



Universidad Politécnica Internacional
Escuela de Ingeniería Informática
Técnicas de programación

Proyecto#2 Gestor de Gastos

Integrantes:

Yanicsa Galilea Casco

Melissa Fallas Jiménez

Profesor: Luis Felipe Mora Umaña

Centro Universitario Heredia

23 de agosto del 2025

Índice

Introducción.....	3
Decisiones de diseño	4
Estructura General del Proyecto	4
Controlador	5
Gestor de datos.....	6
Modelo	7
Vista	9
Desarrollo	10
Ejecución y Uso del Sistema	11
1. Inicio del Programa.....	11
2. Funcionalidades	11
Testing	17
Organización del Proyecto.....	18
Tecnologías Utilizadas	19
Análisis de Resultados.....	20
Aprendizajes	21
Conclusiones.....	22
Cosas por mejorar o que no se hicieron.....	23

Introducción

El presente documento corresponde a la segunda iteración del proyecto Gestor de Gatos, una aplicación orientada a la gestión de gastos compartidos entre grupos de usuarios. En esta versión se incorporan mejoras significativas respecto a la primera entrega, aplicando principios de programación orientada a objetos, arquitectura MVC, y lineamientos de Clean Code, con el propósito de obtener un sistema más robusto, escalable y fácil de mantener.

Este trabajo no solo busca cumplir con los requerimientos funcionales planteados, sino también fortalecer el proceso de desarrollo mediante el uso de buenas prácticas, herramientas modernas y metodologías que garanticen calidad en el producto final. De esta manera, se fomenta una visión integral que combina la parte técnica con la organización y planificación del proyecto, utilizando Trello como herramienta de seguimiento y Git como sistema de control de versiones.

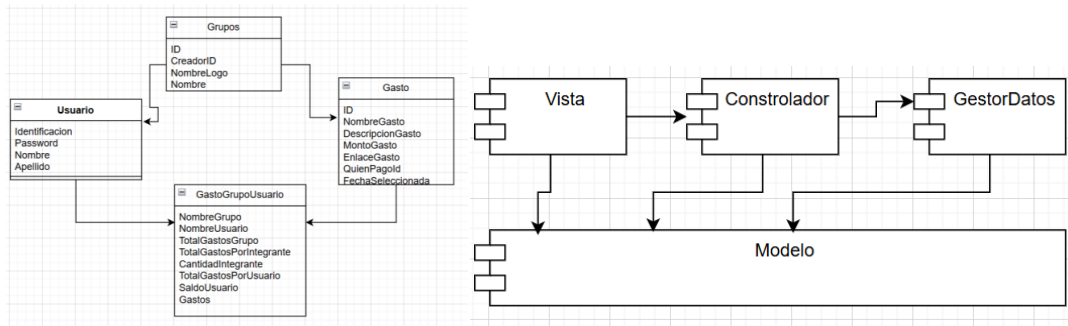
La documentación aquí presentada ofrece una descripción de las decisiones de diseño adoptadas, el proceso de desarrollo, la manera en que el sistema es utilizado y ejecutado, así como los aprendizajes y conclusiones obtenidas durante su construcción.

Decisiones de diseño

Estructura General del Proyecto

A continuación, se presenta el diagrama diseñado en Draw.io, el cual permite ilustrar de forma gráfica la estructura del proyecto y mostrar cómo se organizan los principales componentes que conforman el sistema.

Cabe destacar que, las decisiones de diseño son las mismas utilizadas en el Proyecto 1, ya que este Proyecto 2 es la continuidad del proyecto 1, constando la presente versión 2 con mejoras, pero en el área de desarrollo, no interfirió en las decisiones de diseño.



La estructura y el código del proyecto se divide en cuatro partes, una que es el controlador, otro gestor de datos, modelo y la parte de la vista.

Se utilizó el patrón MVC para una mejor organización y se incorpora un Gestor Datos para cumplir con el principio SOLID de responsabilidad única, se separa el manejo de datos del controlador vista y modelo para que pueda gestionar todas las operaciones de escritura y lectura sin modificar los comportamientos del sistema en caso se desee cambiar el origen de los datos.

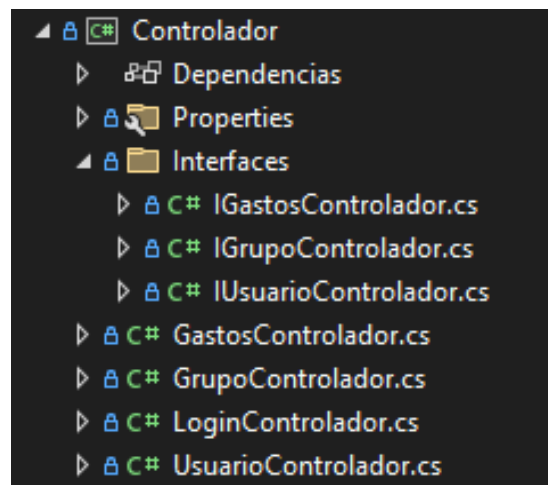
Controlador

Este es fundamental en la arquitectura del sistema ya que, se encarga de la lógica del proyecto y es el responsable de manejar la interacción entre la vista, el modelo y en este caso también interactúa con el gestor datos, este controlador trabaja con interfaces para que no fueran consumidos directamente por las vistas mediante clases concretas, sino a través de interfaces bien definidas.

Se dividió en cuatro controladores:

1. Una parte donde se va a gestionar todo lo que es la información de los gastos.
2. Otra parte donde se gestiona toda la información de los grupos.
3. Otra parte que es específicamente para la validación del login donde se van a manejar el usuario y la contraseña.
4. La otra es un controlador de usuarios donde se va a manejar la información de los usuarios para buscarlos, obtenerlos, crearlos.

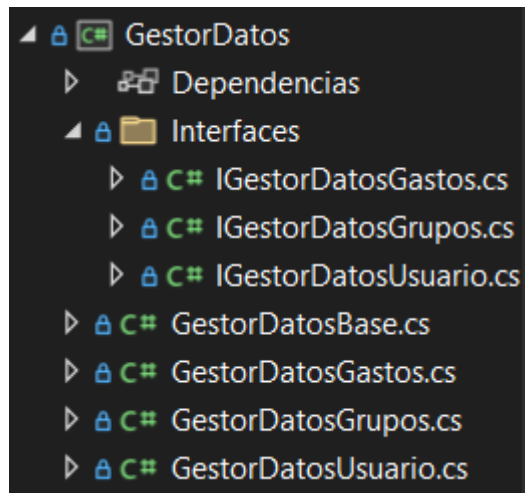
Y se crearon 3 interfaces que van a ser representadas para la parte del usuario, grupo y gasto



Gestor de datos

Este fue diseñado con el objetivo de encapsular todas las operaciones relacionadas con la persistencia y recuperación de datos, siguiendo el principio de separación de responsabilidades. Esta capa actúa como intermediaria entre los controladores del sistema y las fuentes de datos, que en esta iteración inicial son archivos JSON

Su principal responsabilidad es proporcionar métodos y funciones reutilizables, eficientes y desacoplados que permite cargar, guardar y actualizar la información necesaria para el funcionamiento del sistema según sea el modelo. En lugar de permitir que los controladores accedan directamente a los archivos del sistema, se introdujo una arquitectura basada en tres interfaces, IGestorDatosUsuario, IGestorDatosGrupo e IGestorDatosGasto.

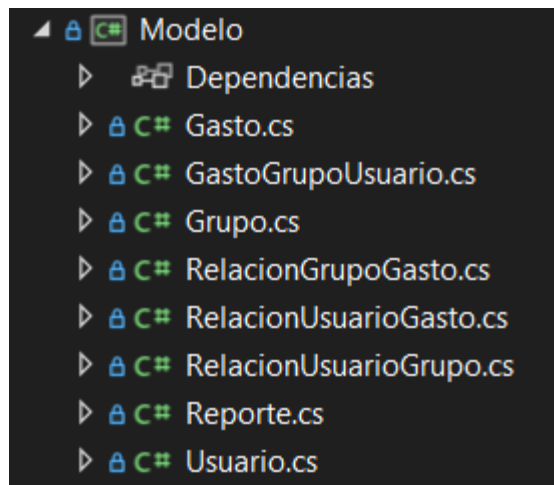


En la parte de gestor de datos también se utilizaron cuatro gestores de datos:

1. Uno general que se encarga de leer el json y escribir en el json.
2. El del gestor de gastos se va a encargar de obtener información de los gastos, cargarlos y guardarlos.
3. La parte del gestor de datos grupos, este se encarga de cargar grupos, crearlos para guardarlos en los json.
4. Gestor de datos usuario es el encargado de crear los usuarios y guardarlos en los jsons.

Modelo

Este se encarga de las entidades como usuario, grupo y gasto, etc. Cada clase del modelo se diseñó utilizando principios claves de POO aplicando conceptos como herencia, Por ejemplo, la clase Gasto contiene atributos como Nombre, Descripción, Monto, Fecha. Estos métodos son internos al modelo, lo que asegura que la lógica esté centralizada en el lugar correcto, sin depender de la vista o del controlador para validaciones básicas.



Contiene:

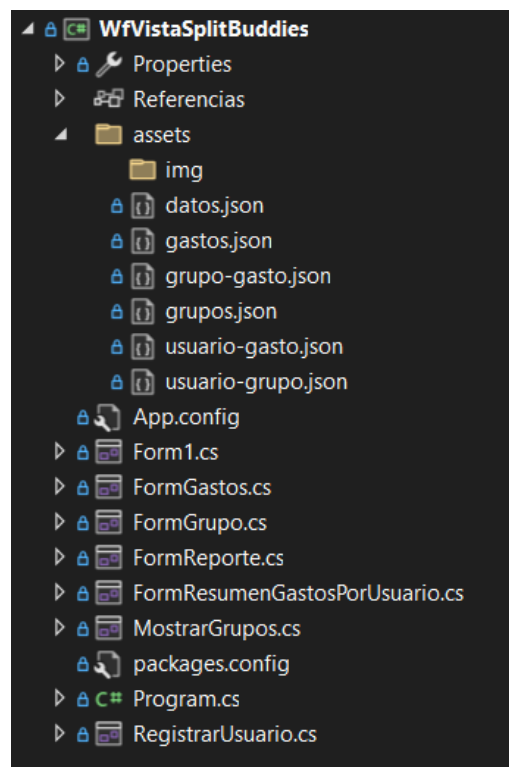
- Una clase para gastos
- Una clase para la relación de gasto grupo usuario. La relación de grupo usuario es para relacionar un grupo tiene ciertos usuarios, y el usuario tiene ciertos grupos. Esta tiene el nombre del Grupo, nombre del usuario, total gastos grupos, total gastos por integrante, cantidad de integrantes, total gastos por usuario, y el saldo usuario.
- Una clase de grupo, que va a representar los grupos creados.
- La relación grupo gasto, que es para relacionar todos los gastos que tiene un grupo.
- La relación usuario gasto es para relacionar todos los gastos que tiene un usuario.

- Y relación usuario grupo es para relacionar todos los grupos en los cual puede estar un usuario.
- La clase reporte es la que va a manejar los gastos totales que ha tenido el usuario, qué ha pagado y la deuda, que es lo que debe, así como, la disponibilidad de lo de lo que tiene a favor y en contra.
- La clase usuario, que contiene toda la información personal del usuario, que también es la que se usa para crear el usuario y validar el login.

Vista

Es la responsable de toda la interacción directa con el usuario. En esta primera iteración del sistema, se ha implementado utilizando el framework Windows Forms. Cada formulario FormUsuarios, FormGrupos, FormGastos, FormEstadisticas, representa una unidad visual independiente, diseñada para mostrar datos provenientes del modelo y recoger entradas del usuario que luego son delegadas al controlador correspondiente. Esta capa nunca contiene lógica de negocio ni lógica de persistencia, respetando estrictamente el principio de separación de responsabilidades.

Un aspecto relevante del diseño de las vistas es que cada formulario interactúa exclusivamente con interfaces de los controladores como IGastosControlador, IGrupoControlador, en lugar de depender de instancias concretas. También, se agrega una carpeta assets con los archivos json para facilitar el acceso a los datos desde los formularios de Windows Forms durante la ejecución del sistema sin necesidad de configurar rutas absolutas o entornos de despliegue más complejos; y una carpeta de imágenes donde están todos los logos de los grupos



Desarrollo

Gestor de Gastos es un sistema que aborda el problema del control de gastos compartidos entre múltiples usuarios y se desarrolló bajo el enfoque de la programación orientada a objetos, implementando principios de herencia, polimorfismo y encapsulamiento como base del diseño. Además, se aplicaron buenas prácticas de desarrollo tales como los principios SOLID, Clean Code, y la arquitectura MVC, lo que contribuye a la reutilización de código, mantenibilidad, escalabilidad y robustez del sistema.

En esta segunda iteración, se mantuvo la misma estructura y funcionalidad que en la primera. Sin embargo, se realizaron mejoras a nivel visual, optimizando la presentación de la interfaz para una experiencia más agradable. Además, se incorporaron pruebas Testing para la capa del controlador, garantizando una mayor robustez y fiabilidad del sistema. En cuanto a patrones de diseño, se implementa el patrón Singleton, asegurando un control centralizado en componentes clave del sistema.

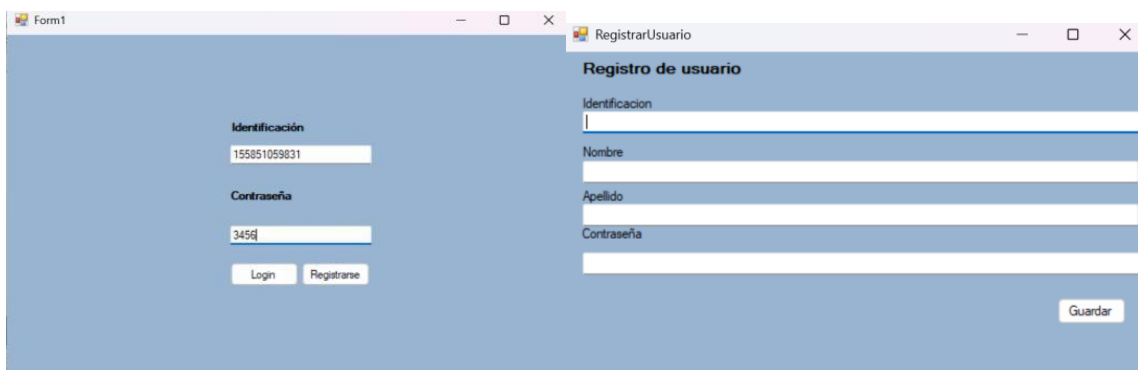
A continuación, se detalla el proceso de ejecución del sistema y las interacciones fundamentales que ofrece, junto con las decisiones de diseño que respaldan su implementación:

Ejecución y Uso del Sistema

Primeramente, el sistema no requiere de instalación. Para ejecutarlo basta con compilar el proyecto en Visual Studio y ejecutar el formulario principal.

1. Inicio del Programa

Al iniciar el programa aparece una pantalla de gestión de usuarios el cual permite registrarse con id, nombre, apellido y contraseña o iniciar sesión con el Id y contraseña el cual contiene validaciones de error.



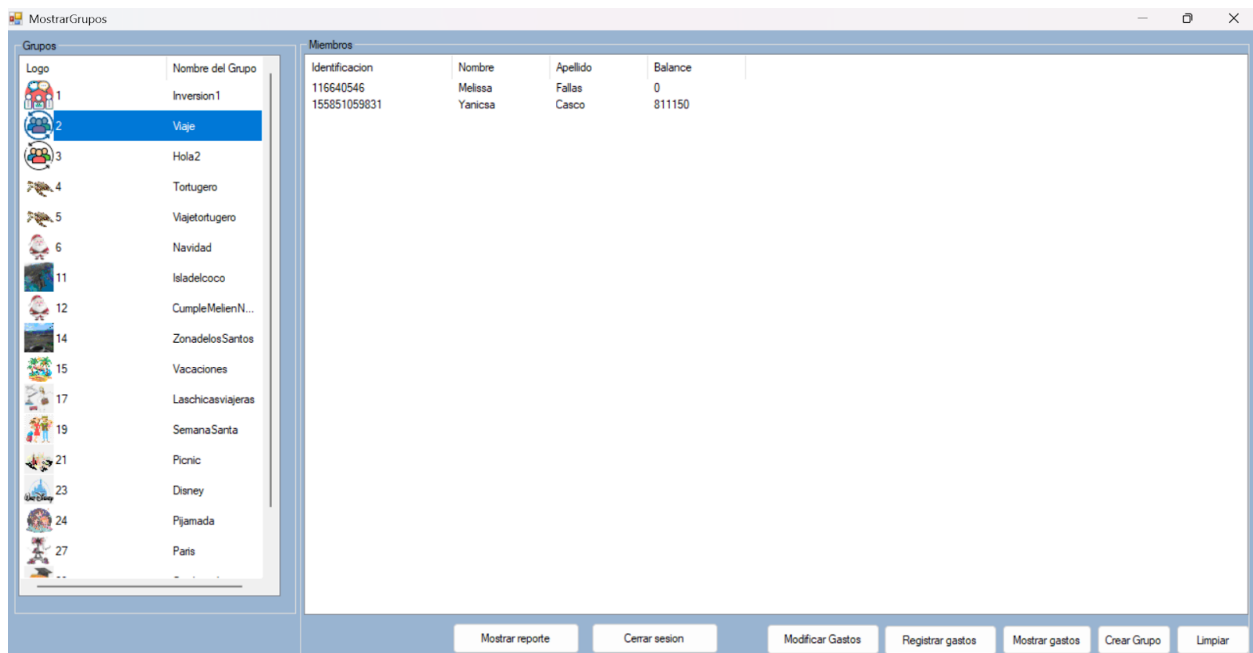
El sistema carga automáticamente archivos JSON con datos simulados: 50 usuarios, 30 grupos, y más de 100 gastos, distribuidos en fechas diversas. Estos archivos se encuentran dentro de la carpeta Assets de la Vista para facilitar su localización

2. Funcionalidades

Una vez se de en registrar o login y esté dentro del programa se mostrará la primera pantalla donde se muestra los grupos y sus miembros, los usuarios pueden visualizar todos los grupos registrados, cada uno representado con su imagen y nombre.

Al seleccionar un grupo se muestran los miembros del grupo con su información y balance. Esta pantalla incluye botones a otras pantallas que serían mostrar reporte, Registrar gastos, mostrar gastos, crear grupo y un botón de limpiar que este lo que hace es que se quita el grupo seleccionado y se deja de mostrar la información de los miembros en pantalla.

En esta nueva iteración se agregaron los botones de cerrar sesión y modificar gastos.



Cuando se selecciona el botón de mostrar reporte se despliega la segunda pantalla con el reporte individual del usuario con el que accedió al sistema. Esta sección permite filtrar y visualizar datos financieros en función de un período determinado, seleccionado mediante los campos de fecha. El botón Generar activa el análisis de datos para el rango seleccionado, mientras que Limpiar permite reiniciar la información.

Usuario: Yanicsa Casco

Desde: 24 julio 2025 Hasta: 30 julio 2025

Mes: julio 2025

Año: 2025


Monto pagado en el periodo: 511150

Monto adeudado en el periodo: 200000

Monto disponible en el periodo: 311150

Ahora volviendo a la pantalla de Mostrar Grupos, al seleccionar el siguiente botón que sería Registrar gastos se despliega la tercera pantalla la cual contiene un formulario con campos para ingresar el nombre del gasto, su descripción, un enlace relacionado, el monto involucrado, así como el responsable del pago, los integrantes participantes, la

fecha en que se registró y un botón de guardar y otro de cancelar. La inclusión de enlaces y participantes ayuda a mejorar la trazabilidad y claridad de cada gasto, evitando cualquier confusión dentro del grupo.



Formulario de gastos

Nombre

Descripción

Enlace

Monto

Quién lo pagó

Integrantes
☐ Melissa
☐ Yanicsa
☐ yuy

Fecha

Al seleccionar un grupo y presionar el botón Modificar gastos, el sistema despliega una pantalla dedicada a la edición de gastos. Esta pantalla presenta un formulario similar al utilizado para agregar gastos. A un lado se incluye una sección donde se listan los nombres de los gastos asociados al grupo seleccionado. El usuario puede seleccionar un gasto específico mediante su identificador y, al pulsar el botón Cargar gasto, el sistema completa automáticamente el formulario con toda la información existente de dicho gasto (como descripción, monto, fecha y categoría).

Una vez cargados los datos, el usuario puede modificar cualquier campo necesario directamente en el formulario. Posteriormente, al presionar el botón Guardar, el sistema actualiza el gasto aplicando las modificaciones realizadas.

También se incorpora la funcionalidad para eliminar un gasto específico. Para ello, el usuario selecciona el Nombre del gasto correspondiente y, a través de la opción Eliminar, el sistema procede a removerlo.

Formulario de gastos

Nombre

Nombre del Gasto

Descripción

Enlace

Monto

Quién lo pagó

Integrantes

Fecha

Cargar gasto Eliminar

Agregar

Actualizar Cancelar

De la misma forma, volviendo a la pantalla principal se selecciona el botón de Mostrar Gastos y se mostrará una pantalla el cual permite seleccionar el usuario y el grupo y un botón de mostrar el balance que mostrará una tabla con información de los gastos del usuario dentro de ese grupo. Debajo de esta tabla, se encuentran datos como el total de gastos del grupo, total por integrante, tu saldo personal, la cantidad de miembros del grupo y los gastos que tú has pagado, si sale en verde está positivo es decir pagaste lo que debías, si sale rojo es negativo, es decir, que debes dinero.

Gastos usuario
Yanicsa

Grupo
Yanicsa Viaje

Mostrar balance

Lista de gastos

NombreGasto	DescripcionGasto	EnlaceGasto	MontoGasto	QuienPagoid	FechaSele
Viaticos	viaticos de negocio	google.com/viaje	11150	155851059831	28/7/2025
Compre acciones	se compran acci...	bola.com	500000	155851059831	28/7/2025
Viaje a Punta Cana	Viaje a Punta Ca...	https://www.goo...	500000	155851059831	18/3/2025
...

Gastos totales del grupo	Gastos totales por integrante	Saldo del usuario
811150	405575	3854575
Cantidad de integrantes	Gastos pagados del usuario	
2	4260150	

Por último, si se selecciona el botón de crear grupos desde la pantalla principal se desplegará una pantalla con un formulario con espacios para ingresar datos como nombre del grupo que se desea crear, seleccionar una imagen representativa y elegir a los miembros que se desea integrar. Los botones de Crear y Cancelar permiten confirmar o descartar la creación del grupo



Formulario para crear un grupo

Nombre del grupo

Seleccione la imagen del grupo



Seleccionar

Seleccionar los integrantes

☐ Melissa
☐ yuy
☐ 989
☐ Lucia

Testing

En esta segunda iteración se implementó el Testing para la capa del controlador con el objetivo de garantizar la correcta funcionalidad del sistema y mantener la calidad del código. Estas pruebas se desarrollaron utilizando el framework MSTest, junto con la técnica de Mocks para simular dependencias y aislar la lógica a evaluar. Además, se aplicó la metodología AAA que sería el Arrange-Act-Assert, lo que permitió estructurar cada caso de prueba de forma clara y ordenada. En total se realizaron 24 pruebas distribuidas en diferentes módulos del controlador, con un tiempo de ejecución total de 348 ms, lo que demuestra un rendimiento eficiente y la solidez de la lógica implementada. La incorporación de estas pruebas es fundamental ya que permite detectar posibles errores en futuras iteraciones y preservar la integridad del software frente a cambios o nuevas funcionalidades

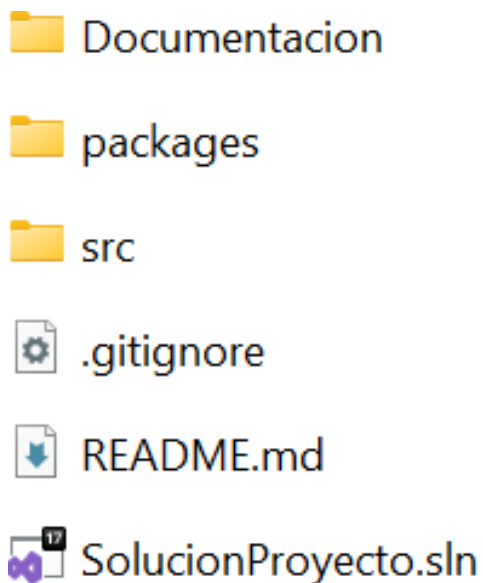
Prueba	Duración	Rasgos
▲ ✓ Controlador.Test (24)	348 ms	
▲ ✓ Controlador.Tests (4)	2 ms	
▸ ✓ LoginControladorTests (4)	2 ms	
▲ ✓ ControladorTest (6)	264 ms	
▸ ✓ UsuarioControladorTest (6)	264 ms	
▲ ✓ ControladorTests (4)	22 ms	
▸ ✓ GrupoControladorTests (4)	22 ms	
▲ ✓ Tests.Controlador (10)	60 ms	
▸ ✓ GastosControladorTests (10)	60 ms	

▼ Name	▼ Covered	▼ Uncovered	▼ Coverable	▼ Total	▼ Line coverage	
+ Controlador.Test	335	2	337	674	99.4%	<div></div>
- Controlador	165	6	171	507	96.4%	<div></div>
Controlador.GrupoControlador	32	3	35	115	91.4%	<div></div>
Controlador.UsuarioControlador	29	3	32	94	90.6%	<div></div>
Controlador.GastosControlador	93	0	93	253	100%	<div></div>
Controlador.LoginControlador	11	0	11	45	100%	<div></div>

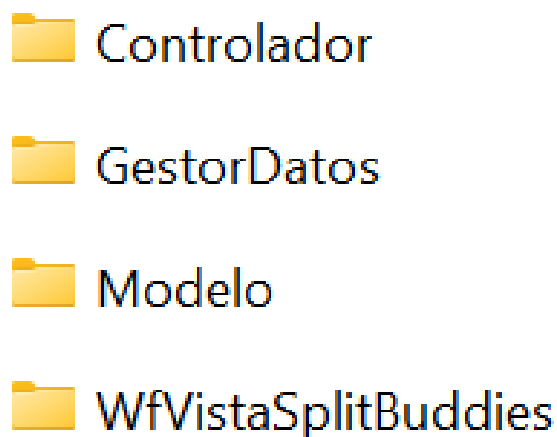
Organización del Proyecto

Para mantener una buena claridad se implementó una estructura de carpetas organizadas según lo visto en clases.

La carpeta principal del repositorio contiene los siguientes directorios:



Dentro de la carpeta SRC se encuentra el código fuente, su estructura MVC y la capa adicional de persistencia denominada Gestor Datos.



Tecnologías Utilizadas

- Se utilizó el lenguaje C#
- Windows Forms para las pantallas
- Archivos JSON que son la simulación de la base de datos
- Git y GitHub
- Doxygen
- .NET 9
- Trello
- Sonar Analyzer para las buenas prácticas de codificación.

Análisis de Resultados

En esta segunda iteración del Gestor de Gastos se logró implementar una versión más robusta y funcional del sistema, corrigiendo y ampliando las limitaciones detectadas en el proyecto anterior.

- Se garantizó que cada usuario únicamente pueda visualizar los grupos a los que pertenece, fortaleciendo la seguridad y personalización del sistema.
- Se añadió la funcionalidad de edición y eliminación de gastos, incluyendo la posibilidad de modificar integrantes, lo que otorga mayor flexibilidad en la gestión de la información.
- Se desarrolló el balance general de cada usuario, con reportes por usuario y grupo que muestran un panorama completo de deudas y pagos.
- Se consideró la opción de que algunos usuarios participen o no en determinados gastos, dividiendo los montos de manera proporcional según corresponda.
- Se implementaron pruebas en la capa del controlador, alcanzando la meta mínima de cobertura establecida.
- La documentación del código fue generada mediante Doxygen, facilitando la comprensión del diseño y la estructura del proyecto.

Aprendizajes

El desarrollo del Proyecto 2 permitió afianzar conocimientos técnicos y de gestión, entre los que destacan:

- Profundización en el uso de los principios SOLID, aplicándolos de forma más estricta para garantizar una arquitectura clara, desacoplada y mantenible.
- Implementación de patrones de diseño, incluyendo el patrón Singleton, que se utilizó en componentes críticos del sistema para garantizar instancias únicas y controladas.
- Fortalecimiento en el uso del patrón MVC, logrando una separación más clara entre la lógica de negocio, la presentación y el acceso a datos.
- Uso de Git y GitHub como herramientas de control de versiones, manteniendo un flujo de trabajo ordenado con ramas organizadas y un historial verificable.
- Implementación de reportes y generación de balances exportables, entendiendo la importancia de ofrecer información clara y descargable en diferentes formatos.
- Práctica de pruebas unitarias en la capa del controlador, mejorando la calidad y confiabilidad del sistema.
- Mayor dominio de herramientas de planificación como Trello, que permitió gestionar los PBIs, dar seguimiento al avance y priorizar tareas de forma profesional.
- Análisis de calidad del código con Sonar Analyzer: Para detectar errores, mejorar seguridad y mantener buenas prácticas de codificación.

Conclusiones

El Gestor de Gastos v2 representa una evolución significativa respecto a la primera versión. Gracias a la incorporación de nuevas funcionalidades y la aplicación de principios avanzados de programación orientada a objetos, se obtuvo un sistema más completo, confiable y escalable.

El uso de patrones de diseño, en particular el Singleton y el MVC, junto con las buenas prácticas de Clean Code y SOLID, reforzó la mantenibilidad del proyecto y sentó bases sólidas para futuras ampliaciones.

La generación de documentación con Doxygen consolidaron la profesionalidad del trabajo realizado.

En conclusión, este proyecto no solo permitió construir un sistema funcional, sino también aplicar de forma práctica conocimientos clave en desarrollo de software, documentación, pruebas y gestión de proyectos.

Cosas por mejorar o que no se hicieron

- Optimizar la interfaz web, agregando un diseño más atractivo y adaptable a dispositivos móviles.
- PBIs que no se lograron completar y que pudieron ser muy funcionales, como el ingresar a un integrante por invitación.
- Ampliar las opciones de reportes, incorporando filtros avanzados y visualizaciones gráficas de balances.
- Integrar persistencia en bases de datos SQL u otros servicios externos, para asegurar escalabilidad y almacenamiento seguro a largo plazo.