transacciones ACID:** A pesar de ser ligero, SQLite soporta transacciones completas con propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que garantiza la integridad de los datos.

- **Portabilidad:** Almacena los datos en un único archivo, lo que facilita su portabilidad y respaldo.

2.4.1 Uso de SQLite en la TPV:

En este proyecto, SQLite se utilizará para proporcionar **almacenamiento local** en caso de que se produzca **una falla en la conexión a la base de datos principal (PostgreSQL)** y para garantizar la rapidez a la hora de hacer ventas. Esto garantiza que los empleados puedan seguir registrando ventas y realizando otras operaciones, aunque no haya conexión a Internet.

El uso de SQLite permite que el sistema sea más robusto, garantizando la disponibilidad de la información y ofreciendo una solución eficiente para el almacenamiento local de datos, sin la necesidad de un servidor adicional.

2.5 ¿Qué es C#?

C# (C-Sharp) es un lenguaje de programación moderno, desarrollado por Microsoft como parte del .NET Framework. Está diseñado para ser sencillo, potente y orientado a objetos, permitiendo la creación de aplicaciones robustas y eficientes. C# se utiliza ampliamente en el desarrollo de aplicaciones de escritorio, aplicaciones web, aplicaciones móviles y videojuegos.

Principales características de C#:

- **Orientado a objetos**: C# es un lenguaje completamente orientado a objetos, lo que facilita la estructuración del código y el mantenimiento de aplicaciones grandes y complejas.
- **Sintaxis clara y concisa**: Su sintaxis es similar a otros lenguajes populares como Java y C++, lo que facilita el aprendizaje y la transición de desarrolladores con experiencia en otros lenguajes.
- **Gestión automática de memoria**: C# utiliza un recolector de basura (garbage collector) que gestiona la memoria automáticamente, ayudando a evitar errores comunes de gestión de memoria.
- **Soporte para LINQ**: C# incluye características avanzadas como Language Integrated Query (LINQ), que permite realizar consultas sobre colecciones de datos de manera eficiente y legible.
- **Compatibilidad multiplataforma**: Gracias a .NET Core, C# es capaz de ejecutarse en diferentes plataformas, incluyendo Windows, macOS y Linux.
- **Rendimiento optimizado**: C# está diseñado para ofrecer un alto rendimiento, especialmente cuando se combina con el entorno de ejecución Common Language Runtime (CLR) de .NET.

2.5.1 Uso de C# en la TPV:

En este proyecto, C# es el lenguaje principal utilizado para el desarrollo de la TPV debido a sus múltiples ventajas:

- **Desarrollo eficiente y rápido**: La sintaxis moderna y las herramientas integradas de Visual Studio permiten desarrollar la aplicación de manera ágil y con menos errores.
- **Integración con .NET Framework**: C# se integra perfectamente con .NET Framework, lo que facilita el uso de bibliotecas avanzadas, la creación de interfaces de usuario con WinForms y el acceso a bases de datos.
- **Seguridad**: C# proporciona soporte para la implementación de seguridad y autenticación robusta, como el hashing de contraseñas y la gestión de accesos, lo cual es fundamental para proteger los datos sensibles en la TPV.
- **Escalabilidad**: Al estar basado en una arquitectura modular, las aplicaciones desarrolladas con C# pueden escalar fácilmente y adaptarse a las necesidades de negocio en crecimiento.

Gracias a la robustez y flexibilidad de C#, el desarrollo de la TPV se realiza de manera eficiente, asegurando una solución fiable y fácil de mantener.

2.6 Estado del arte

En los últimos años, los sistemas TPV (Terminal Punto de Venta) han evolucionado significativamente para adaptarse a las nuevas necesidades del comercio moderno. Tradicionalmente, estos sistemas estaban limitados a funciones básicas de registro de ventas, pero hoy en día ofrecen una amplia gama de funcionalidades: gestión de inventario, control de usuarios, análisis de ventas, generación de informes y conexión con bases de datos locales o en la nube.

Diversas soluciones comerciales ofrecen plataformas TPV completas como **TPV Center** o **TPV Escola,** que integran hardware y software con interfaces intuitivas y soporte para pagos electrónicos. Sin embargo, estas soluciones suelen estar ligadas a modelos de suscripción o a costes elevados y son bastante difíciles de entender, lo cual no siempre resulta viable para pequeños negocios o bares locales que buscan sistemas más adaptados a sus necesidades concretas.

Desde el punto de vista tecnológico, muchas soluciones TPV modernas han optado por el desarrollo utilizando frameworks como **React, Angular, Java Swing/JavaFX** o plataformas híbridas como **Flutter**. No obstante, sigue existiendo un nicho importante para aplicaciones de escritorio en entornos Windows, especialmente cuando se requiere una integración sencilla con hardware local y no se dispone de conexión constante a internet.

En este contexto, el uso de **C#** y **WinForms sobre .NET Framework** representa una opción sólida y estable para el desarrollo de aplicaciones de escritorio. WinForms, aunque no tan moderno como **WPF o MAUI**, sigue siendo ampliamente utilizado en el sector industrial y comercial por su simplicidad y rapidez de desarrollo.

La integración con SQLite como sistema gestor de base de datos local permite una solución ligera, sin necesidad de estar conectado a internet en todo momento.

Entre las alternativas de código abierto y desarrollos académicos, existen múltiples proyectos en **GitHub** que replican funcionalidades básicas de un TPV utilizando tecnologías similares. Estos proyectos suelen centrarse en la venta de productos, el control de stock y la emisión de facturas. No obstante, pocos abordan el ciclo completo de gestión de datos, interfaz gráfica personalizada y registro persistente de las operaciones diarias de forma modular y extensible, como se plantea en este trabajo.

Asimismo, plataformas más amplias como **Odoo** también ofrecen módulos de punto de venta integrados dentro de su ecosistema ERP. Odoo destaca por su enfoque modular, su código abierto y la posibilidad de personalización, aunque su **complejidad técnica** y requisitos de infraestructura lo hacen menos accesible para pequeños establecimientos sin conocimientos técnicos avanzados.

Este **Trabajo Fin de Grado** se inscribe dentro de esta línea, proponiendo una solución **adaptada** a un entorno concreto (bar o cafetería), implementada con tecnologías maduras, gratuitas y de fácil despliegue. La intención no es competir con soluciones comerciales, sino ofrecer una **alternativa sencilla y eficiente** que pueda ser adoptada por pequeños comercios o ampliada como base para futuros desarrollos.

3. Planificación

Para la planificación del proyecto he decidido realizar jornadas de 2 horas diarias dependiendo de la cantidad de tareas por hacer y el tiempo del sprint restante

3.1 Riesgos posibles

- Problemas personales: Enfermedad, accidente, cambios en el entorno laboral o académico que repercutan en el número de horas a dedicar en el desarrollo proyecto, etc...
- Errores en la estimación de fechas, provocados por la falta de un horario estricto y fijo para el proyecto o a la inexperiencia en algunos de los campos que abarca el proyecto.
- Desconocimiento de alguna herramienta necesaria para el desarrollo del proyecto.

3.2 Actuaciones previstas ante los riesgos

- Problemas personales: Como el equipo de desarrollo está compuesto únicamente por el alumno, la planificación de fechas y horas dedicadas al proyecto se verá afectada.
- Errores en la estimación de fechas: Se documentará adecuadamente el error de la estimación y se replanificarán las tareas si es conveniente.
- Falta de viabilidad en el proyecto: Esta situación debe evitarse a toda costa. Para ello es fundamental realizar un buen análisis y un estudio previo.

Si se produce la aparición de alguna de las incidencias citadas anteriormente o de otras que no se hayan contemplado, dadas las características del proyecto podría producirse un impacto en la duración del mismo. Ante cualquier incidencia, nos veremos obligados a realizar un esfuerzo por aumentar el número de horas diarias dedicadas al desarrollo del proyecto.

3.3 Presupuesto

Dado que utilizo un alojamiento para la base de datos gratuito (neon.tech), el equipo de desarrollo está compuesto por solamente el alumno (Unai Pastor Martínez) y se va a realizar en el equipo del propio alumno, este proyecto no tiene ningún coste.

3.4 Entregables

- Memoria del proyecto.
- Producto final. La plataforma completa se compondrá de:
    - Aplicación de escritorio.
    - Listado de pruebas de integración.
    - Manuales de usuario.

3.5 Metodologías ágiles

Para este proyecto he decidido utilizar **metodologías ágiles**. Estoy utilizando sprints de un tiempo de dos semanas en el cual tengo que hacer las tareas propuestas. Para gestionar los Sprints estoy utilizando una tabla **Kanban en GitHub**, que me permite organizar y visualizar el flujo de trabajo. Este tipo de tablero divide las tareas en columnas según su estado, como “Por hacer”, “En progreso” y “Hecho”, lo que facilita el seguimiento del avance y la gestión de prioridades de manera clara y visual.

3.5.1 Sprint 1

Fecha:4-18 de marzo de 2025

Para mi primer sprint me he propuesto realizar el registro y el login tanto de usuario como de admin, comenzar a documentar (memoria) y crear la base de datos PostgreSQL.

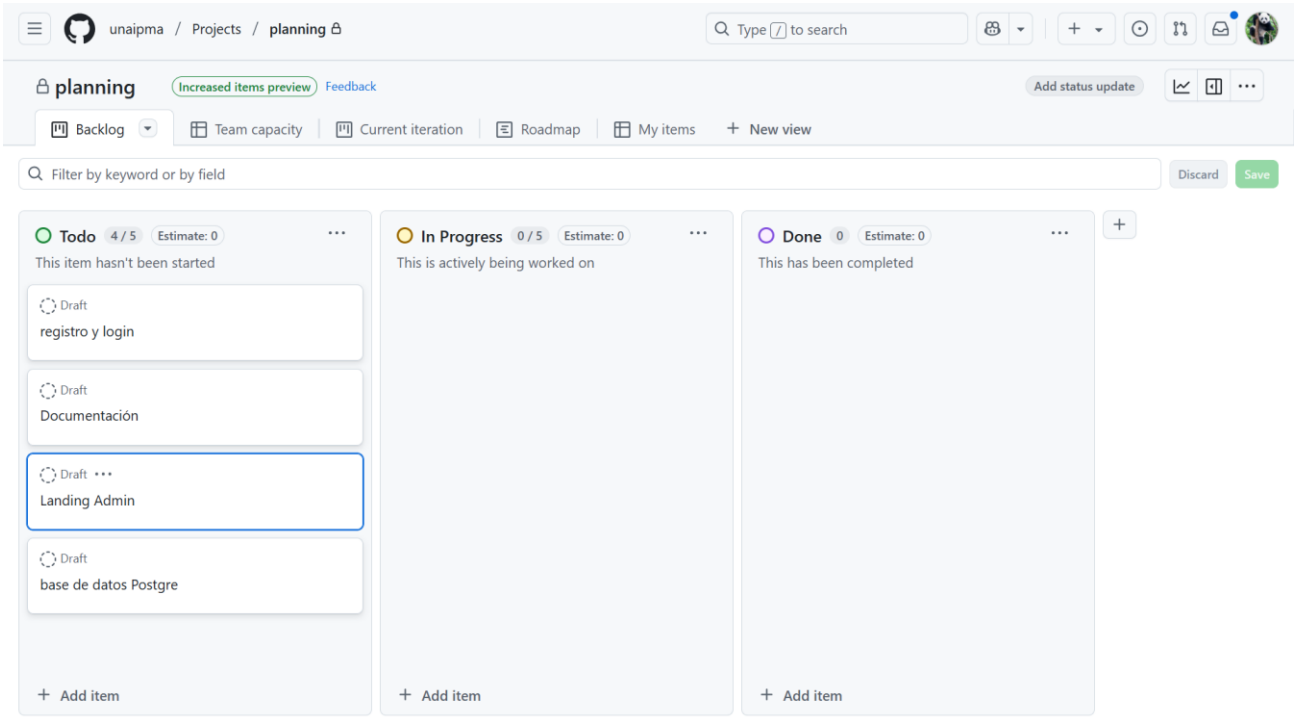

Figura 1 Tablero Kanban Sprint 1

3.5.2 Sprint 2

Fecha:18 de marzo al 1 de abril de 2025

Este es el aspecto de mi tablero Kanban al finalizar mi primer sprint.

Para el segundo sprint he propuesto realizar toda la funcionalidad del menú administrador (gestión de facturas, productos y usuarios), hacer el diseño gráfico y la explicación de la base de datos y comenzar la interfaz del usuario y seguir documentando, ya que es algo que voy a hacer a lo largo de todo el proyecto.

He conseguido finalizar con el tiempo esperado el registro, el login, el landing de administrador y crear la base de datos PostgreSQL alojada en Neon.tech.

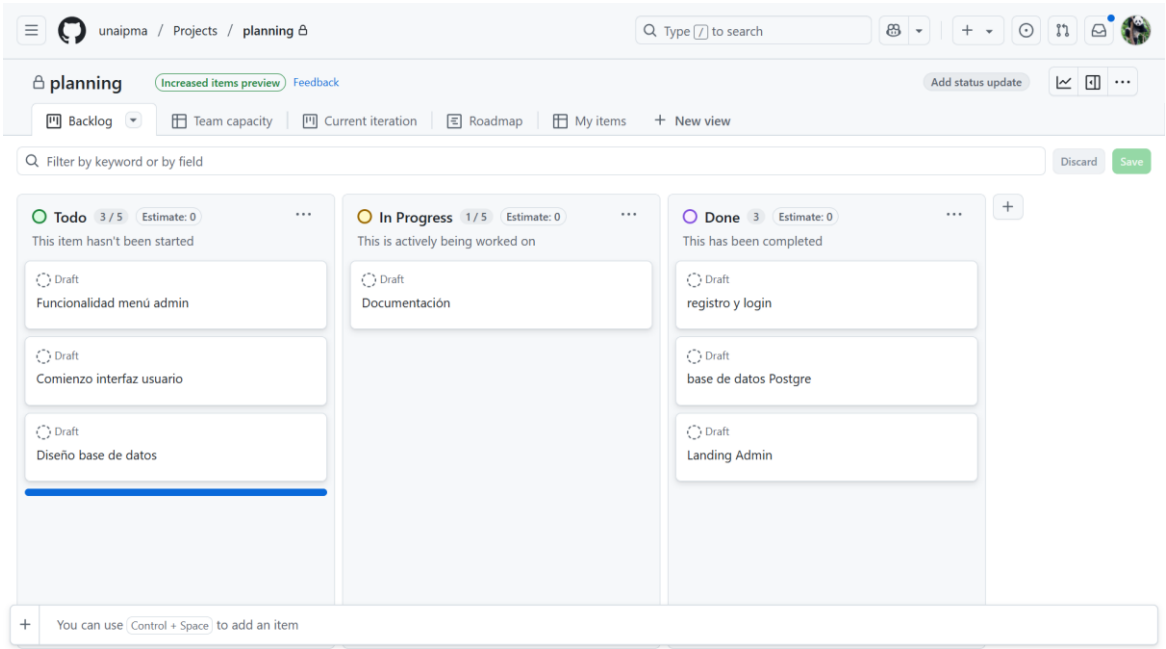

Figura 2 Tablero Kanban Sprint 2

3.5.3 Sprint 3

Fecha: 1 de abril al 22 de abril de 2025

Este es el aspecto de mi tablero Kanban tras mi segundo sprint.

Para el tercer sprint he propuesto acabar de implementar la facturación para poder guardar las ventas, acabar de hacer el diseño de la base de datos, el cual no fue posible finalizar en el sprint anterior por cambios constantes que he realizado en ambas bases de datos y revisar una guía sobre una API para intentar una futura implementación.

He conseguido finalizar La funcionalidad del menú de administrador y he avanzado bastante la interfaz de usuario (casi finalizada).

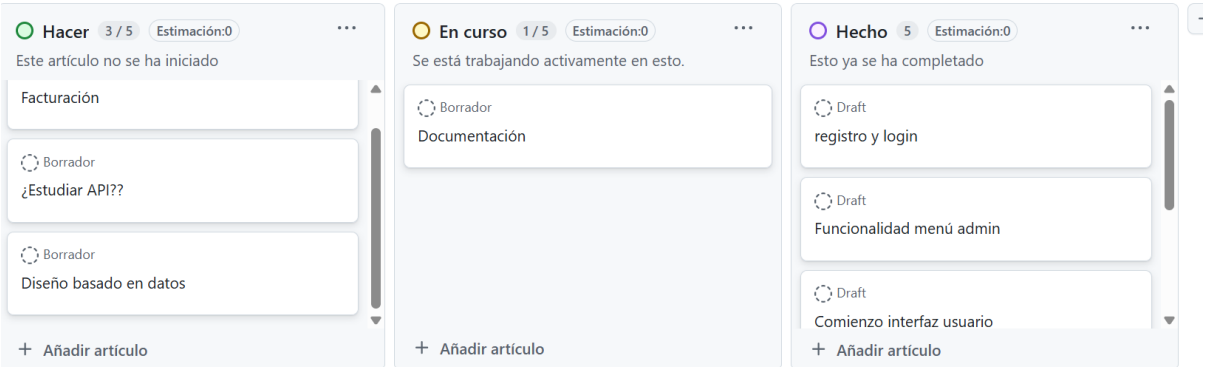

Figura 3 Tablero Kanban Sprint 3

3.5.4 Sprint 4

Fecha: 22 de abril al 6 de mayo de 2025

Para este sprint me he propuesto implementar filtros de búsqueda, implementar la base de datos sqlite como bd secundaria para poder usar la tpv sin necesidad de tener conexión y el diseño de las facturas.

Sigo realizando documentación del código.

He conseguido finalizar el diseño de la base de datos, todo el apartado de la funcionalidad de facturación y he descartado el uso de la API.

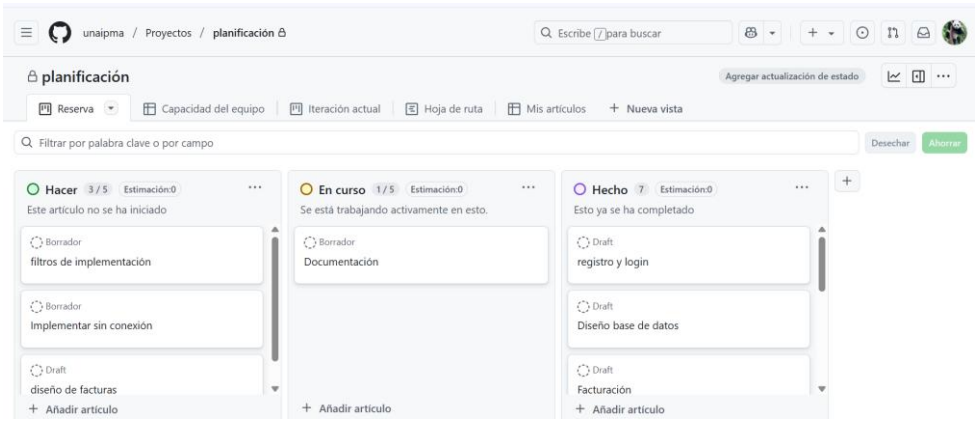

Figura 4 Tablero Kanban Sprint 4

3.5.5 Sprint 5

Fecha: 6 de mayo al 20 de mayo de 2025

Para este último sprint solo necesito acabar la memoria con sus correspondientes manuales.

He conseguido finalizar el diseño de las facturas, implementar el modo sin conexión, implementar los filtros de búsqueda y he documentado todo el código.

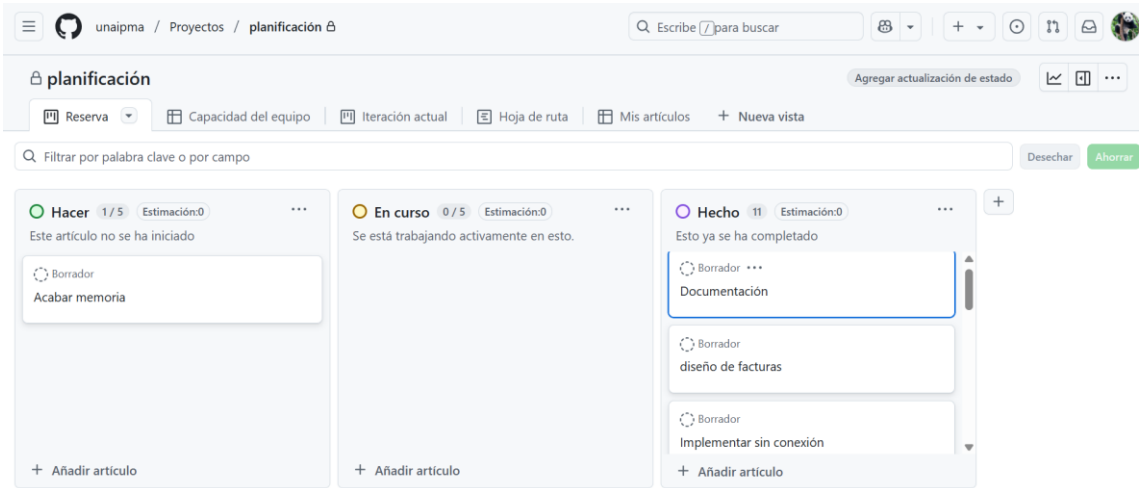

Figura 5 Tablero Kanban Sprint 5

4. Análisis

4.1 Tipos de usuarios

- **Trabajador**: este tipo de usuario podrá hacer el login sin contraseña solamente seleccionando la ubicación donde este trabaja y su usuario.

  Podrá registrar ventas, revisar productos y sus correspondientes precios, hacer cálculos…

- **Administrador**: este tipo de usuario podrá hacer login con su correo y su contraseña.

  Desde el menú este usuario puede gestionar los usuarios editando su información, eliminando el usuario o agregando uno nuevo. También podrá gestionar los productos, cambiar el precio, agregar nuevos productos… y obtener facturas.

4.2 Requisitos funcionales

Código	Nombre requisito
R1	**Gestión de usuarios**
R1.1	Creación de usuario R1.1.1 Usuario creado con éxito. R1.1.2 Error en la validación de datos de nuevo usuario. R1.1.3 Faltan datos requeridos por el Sistema. R1.1.4 Error de conexión con la BD.
R1.2	Edición de usuario R1.2.1 Usuario editado con éxito. R1.2.2 Error en la validación de datos de usuario. R1.2.3 Faltan datos requeridos por el Sistema. R1.2.4 Error de conexión con la BD.
R1.3	Eliminación de usuario R1.3.1 Usuario eliminado con éxito. R1.3.2 Fallo de conexión con la BD durante la eliminación.
R2	**Gestión de productos**
R2.1	Creación de producto R2.1.1 Producto creado con éxito. R2.1.2 Error en la validación de datos de nuevo producto. R2.1.3 Faltan datos requeridos por el Sistema. R2.1.4 Error de conexión con la BD.

R2.2	Edición de producto R2.2.1 Producto editado con éxito. R2.2.2 Error en la validación de datos del producto. R2.2.3 Faltan datos requeridos por el Sistema. R2.2.4 Error de conexión con la BD.
R2.3	Eliminación de producto R2.3.1 Producto eliminado con éxito. R2.3.2 Fallo de conexión con la BD durante la eliminación.
R3	**Gestión de ventas**
R3.1	Registrar venta R3.1.1 Venta registrada con éxito R3.1.2 Fallo de conexión con la BD durante el registro
R3.2	Eliminar venta R3.1.1 Venta eliminada con éxito R3.1.2 Fallo de conexión con la BD durante la eliminación
R4	**Gestión de facturas**
R4.1	Obtener factura R3.1.1 Factura obtenida con éxito R3.1.2 Fallo de conexión con la BD durante el registro
R4.2	Eliminar factura R3.1.1 Factura eliminada con éxito R3.1.2 Fallo de conexión con la BD durante la eliminación

4.3 Identificación de casos de uso

Actores:

- **Administrador:** Es el usuario encargado de gestionar todos los productos, usuarios y facturas.
- **Trabajador:** Es el usuario encargado de registrar las ventas que hace.

4.3.1 Gestión

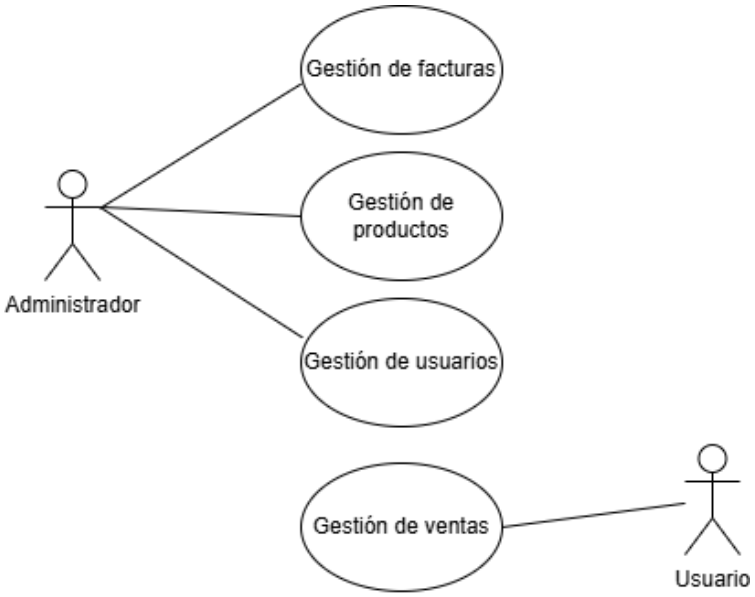

Figura 6 Personajes

5. Diseño de la base de datos

Para este proyecto he utilizado dos tipos de bases de datos, **PostgreSQL y SQLite**, donde se **sincronizan los datos** de la tabla SQLite con los de la base de datos PostgreSQL cada vez que se inicia sesión de manera online.

Para la base de datos PostgreSQL estoy utilizando el servicio de **Neon.tech** para alojar en este servicio la base de datos.

Los datos de la tabla facturación de PostgreSQL se crean cuando se cierra la caja en el programa a partir de los datos de SQLite de la tabla facturas

5.1 PostgreSQL:

En la base de datos PostgreSQL (nube) tenemos el apartado de facturación donde guardamos las facturas totales, la tabla logintrack con clave foránea en correo, donde se registra cuando accede cada administrador, la tabla productos donde están todos los productos y la tabla users donde se guardan todos los usuarios.

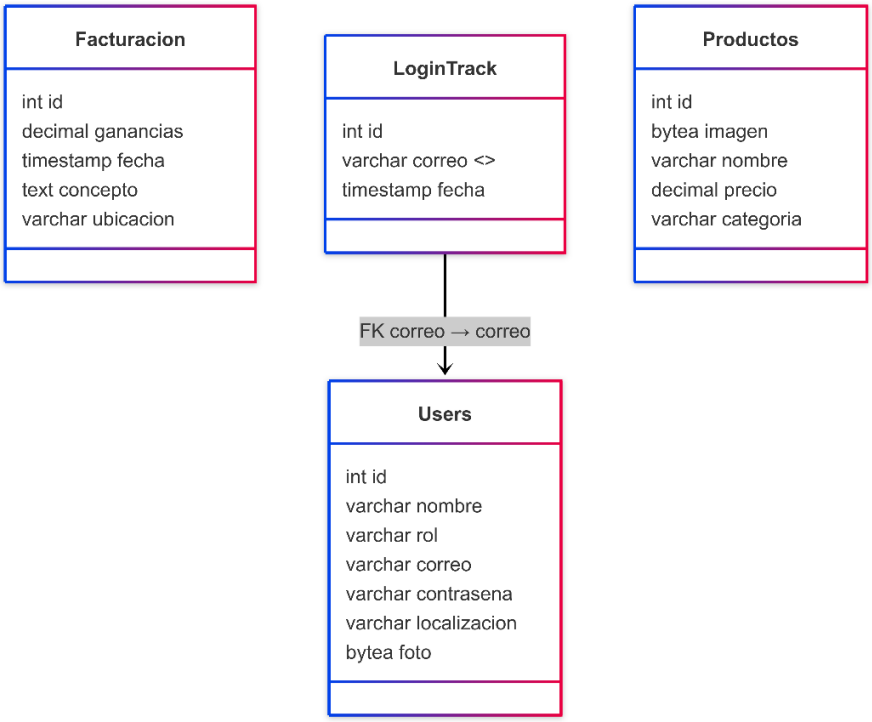

Figura 7 Base de datos PostgreSQL

5.2 SQLite:

La base de datos SQLite es una base de datos individual que estará en cada dispositivo y solo se sincronizaran de la nube los productos y los usuarios para en caso de no tener conexión, que se pueda seguir usando el programa. Esta base de datos cuenta la tabla ajustes, Facturaproductos , que sirve para identificar los productos que se han vendido en cada venta y tiene clave foránea con Facturas y productos. La tabla Facturas sirve para guardar las ventas realizadas y la tabla productos para guardar los productos. Los productos son los productos sincronizados desde la nube. La tabla users guarda los usuarios los cuales son obtenidos a través de la sincronización de la nube. La sincronización se realiza cada vez que se inicia sesión como usuario y hay conexión.

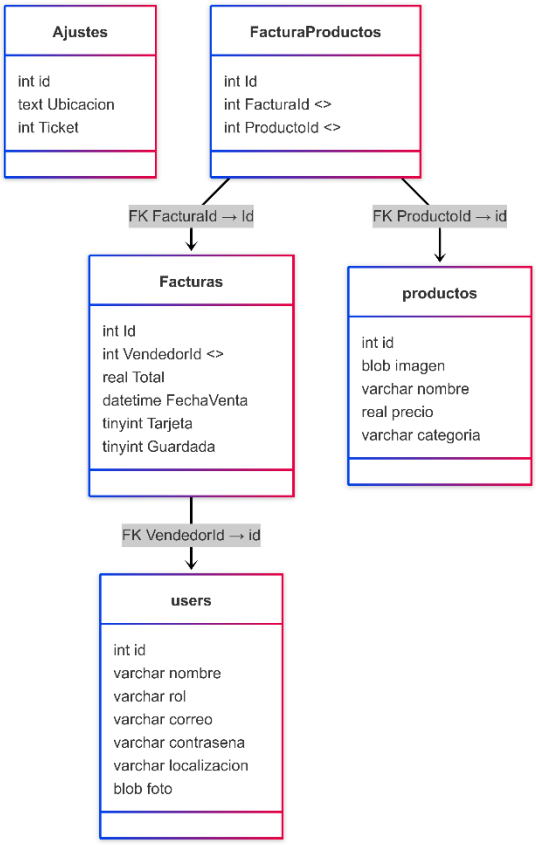

Figura 8 Base de datos SQLite

6. Implementación del sistema

La implementación del sistema TPV se ha desarrollado utilizando C# sobre el entorno .NET Framework, haciendo uso de Windows Forms (WinForms) para la interfaz gráfica. El diseño se ha basado en una arquitectura en tres capas que separa la presentación, la lógica del negocio y el acceso a datos, facilitando el mantenimiento y la escalabilidad del sistema.

6.1 Arquitectura del sistema

Capa de Presentación: compuesta por formularios WinForms que permiten la interacción del usuario con el sistema. Los menús y formularios están diseñados para ofrecer una navegación intuitiva por las funcionalidades principales: ventas, gestión de productos, usuarios y visualización de facturas.

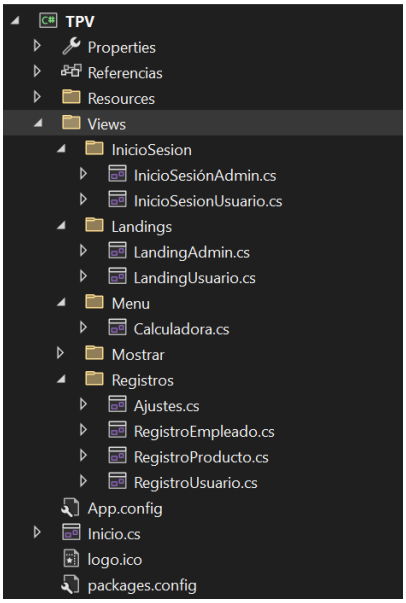

Figura 9 Capa de presentación

Capa de Negocio: contiene la lógica que regula el comportamiento de las operaciones. Aquí se validan las acciones del usuario, se gestionan reglas de negocio y se delegan las tareas a la capa de datos.

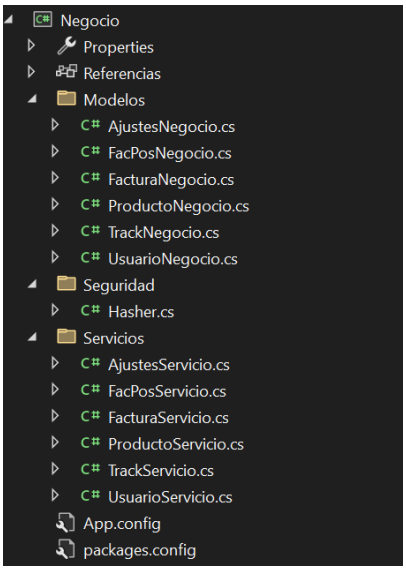

Figura 10 Capa de Negocio

Capa de Datos: implementa la comunicación con dos tipos de bases de datos y tiene los metodos que interactuan con la base de datos

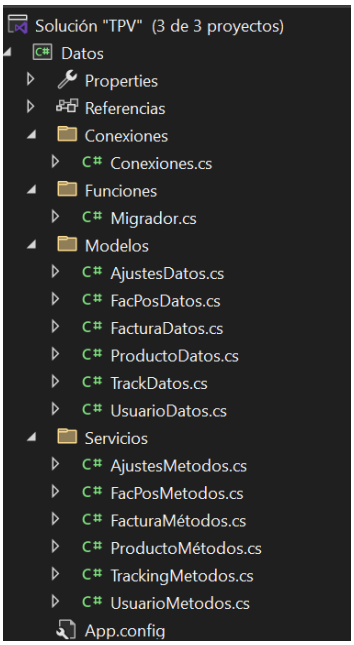

Figura 11 Capa de datos

```
private readonly string connectionString = "Host=ep-still-brook-a2m8tgov-pooler.eu-central-1.aws.neon.tech;" +
                                           "Username=tfg_owner;" +
                                           "Password=npg_ElXtVLGa6ZP1;" +
                                           "Database=tfg;" +
                                           "SslMode=Require;" +
                                           "Trust Server Certificate=true";


private readonly string RutaBD = Path.GetFullPath(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "TPV.db"));
```

Figura 12 Conexiones

```
public NpgsqlConnection ObtenerConexion()
{
    try
    {
        var conexion = new NpgsqlConnection(connectionString);
        conexion.Open();
        return conexion;
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

Figura 13 Conexión PostgreSQL

```
public SQLiteConnection ObtenerConexionSql()
{
    try
    {
        var conexion = new SQLiteConnection($"Data Source={RutaBD};Version=3;");
        conexion.Open();
        return conexion;
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

Figura 14 Conexión SQLite

Aquí muestro como realizo la conexión a ambas bases de datos. Hacer la conexión de esta manera nos permite poder cambiar de base de datos de una manera sencilla para en un futuro poder habilitar la TPV a varios clientes de una manera muy sencilla ya que solo habría que cambiar la cadena de conexión

Ejemplo de consulta con las dos bases de datos :

```
public List<ProductoDatos> ObtenerProductos()
{
    List<ProductoDatos> productos = new List<ProductoDatos>();

    try
    {
        // postgre
        using (var conexion = ConexionDB.ObtenerConexion())
        using (var cmd = new NpgsqlCommand("SELECT id, imagen, nombre, precio, categoria FROM productos", conexion))
        using (var reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                productos.Add(new ProductoDatos
                {
                    Id = reader.GetInt32(0),
                    Imagen = reader.IsDBNull(1) ? null : (byte[])reader["imagen"],
                    Nombre = reader.GetString(2),
                    Precio = reader.GetDecimal(3),
                    Categoria = reader.GetString(4)
                });
            }

            return productos;
        }
    }
    catch (Exception exPg)
    {
        try
        {
            // sqlite
            using (var conexion = ConexionDB.ObtenerConexionSql())
            using (var cmd = new SQLiteCommand("SELECT id, imagen, nombre, precio, categoria FROM productos", conexion))
            using (var reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    productos.Add(new ProductoDatos
                    {
                        Id = reader.GetInt32(0),
                        Imagen = reader.IsDBNull(1) ? null : (byte[])reader["imagen"],
                        Nombre = reader.GetString(2),
                        Precio = Convert.ToDecimal(reader.GetDouble(3)),
                        Categoria = reader.GetString(4)
                    });
                }

                return productos;
            }
        }
        catch (Exception exSql)
        {
            Console.WriteLine("❌ También falló SQLite: " + exSql.Message);
            return new List<ProductoDatos>();
        }
    }
}
```

SQLite: utilizada para almacenar datos de forma local en el equipo, permitiendo el funcionamiento del sistema incluso sin conexión a internet. Se usa principalmente para el almacenamiento de facturas, configuración local y datos temporales. Esta base de datos está situada en la capa de datos en un archivo .db. Al utilizar esta base de datos el proyecto debe localizarse en una carpeta diferente a la de archivos del programa o ejecutar el programa cómo administrador, ya que requiere permiso de escritura.

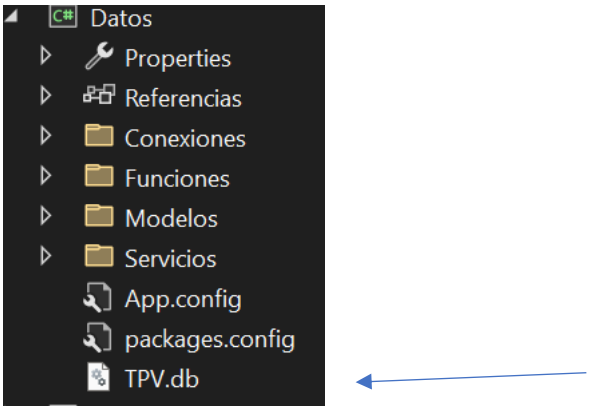

Figura 15 Ubicación base de datos SQLite

PostgreSQL (en la nube): sirve como repositorio remoto y centralizado para almacenar ciertos registros clave, como copias de seguridad, estadísticas diarias o consolidación de ventas de varios terminales. La conexión se realiza mediante una cadena de conexión segura gestionada a través de variables de entorno y almacenada fuera del en la capa de datos, lo que evita exponer credenciales sensibles, y las operaciones se ejecutan con sentencias parametrizadas para garantizar la seguridad de los datos. Se hace uso de librerías específicas para PostgreSQL en C#, como Npgsql, que permiten ejecutar consultas y transacciones utilizando sentencias parametrizadas, protegiendo el sistema frente a inyecciones SQL y otros ataques comunes.

Esta dualidad de almacenamiento permite combinar la velocidad y autonomía de SQLite en local con la seguridad y centralización de PostgreSQL en la nube, lo que dota al sistema de gran flexibilidad para distintos entornos de uso. Gracias al uso de una base de datos en la nube, todos los negocios tienen los mismos precios si es necesario, la misma información que los otros negocios y además permite hacer cambios desde la oficina/casa sin tener que ir al negocio a hacer el cambio de los precios, ver las facturas…

6.2 Gestión de productos y usuarios

El sistema permite gestionar productos y usuarios a través de formularios dedicados. Las operaciones **CRUD** (crear, leer, actualizar y eliminar) permiten modificar, crear, leer información y eliminar usuarios y productos de la base de datos PostgreSQL. Estos datos son sincronizados con la base de datos local SQLite cada vez que se inicia sesión como usuario y hay conexión. Esto permite trabajar con los datos más actualizados en caso de que haya conexión o con los datos mas recientes en caso de que no haya conexión a Internet.

6.3 Facturación y generación de tickets

Durante una venta, el sistema permite seleccionar productos y generar una factura que asocia al vendedor, la fecha y el importe total. Estas facturas se almacenan localmente en SQLite para acceso rápido y posibilidad de **funcionamiento offline**. Además, se incluye una opción para generar un ticket o factura en PDF, facilitando su impresión o envío digital. Estas facturas son subidas a la nube me forma conjunta cada vez que se realiza un cierre de caja.

6.4 Registro de estadísticas y cierre diario

Al final de cada jornada, se ejecuta un proceso de cierre diario, que calcula las ganancias y registra un resumen en una tabla de estadísticas. Esta información se envía a la base de datos PostgreSQL, asegurando su persistencia y disponibilidad remota para futuros análisis o auditorías.

6.5 Seguridad y autenticación

El sistema incorpora un mecanismo de autenticación de usuarios, con credenciales almacenadas de forma segura mediante **hashing** de contraseñas. Además, las operaciones sensibles están restringidas por roles, permitiendo que solo usuarios administradores puedan gestionar usuarios o acceder a ciertos informes. Para iniciar sesión como usuario no hace falta contraseña, basta con poner la ubicación en la que estás trabajando y seleccionar tu usuario lo que permite que cuando haga falta incorporar nuevos empleados, el administrador lo podrá crear previamente desde su oficina y los empleados no tendrán que realizar ninguna acción extra.

6.6 Despliegue y portabilidad

Gracias a su **arquitectura híbrida**, el sistema puede desplegarse fácilmente en cualquier equipo Windows con .NET Framework. La base de datos local en SQLite se almacena como un único archivo .db, y la sincronización con PostgreSQL se realiza automáticamente al detectar conexión. Esto permite combinar portabilidad, autonomía y conectividad, adaptándose a diversos escenarios de uso en un entorno comercial real.

6.7 Extensiones

He utilizado itext7, una herramienta para crear pdf de manera más sencilla y ordenada, BouncyCastle que proporciona el soporte criptográfico requerido por iText y las extensiones de PostgreSQL y SQLite para poder conectar la base de datos.

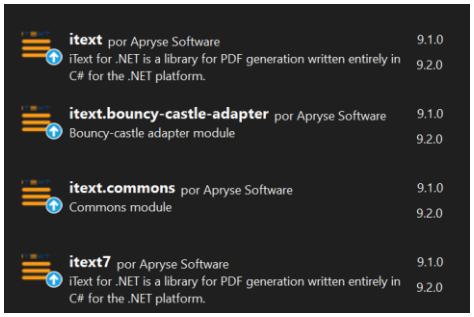

Figura 16 Extension itext desde administrador de paquetes Nuget

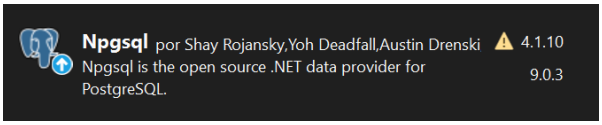

Figura 17 Extensión para la conexión PostgreSQL

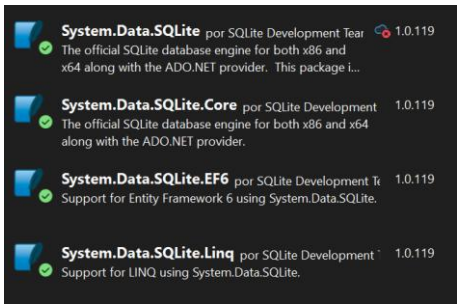

Figura 18 Extensión para la conexión SQLite

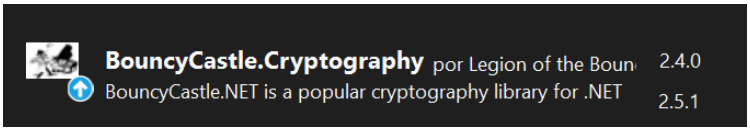

Figura 19 Extensión BouncyCastle

También he hecho uso de la aplicación advanced installer para generar el msi y el .exe de una manera intuitiva y sencilla

Figura 20 Logo Advanced Installer

6.8 Rendimiento

Esta aplicación al usar C# con .NET es una aplicación muy eficiente:

Figura 21 Uso de recursos

no consume apenas CPU y solo consume 71,4 MB de RAM

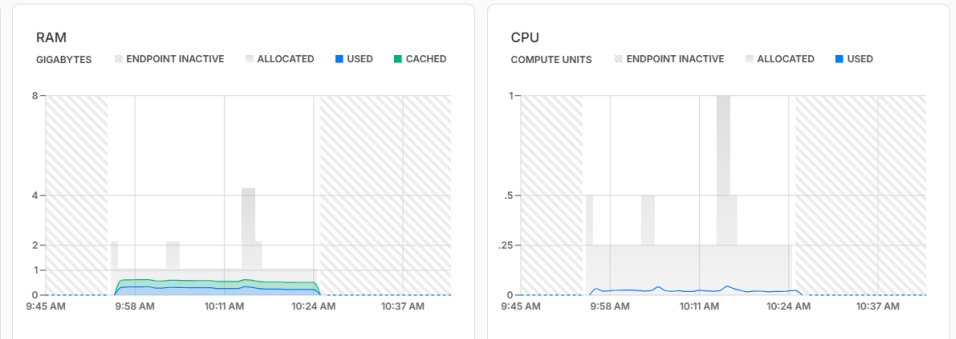

Figura 22 Rendimiento Base de datos

Y este es el rendimiento de la base de datos de la nube (PostgreSQL) haciéndole peticiones de inicio de sesión, obtener datos y subir facturas.

6.9 Problemas encontrados

Al imprimir copia de la venta y más si todavía si está activada la opción de imprimir siempre la copia, se quedaba bloqueada la tpv y no se podía utilizar hasta que se imprimiera el ticket por lo que he implementado el uso de tasks, lo que hace que se asigne esa tarea a un hilo que ya ha sido creado.

```
1 referencia
private async void btnImprimirCopia_Click(object sender, EventArgs e)
{
    await Task.Run(() => imprimir());
}
```

Figura 23 Uso de Tasks

He decidido utilizar **tasks** en vez de hilos porque es mucho mas eficiente ya que el uso de hilos conlleva crear un nuevo hilo asignarle esa tarea y después destruirlo, aquí sin embargo solo asigno la tarea a un hilo ya creado.

También he utilizado tasks en el método en el que cargo facturas desde local ya que hay que realizar muchas acciones y así funciona de manera más eficiente ya sin las tareas le cuesta el doble. Gracias a esta tarea mientras cargo facturas también cargo los productos de dicha factura a la vez.

7. Pruebas de integración

En esta sección se desarrollarán pruebas manuales, conforme la funcionalidad del sistema se vaya desarrollando. Son muy importantes dentro del desarrollo de la aplicación, ya que la superación de dichas pruebas, dará paso al desarrollo del resto de la aplicación posteriormente y de esta manera comprobar, en cada momento, si el resto del desarrollo sigue cumpliendo con los requisitos descritos inicialmente.

A continuación, se proponen los casos de prueba que se deberán desarrollar a medida que se va dotando de funcionalidad al sistema.

7.1 Gestión de usuarios

Registrar Usuario

Caso de Prueba: CP1.1.1	
Entrada	**Resultado esperado**
Registrar un usuario no existente en la aplicación	El sistema posee un usuario más
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP1.1.2	
Entrada	**Resultado esperado**
Registrar un usuario ya existente en la aplicación	El sistema no posee un usuario más y se muestra un mensaje indicando que el usuario ya existe en la aplicación
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP1.1.3	
Entrada	**Resultado esperado**
Registrar un usuario con datos incorrectos	El sistema no posee un usuario más y se muestra un mensaje indicando que se han introducido datos incorrectos
	Resultado obtenido
	Prueba correcta

Editar Usuario

Caso de Prueba: CP1.2.1	
Entrada	**Resultado esperado**
Editar un usuario de la aplicación	Los datos del usuario son modificados de acuerdo a los nuevos datos.
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP1.2.2	
Entrada	**Resultado esperado**
Editar un usuario de la aplicación con datos no válidos	No se actualizan los datos del usuario y se muestra un mensaje de error indicando que se introdujeron datos que no superaron la validación.
	Resultado obtenido
	Prueba correcta

Eliminar Usuario

Caso de Prueba: CP1.3.1	
Entrada	**Resultado esperado**
Eliminar un usuario de la aplicación	El sistema elimina el usuario.
	Resultado obtenido
	Prueba correcta

7.2 Gestión de productos

Registrar producto

Caso de Prueba: CP2.1.1	
Entrada	**Resultado esperado**
Registrar un producto	El sistema posee un producto más
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP2.1.2	
Entrada	**Resultado esperado**
Registrar un producto con datos incorrectos	El sistema no posee un producto más y se muestra un mensaje indicando que se han introducido datos incorrectos
	Resultado obtenido
	Prueba correcta

Editar producto

Caso de Prueba: CP2.2.1	
Entrada	**Resultado esperado**
Editar un producto de la aplicación	Los datos del producto son modificados de acuerdo a los nuevos datos.
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP2.2.2	
Entrada	**Resultado esperado**
Editar un producto de la aplicación con datos no válidos	No se actualizan los datos del producto y se muestra un mensaje de error indicando que se introdujeron datos que no superaron la validación.
	Resultado obtenido
	Prueba correcta

Eliminar producto

Caso de Prueba: CP2.3.1	
Entrada	**Resultado esperado**
Eliminar un producto de la aplicación	El sistema elimina el producto.
	Resultado obtenido
	Prueba correcta

7.3 Gestión de ventas

Registrar venta

Caso de Prueba: CP3.1.1	
Entrada	**Resultado esperado**
Registrar una venta con los datos correctos	El sistema registra la venta.
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP3.1.2	
Entrada	**Resultado esperado**
Registrar una venta sin seleccionar ningún usuario	El sistema avisa que hace falta seleccionar un usuario
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP3.1.3	
Entrada	**Resultado esperado**
Registrar una venta sin seleccionar ningún producto	El sistema avisa que no hay ningún producto seleccionado
	Resultado obtenido
	Prueba correcta

Eliminar venta

Caso de Prueba: CP3.2.1	
Entrada	**Resultado esperado**
Eliminar una venta	El sistema elimina la venta
	Resultado obtenido
	Prueba correcta

Eliminar producto de la venta

Caso de Prueba: CP3.2.1	
Entrada	**Resultado esperado**
Eliminar un producto de la venta habiendo productos añadidos	El sistema elimina el producto de la venta
	Resultado obtenido
	Prueba correcta

Caso de Prueba: CP3.2.1	
Entrada	**Resultado esperado**
Eliminar un producto de la venta sin haber productos	El sistema avisa de que no hay productos que eliminar
	Resultado obtenido
	Prueba correcta

7.4 Facturación

Eliminar factura

Caso de Prueba: CP4.1.1	
Entrada	**Resultado esperado**
Eliminar una factura del sistema	El sistema elimina la factura
	Resultado obtenido
	Prueba correcta

Ver factura

Caso de Prueba: CP4.2.1	
Entrada	**Resultado esperado**
Ver la factura seleccionada/ facturas	El sistema descarga un pdf con el contenido de la factura
	Resultado obtenido
	Prueba correcta

8. Manuales

He realizado manuales de usuario, para que los empleados sepan todas las funciones de la TPV, de administrador, para que el administrador sepa usar la TPV sin problema y el manual de instalación para la primera vez que se use el programa.

9. Conclusión y ampliaciones

A lo largo del desarrollo de este proyecto, he conseguido implementar una **TPV eficiente** orientada a cubrir las necesidades reales de facturación de pequeños negocios de hostelería. El sistema desarrollado destaca por su **sencillez** de uso, su **arquitectura modular** en tres capas, su capacidad de funcionar tanto **online** como **offline** y su integración con **dos motores de bases de datos** (PostgreSQL y SQLite). Estas características no solo mejoran la experiencia del usuario, sino que también garantizan la disponibilidad y seguridad de los datos.

El estar familiarizado con tecnologías como C#, WinForms y .NET Framework, me ha permitido un buen desarrollo y con buenos resultados en cuanto a rendimiento y mantenimiento. La inclusión de funcionalidades como la facturación con generación de tickets PDF, la gestión de productos y usuarios, el registro estadístico diario y la sincronización entre bases de datos son logros que reflejan tanto la planificación como la ejecución efectiva del proyecto. Durante el proceso también han surgido problemas como los bloqueos en la interfaz provocados por la impresión de tickets y la tardanza de la carga de las ventas, que fueron solventados eficientemente gracias al uso de tareas asincrónicas (Task). Estas soluciones mejoran considerablemente la usabilidad del sistema y permiten una experiencia fluida.

- Posibles ampliaciones

Aunque el sistema cumple satisfactoriamente con los objetivos iniciales, se identifican varias posibles ampliaciones para futuras versiones:

- Interfaz web o móvil que complemente la versión de escritorio, facilitando el acceso remoto a estadísticas o gestión desde cualquier lugar.
- Sincronización automática en tiempo real entre SQLite y PostgreSQL sin necesidad de reiniciar sesión.
- Módulo de inventario para registrar stock de productos y alertas cuando los niveles estén bajos.
- Nuevo apartado para gestionar mesas del local.

Con estas posibles mejoras, el sistema TPV podría convertirse en una solución aún más completa y profesional, adaptándose a más contextos y necesidades dentro del sector hostelero.

10. Bibliografía

Diez, A. (2012). Proyecto Fin de Master Universidad de Oviedo [Software]. En *DESARROLLO DEL PORTAL MULTICANAL «ORGTRANSPARENTE 2.0» MEDIANTE LIFERAY"*.

https://docs.google.com/document/d/1AsVrGjBMTQG2DANexW9UloeX_o-fuJx3/edit

Freepik | Diseña mejor y más rápido. (s. f.). Freepik. https://www.freepik.es/

GitHub · Build and ship software on a single, collaborative platform. (2025). GitHub.

https://github.com/

https://github.com/unaipma/tfg.git

Mermaid | Diagramming and charting tool. (s. f.). https://mermaid.js.org/

Scribbr. (2025, 3 febrero). *Formato APA con el Generador APA de Scribbr*.

https://www.scribbr.es/citar/generador/apa

Draw.io. (s. f.). https://www.drawio.com/

Advanced Installer. (s. f.). *Free Windows Installer - MSI Installer Tool - Advanced Installer*.

https://www.advancedinstaller.com/

SQLite Home page. (s. f.). https://sqlite.org/