

Universidad Politécnica Internacional

Curso: Técnicas de Programación

Profesor: Luis Felipe Mora Umaña

Proyecto 1

SplitBuddies

Estudiantes

Daniel Benavides Robles

Freddy Martinez Espinoza

II Cuatrimestre, 2025

Índice

1. Introducción.....	4
2. Decisiones de diseño.....	5
2.1. Cómo usar y ejecutar el programa.....	5
2.2. Capturas de pantalla con explicación de interacciones.....	6
2.2. Aplicación de Principios SOLID.....	9
2.3 Estructura del Proyecto.....	10
2.5 Ejemplo de Archivos JSON.....	10
3.1 Patrones de Diseño Aplicados.....	11
3.2 Revisión con SonarAnalyzer.....	12
4. Análisis de resultados.....	12
5. Conclusiones y aprendizaje.....	13

Índice de Ilustraciones

Ilustración 1. Registro de usuario: Lista de usuarios disponibles para seleccionar o crear uno nuevo.	6
Ilustración 2. Panel de grupos: Visualización y creación de grupos.	7
Ilustración 3. Formulario de gasto: Registro de nombre, monto y participantes del gasto.	8
Ilustración 4. Vista de balances: Balance de gastos y deudas entre los miembros del grupo.	9

1. Introducción

SplitBuddies es una aplicación de escritorio desarrollada en C# utilizando el paradigma de programación orientada a objetos y el patrón MVC.

Su objetivo es permitir a los usuarios gestionar gastos compartidos dentro de grupos, mostrando balances, estadísticas y reportes de gastos. Esta es la primera iteración del sistema y sienta la base para futuras ampliaciones.

2. Decisiones de diseño

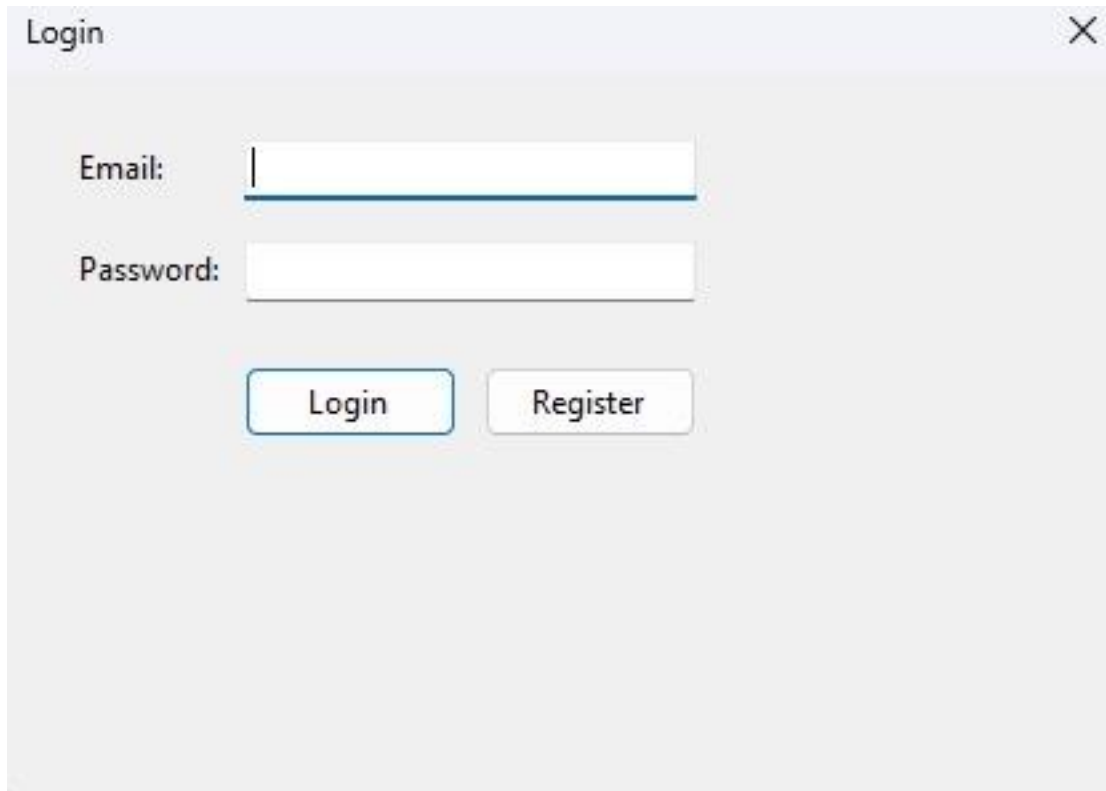
Entre las decisiones de diseño se encuentran los siguientes:

- Lenguaje: C# con Windows Forms
- Arquitectura: Patrón MVC
- Persistencia: Archivos JSON
- Extensibilidad: Código modular preparado para futuras iteraciones

2.1. Cómo usar y ejecutar el programa

- Abrir el proyecto en Visual Studio
- Ejecutar la solución (F5)
- Registrarse como usuario nuevo
- Crear o unirse a un grupo
- Agregar gastos
- Consultar balances desde el panel de estadísticas
- Explorar reportes filtrando por fechas

2.2. Capturas de pantalla con explicación de interacciones



A screenshot of a 'Login' dialog box. The dialog has a title bar with the word 'Login' on the left and a close button (an 'X' icon) on the right. Inside the dialog, there are two input fields: the first is labeled 'Email:' and the second is labeled 'Password:'. Below these fields are two buttons: 'Login' and 'Register'. The 'Login' button is highlighted with a blue border, while the 'Register' button has a grey border. The background of the dialog is a light grey.

Ilustración 1. Registro de usuario: Lista de usuarios disponibles para seleccionar o crear uno nuevo.

Register

×

Registro de usuario

Nombre:

Correo electrónico:

Contraseña:

Confirmar

Registrar

Cancelar

Ilustración 2. Panel de grupos: Visualización y creación de grupos.

The image shows a web form titled "Agregar Gasto" (Add Expense). The form is displayed in a window with standard OS controls (minimize, maximize, close). It contains the following fields:

- Grupo:** A dropdown menu.
- Pagado por:** A dropdown menu.
- Miembros incluidos:** A large text area for listing participants.
- Nombre:** A text input field.
- Descripción:** A text input field.
- Monto:** A text input field.

At the bottom of the form is a button labeled "Agregar Gasto".

Ilustración 3. Formulario de gasto: Registro de nombre, monto y participantes del gasto.

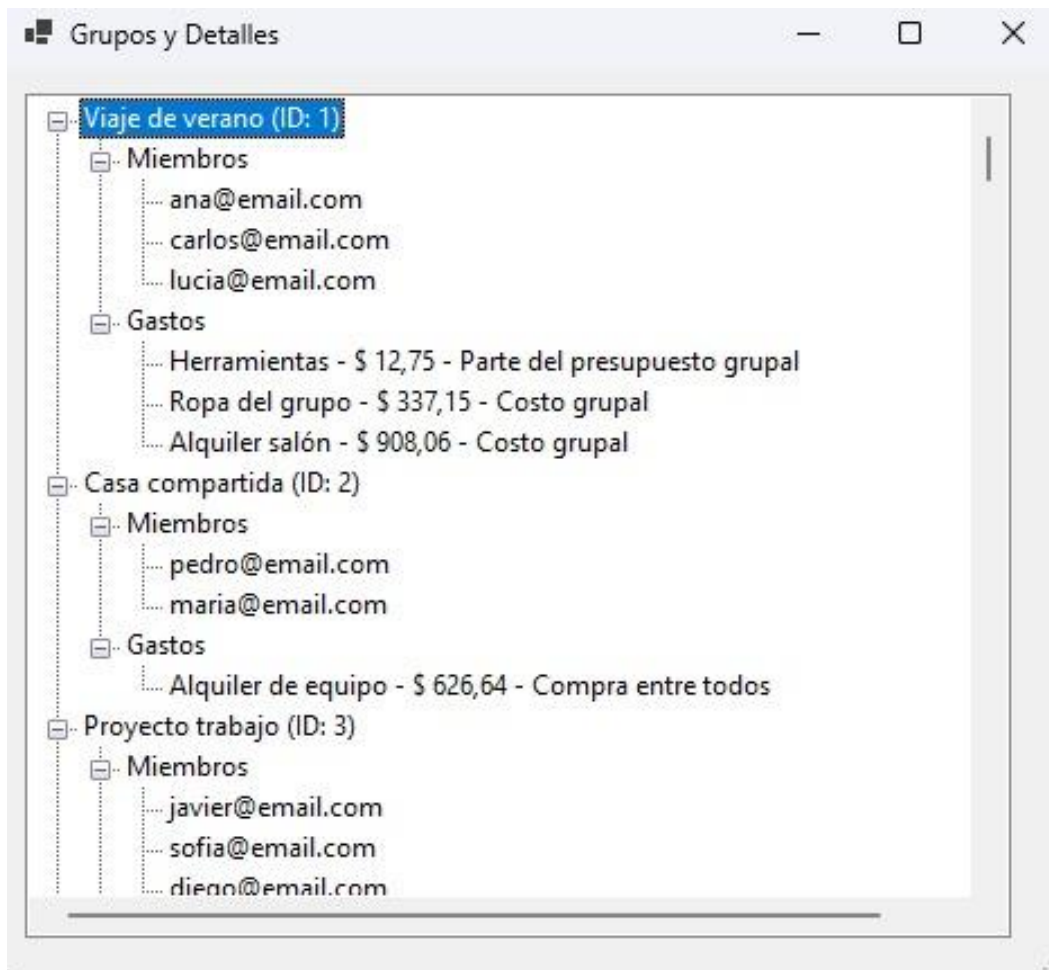


Ilustración 4. Vista de balances: Balance de gastos y deudas entre los miembros del grupo.

2.2. Aplicación de Principios SOLID

- **S – Single Responsibility Principle:**

Cada clase en el sistema fue diseñada con una responsabilidad única y específica. Por ejemplo, la clase **UserController** se encarga exclusivamente de gestionar operaciones relacionadas con los usuarios, mientras que **DataStorage** se ocupa únicamente del manejo de persistencia de datos.

- **O – Open/Closed Principle:**

Las clases del proyecto están abiertas para extensión, pero cerradas para modificación. Esto permite, por ejemplo, ampliar la funcionalidad de los controladores o modificar la fuente de datos sin alterar el código base ya existente.

- **L – Liskov Substitution Principle:**

Se garantiza que las clases derivadas o que implementan comportamientos específicos pueden sustituir sin inconvenientes a sus clases base, sin afectar el funcionamiento general

del sistema. Esta práctica se refleja en el uso coherente de las entidades del modelo en las distintas capas del proyecto.

- **I – Interface Segregation Principle:**

Aunque no se implementaron múltiples interfaces explícitas en esta iteración, la distribución de responsabilidades está estructurada de manera que cada clase interactúe solo con los métodos que realmente necesita, evitando la sobrecarga innecesaria.

- **D – Dependency Inversion Principle:**

La lógica de negocio se basa en clases concretas como **DataStorage**, pero su estructura modular permite sustituirlas fácilmente por dependencias abstraídas mediante interfaces en versiones futuras, lo cual facilita el mantenimiento, la escalabilidad y la realización de pruebas unitarias.

2.3 Estructura del Proyecto

- /Models/ – Entidades básicas como User, Group, Expense.
- /Views/ – Formularios visuales de usuario en Windows Forms (MainForm, LoginForm, etc.).
- /Controllers/ – Lógica que conecta los modelos y las vistas (UserController.cs, GroupController.cs, ExpenseController.cs).
- /Data/ – Manejo de archivos JSON y persistencia (DataStorage.cs).
- /bin/ – Archivos de compilación y datos de prueba (users.json, groups.json, expenses.json).

2.5 Ejemplo de Archivos JSON

usuarios.json

```
[
  {
    "Email": "carlos@example.com",
    "Name": "Carlos Rodríguez"
  },
  {
    "Email": "laura@example.com",
    "Name": "Laura Méndez"
  }
]
```

grupos.json

```
[
  {
    "GroupId": 1,
    "GroupName": "Viaje de verano",
    "IMAGE": "IMAGE/viajeverano.jpg",
    "Members": [
      "ana@email.com",
      "carlos@email.com",
      "lucia@email.com"
    ]
  }
]
```

gastos.json

```
[
  {
    "Name": "Pizza entre amigos",
    "Description": "Cena compartida en casa",
    "Amount": 20000,
    "PaidByEmail": "laura@example.com",
    "InvolvedUsersEmails": ["carlos@example.com", "laura@example.com"],
    "Date": "2025-07-15"
  }
]
```

3.1 Patrones de Diseño Aplicados

Modelo-Vista-Controlador (MVC):

El código está dividido en capas:

- Modelo: User.cs, Expense.cs, Group.cs.
- Vista: Formularios Windows Forms (por ejemplo, GroupForm, MainForm).
- Controlador: UserController.cs, GroupController.cs, ExpenseController.cs.

- Repository Pattern (implícito):

La clase DataStorage.cs centraliza la lectura y escritura de datos desde archivos .json,

3.2 Revisión con SonarAnalyzer

El proyecto fue analizado con SonarAnalyzer en Visual Studio, permitiendo identificar y corregir:

Métodos con demasiadas líneas que se dividieron.

Variables sin uso eliminadas.

Validación de excepciones y errores de lectura de archivos.

Uso de nombres descriptivos y consistentes.

4. Análisis de resultados

Se logra implementar de forma exitosa la base del sistema, incluyendo usuarios, grupos y gastos. El sistema calcula correctamente los balances por grupo y por usuario. La estructura del código sigue principios de POO y Clean Code.

5. Conclusiones y aprendizaje

El desarrollo de este proyecto nos permitió aplicar y reforzar diversos conceptos fundamentales en la programación orientada a objetos (POO), así como el uso práctico del patrón de arquitectura MVC (Modelo-Vista-Controlador) para lograr una correcta separación de responsabilidades.

Durante la implementación del proyecto, aprendimos a:

- Diseñar e implementar estructuras de clases que representan entidades reales (usuarios, grupos, gastos), respetando los principios de encapsulamiento, herencia y responsabilidad única.
- Utilizar el patrón MVC para dividir claramente:
 - Los modelos, que encapsulan la lógica de datos.
 - Los controladores, que manejan la lógica de negocio.
 - Las vistas, que corresponden a los formularios de Windows Forms.
- Trabajar en equipo de manera eficiente, coordinando tareas, versionando código mediante Git, y organizando el desarrollo con herramientas de gestión como Jira.
- Estructurar el proyecto desde cero, incluyendo configuración inicial, planificación de casos de uso y distribución de responsabilidades.
- Escribir código limpio, legible y mantenible, con especial énfasis en la reutilización de componentes y el uso adecuado de buenas prácticas como el manejo de excepciones, validación de entradas y separación de lógica.
- Aplicar principios de diseño como:
 - DRY (Don't Repeat Yourself) para evitar redundancias.
 - KISS (Keep It Simple, Stupid) para mantener la solución lo más clara posible.
 - Y en algunos casos, el uso de patrones como Factory, para instanciar objetos de forma flexible.

Además, el proceso nos permitió mejorar nuestra capacidad para:

- Depurar errores y entender cómo funciona la ejecución paso a paso dentro de una aplicación de escritorio.
- Integrar distintas capas del sistema (datos, lógica y presentación).
- Entender la importancia de pruebas manuales y la validación constante del flujo funcional.