Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Diseño de Aplicaciones 1

Obligatorio 2

https://github.com/IngSoft-DA1-2023-2/187884_303433_303804

Emiliano Marotta – 187884

Sebastian Borjas – 303433

Agustina Martinez – 303804

ÍNDICE:

INTRODUCCIÓN:

Presentamos la documentación del proyecto desarrollado para FinTrac, una startup enfocada en simplificar la gestión de finanzas personales. Nuestra tarea fue crear un Producto Mínimo Viable (MVP) que reflejara la visión de FinTrac de hacer que la administración financiera sea tan fácil como enviar un mensaje de texto.

A lo largo de este proyecto, nos hemos enfocado en el desarrollo de software con C# y Blazor Server, adoptando el Desarrollo Guiado por Pruebas (TDD) y adhiriendo a los principios de Clean Code. Estos enfoques no solo han sido esenciales para construir una plataforma funcional y confiable, sino que también han enriquecido nuestras habilidades como desarrolladores.

Nos enfrentamos a diversos desafíos técnicos y de diseño, superándolos mediante soluciones innovadoras y trabajo en equipo efectivo. Este proceso nos ha permitido aplicar teorías y técnicas aprendidas en un contexto práctico, ofreciendo valiosas lecciones y experiencias.

La estructura de esta documentación cubre desde la justificación del diseño hasta los detalles técnicos de la implementación y la cobertura de pruebas. Con este proyecto, no solo demostramos nuestras competencias técnicas, sino también la relevancia práctica de nuestra formación en el campo de la tecnología financiera.

1. Descripción general del trabajo

1.1. Contexto del Proyecto

FinTrac es una solución de software desarrollada para mejorar la gestión de las finanzas personales. Esta aplicación permite a los usuarios registrar y monitorizar sus transacciones financieras, categorizar gastos e ingresos, y establecer y seguir objetivos de ahorro. Nuestro objetivo principal es ofrecer una herramienta intuitiva y completa que facilite la administración financiera personal, convirtiéndola en una tarea tan sencilla como enviar un mensaje de texto.

1.2. Alcance y Objetivos

Objetivos:
El proyecto FinTrac se enfoca en los siguientes objetivos clave:

1. Facilitar el registro y seguimiento de transacciones financieras personales.
2. Permitir a los usuarios categorizar sus gastos e ingresos para un mejor análisis financiero.

3. Ofrecer funcionalidades para la creación y seguimiento de objetivos de ahorro.
4. Proporcionar reportes detallados para ayudar a los usuarios a comprender mejor su comportamiento financiero.
5. Incluir soporte para múltiples monedas, incluyendo dólares estadounidenses, euros y pesos uruguayos, facilitando así su uso en un contexto internacional.

1.3. Importancia de la Aplicación

- Desde la Perspectiva Financiera y Educativa

Desde la perspectiva financiera y educativa, esta aplicación es de vital importancia para abordar los siguientes aspectos:

- **Educación Financiera:** La aplicación ofrece a los usuarios una plataforma que no solo realiza un seguimiento de los ingresos y gastos, sino que también fomenta la educación financiera. Permite a los usuarios comprender mejor sus patrones de gasto, establecer objetivos financieros y tomar decisiones informadas.
- **Registro de Gastos**: La capacidad de llevar un registro detallado de los gastos es fundamental para la gestión financiera. Ayuda a los usuarios a identificar áreas de gasto excesivo, ajustar su presupuesto y trabajar hacia un futuro financiero más sólido.
- **Acceso Continuo a Datos Financieros**: La persistencia de datos en una base de datos garantiza que los usuarios tengan acceso constante a su información financiera, lo que facilita la toma de decisiones financieras bien fundamentadas en cualquier momento y lugar.
- **Soporte para Monedas Internacionales**: La inclusión de soporte para Euros y la consideración de futuras monedas demuestra el compromiso de la aplicación con la diversidad financiera y la adaptación a las necesidades cambiantes de los usuarios.
- **Compartir Objetivos**: La capacidad de compartir objetivos de gastos promueve la transparencia y la colaboración en la gestión financiera entre individuos y grupos, lo que puede ser especialmente útil para familias, amigos o colegas que desean alcanzar objetivos financieros comunes.

- Desde la Perspectiva Tecnológica

Esta aplicación no solo aborda desafíos financieros, sino que también representa un desafío tecnológico significativo. Desde la perspectiva tecnológica, esta aplicación es relevante por varias razones:

- **Aplicación del Desarrollo Guiado por Pruebas (TDD)**: El desarrollo guiado por pruebas es una metodología clave en este proyecto. Se enfatiza escribir pruebas antes del código, lo que garantiza la calidad y la confiabilidad del software.
- **Uso de Tecnologías Modernas**: La aplicación aprovecha tecnologías modernas como Entity Framework Core, Git con Gitflow..
- **Persistencia de Datos y Modelo de Base de Datos**: La implementación de la persistencia de datos en una base de datos Microsoft SQL Server Express 2019 y la discusión sobre cómo el diseño de clases difiere de las tablas finales.
- **Mantenibilidad y Calidad del Código**: La aplicación cumple con los principios de Clean Code y sigue patrones de diseño sólidos como SOLID y GRASP, lo que facilita futuras extensiones y mantenimiento.

- **Pruebas Unitarias y Cobertura**: Se enfatiza la cobertura de pruebas unitarias y se incluye un análisis de la cobertura de pruebas para garantizar la calidad y la robustez del código.

1.4. Funcionalidades no Implementadas o Errores Conocidos:

- Entrega 1

Durante la entrega anterior, identificamos una serie de funcionalidades y errores que quedaron pendientes debido a restricciones de tiempo y otros desafíos. Estos incluyeron ciertas funcionalidades clave tanto en el backend como en el frontend, así como un error conocido relacionado con la edición de datos de tarjetas de crédito.

Funcionalidades Pendientes Anteriormente:
En el backend, nos faltaba implementar la eliminación de usuarios invitados de workspaces y la generación de reportes.
En el frontend, las funcionalidades pendientes incluían la invitación de usuarios a workspaces, la creación de workspaces, la generación de reportes, la visualización del listado de usuarios en workspaces, la asignación de múltiples categorías a un objetivo, la eliminación de workspaces y cuentas de usuario.

Además, enfrentamos un bug en el cual la edición de los últimos 4 dígitos de la tarjeta de crédito no se reflejaba correctamente entre el frontend y el backend.

Desarrollo y Correcciones Actuales:
Actualmente, informamos que en esta fase hemos logrado implementar todas las funcionalidades pendientes y corregir el bug identificado. Las mejoras realizadas son las siguientes:

Backend:

- **Eliminación de Usuarios Invitados de un Workspace**: Implementada con éxito, mejorando la gestión de usuarios en workspaces.
- **Generación de Reportes**: Ahora es posible generar reportes sobre las finanzas personales, sobre todo del reporte de Objetivo de Gatos y de Categorías, que eran los que nos habían quedado por implementar de la entrega anterior.

Frontend:

- **Invitación de Usuarios a Workspaces**: Los usuarios pueden invitar a otros a unirse a sus espacios de trabajo.
- **Creación de Workspaces**: Implementada la capacidad de crear nuevos espacios de trabajo.
- **Generación de Reportes**: Los usuarios pueden generar todos los tipos de reportes, como lo son Reporte de Objetivo de Gastos, Reporte Gastos por Categoría, Reporte Gastos por Tarjeta, Reporte de Balance de Cuentas y Listado de Gastos. De los cuales no habíamos podido implementar ninguno en la entrega anterior.
- **Visualización de Usuarios en Workspaces**: Añadida la función para ver quiénes tienen acceso a un workspace específico.

- **Asignación de Múltiples Categorías a Objetivos**: Posibilidad de asociar múltiples categorías a los objetivos de gasto.
- **Eliminación de Workspaces y Cuentas de Usuario**: Los usuarios ahora pueden eliminar workspaces y sus propias cuentas.

Corrección de Bugs:

- **Edición de los Últimos 4 Dígitos de la Tarjeta**: Corregido el error que permitía una discrepancia entre el frontend y el backend en la edición de datos de tarjetas de crédito.

- Entrega 2

Modificación de Categorías:
Desde nuestra entrega anterior, hemos enfrentado un desafío técnico significativo relacionado con la modificación de categorías. Una vez creadas, las categorías no pueden ser modificadas debido a un problema persistente en nuestro sistema. A pesar de nuestros esfuerzos por identificar y resolver esta cuestión, aún no hemos logrado implementar una solución efectiva. Este problema continúa en la versión actual del sistema.

Abandono de Workspaces por Parte del Usuario:
En la entrega actual, hemos identificado un nuevo problema: un usuario no puede abandonar un workspace si hay otros usuarios presentes. Hemos observado que la funcionalidad no funciona correctamente cuando hay más de un usuario en el workspace. Este problema afecta la gestión eficiente de los workspaces y la capacidad de los usuarios para modificar su participación en ellos.

1.5. Estrategia de branching

En el desarrollo de FinTrac, hemos implementado una estrategia de branching planificada para gestionar eficientemente el flujo de trabajo en nuestro repositorio de GitHub. Esta estrategia ha sido fundamental para mantener la organización y la eficiencia a lo largo del desarrollo del proyecto.

Ramas Principales

- Main: La rama principal del proyecto, que contiene la versión más estable y actualizada del código. Todos los cambios significativos se fusionan aquí después de cada entrega final.
- Develop: Esta rama sirve como una etapa de desarrollo activo donde se integran las nuevas características y correcciones antes de su fusión final en la rama principal.

Ramas de Características y Correcciones
Para cada nueva feature o corrección, se crea una rama específica. Esto permite a los miembros del equipo trabajar en diferentes aspectos del proyecto de manera simultánea sin interferir con el código en las ramas principales.

Algunas de las ramas importantes incluyen:

- Feature Branches: Como feature/LoadingData, feature/UI_workspace, y feature/DailyReport, entre otras. Estas ramas se centran en el desarrollo de nuevas funcionalidades.
- Fix Branches: Ramas como fix/TransactionServiceRefactor, fix/CategoryStatus, y fix/Report, entre otras. Se utilizan para realizar correcciones específicas en el código.

Commits:
Etiquetas de Commit

- feat: Utilizada para nuevos desarrollos o adiciones de funcionalidades. Por ejemplo, un commit titulado "feat(Index) add new workspace" indica la adición de una nueva funcionalidad relacionada con el index de la aplicación.
- refactor: Empleada cuando se realizan ajustes o mejoras en funcionalidades existentes sin alterar su comportamiento. Un commit con "refactor" sugiere una mejora en el código para optimizar su rendimiento o legibilidad.
- fix: Esta etiqueta se usa para correcciones de errores. Un commit que comienza con "fix", como "fix(GoalService): set token nullable", denota la solución a un problema específico encontrado en el servicio de objetivos.
- delete: Se utiliza en casos donde es necesario eliminar archivos o código. Los commits con esta etiqueta indican la eliminación de partes del código que ya no son necesarias o relevantes para la aplicación.

Estructura del Mensaje de Commit
Cada mensaje de commit sigue una estructura donde, después de la etiqueta, especificamos entre paréntesis el archivo o carpeta que se está modificando. Seguido de los dos puntos, dejamos un comentario breve pero descriptivo sobre los cambios realizados. Esta práctica asegura que cada commit transmita claramente su propósito y alcance, facilitando el seguimiento del progreso y las revisiones de código.

Reflexiones sobre la Estrategia de Branching
Esta estrategia de branching nos ha permitido mantener un flujo de trabajo coherente y eficiente, facilitando la revisión de código, la identificación de errores y la colaboración entre los miembros del equipo. Ha sido esencial para gestionar el desarrollo paralelo de múltiples características y correcciones, asegurando que el código en 'main' se mantenga siempre en un estado estable y listo para producción.

2. Requerimientos Funcionales

En esta segunda entrega de FinTrac, se ha ampliado la funcionalidad de nuestra aplicación. A continuación, se presentan los nuevos requerimientos funcionales añadidos, junto con una descripción de cómo han sido implementados:

2.1. Reporte de Ingresos y Egresos

Descripción del Requerimiento

Este nuevo requerimiento implica generar un reporte detallado que muestre los ingresos y egresos totales diarios durante un mes seleccionado, incluyendo una representación gráfica de la evolución de estos montos.

Implementación

Hemos integrado una funcionalidad de análisis de datos que acumula y procesa transacciones diarias. Estas visualizaciones ayudan a los usuarios a comprender mejor sus hábitos de gasto e ingreso durante un mes determinado.

2.2. Soporte para Euros

Descripción del Requerimiento

Con el objetivo de internacionalizar nuestra aplicación, hemos agregado soporte para manejar transacciones en euros, además de incluir este cambio en todas las funcionalidades previamente implementadas.

Implementación

El sistema ha sido modificado para aceptar, procesar y mostrar datos en euros. Hemos actualizado las lógicas de cálculo, conversión de divisas y presentación de datos para garantizar que el soporte de euros sea integral y coherente en todas las áreas de la aplicación. Además, se ha diseñado la implementación de manera que facilita la extensión a otras monedas en el futuro.

2.3. Compartir Objetivos de Gastos

Descripción del Requerimiento

Se ha añadido una función social que permite a los usuarios compartir información sobre sus objetivos de gastos. Esto incluye generar un enlace único que puede ser visitado por otros para ver detalles específicos de un objetivo de gasto.

Implementación

Implementamos un sistema de generación de token único para cada objetivo que se desea compartir. Al activar la opción de compartir, el sistema genera un enlace con formato específico que dirige a una página de vista pública. Esta página muestra detalles del objetivo sin requerir que el visitante inicie sesión, garantizando privacidad y accesibilidad.

2.4. Persistencia de Datos

Descripción del Requerimiento

Todos los datos del sistema deben ser persistidos en una base de datos Microsoft SQL Server Express 2019, usando Entity Framework Core para el modelado y la gestión de datos.

Implementación

Seguimos el enfoque Code First de Entity Framework Core para desarrollar nuestro modelo de base de datos. Este enfoque nos permitió definir nuestras entidades y relaciones directamente en el código, facilitando la sincronización entre nuestra lógica de negocio y el

esquema de la base de datos. Las principales características de nuestra implementación incluyen:

- Modelado de Entidades: Creamos clases en C# que representan las entidades de nuestro dominio, como User, Workspace, Account, y Transaction. Estas clases se mapearon a tablas en la base de datos.
- Relaciones y Lógica de Negocio: Configuramos las relaciones entre entidades utilizando la API Fluent de Entity Framework. Esto incluyó la definición de relaciones uno-a-muchos, muchos-a-muchos, y la implementación de estrategias de herencia de tablas, como Table-per-Type (TPT) para ciertas entidades.
- Actualizaciones en Tiempo Real: La base de datos se actualiza en tiempo real con cada acción realizada por el usuario, asegurando que la información esté siempre al día.
- Herramientas de Entity Framework: Para facilitar nuestro desarrollo, instalamos y utilizamos varias herramientas y extensiones de Entity Framework, incluyendo:
    - Entity Framework Core: para el mapeo objeto-relacional y la gestión de la base de datos.
    - Entity Framework SQL Server: específico para la integración con SQL Server.
    - Entity Framework Tools: proporciona herramientas útiles para el desarrollo, como migraciones y comandos de consola.
    - Entity Framework Design: necesario para el diseño y la implementación de nuestro modelo de datos.

2.5. Videos en Youtube

Para las funcionalidades de Agregar Transacción, Reporte de objetivos de gastos y Reporte de gastos por categoría, publicamos dos videos en YouTube en los cuales se aprecia de manera clara la aplicación ejecutando correctamente estos flujos.
Los links a estos videos quedan a continuación:

- Flujo para agregar Transacción: https://youtu.be/wZRxp9fz7Io
- Flujo para Reporte de Objetivo de Gastos y Reporte de Gastos por Categoría: https://youtu.be/-DawCXkaFmE

3. Diseño y Arquitectura

Durante la primera etapa del desarrollo de FinTrac, se tomaron decisiones fundamentales para la estructuración eficiente de la solución. Una de las decisiones más críticas fue almacenar únicamente la lista de Usuarios en memoria, ya que a través de estos se accede a toda la información necesaria, con los Usuarios conteniendo los Workspaces y estos, a su vez, albergando las listas restantes.

La organización de datos presentó desafíos únicos, especialmente al determinar dónde almacenar las listas. Decidimos guardar listas de algunas clases dentro de otras para simplificar la actualización de datos y minimizar las inconsistencias. Esta decisión implicó que, en casos donde un Workspace es compartido por múltiples usuarios, el usuario administrador no puede eliminarlo directamente; en cambio, pierde acceso y la administración se transfiere a otro usuario.

Las cuentas, transacciones, objetivos, categorías y reportes se asociaron con Workspaces y no directamente con los usuarios. Esta disposición se tomó para facilitar la eliminación de usuarios sin perder la información crucial almacenada en los Workspaces. Además, establecimos que para agregar una Transacción debe haber un tipo de cambio registrado con una fecha anterior a la de la transacción en el Workspace. Los tipos de cambio, una vez registrados, solo permiten modificaciones en el valor del dólar para ese día específico, evitando así inconsistencias en los reportes.

Para las transacciones, implementamos un sistema de identificación basado en un ID autoincrementable, permitiendo la duplicación de transacciones sin generar inconsistencias. En el caso de las categorías, se pueden editar todos los atributos excepto el estado y el tipo si ya están asociadas a una transacción. En cuanto a las cuentas, estas se pueden eliminar junto con todas las transacciones asociadas.

Para esta segunda entrega, el cambio más significativo fue la integración de Entity Framework para la persistencia de datos, reemplazando el almacenamiento en memoria por una base de datos Microsoft SQL Server Express 2019. Este cambio implicó un replanteamiento profundo en la gestión de datos y la arquitectura del sistema, mejorando la eficiencia en la persistencia y recuperación de datos. Además, la integración de una nueva moneda, el euro, no presentó mayores dificultades gracias a nuestra planificación inicial, que ya contemplaba la extensibilidad para diferentes monedas. Se realizaron ajustes en algunas clases y pruebas para acomodar esta extensión.

En cuanto a la implementación de Principios SOLID y Patrones GRASP no hemos logrado, enfocarnos del todo en nuestro diseño y arquitectura con una consideración consciente hacia los principios SOLID y los patrones GRASP. A continuación, se detallan nuestras reflexiones y aplicaciones de estos principios y patrones:

Principios SOLID

- Single Responsibility Principle: Cada clase de servicio en nuestra capa de BusinessLogic ha sido diseñada para tener una única responsabilidad, manejando las operaciones de su respectivo dominio como Add, Create, Edit, y Delete.
- Open/Closed Principle: Nuestras clases de servicio actúan como interfaces, proporcionando un comportamiento extensible sin la necesidad de modificar las clases existentes. Sin embargo, se ha sugerido que tenemos un sobreuso de interfaces. Esto nos lleva a considerar la posibilidad de simplificar nuestra implementación, asegurando que las interfaces se utilizan solo cuando proporcionen beneficios claros de abstracción y extensibilidad.
- Liskov Substitution Principle: Nuestro diseño inicial no ha violado este principio, ya que las subclases pueden sustituir a sus clases base sin alterar la precisión del programa. Como es el caso de los Reportes y de las Cuentas.
- Interface Segregation Principle: La eliminación de la interfaz genérica Operations, que no tenía uso práctico, es un reflejo de nuestra alineación con este principio, evitando que nuestras clases implementen interfaces que no utilizan.
- Dependency Inversion Principle: La ausencia de una interfaz para desacoplar la base de datos de los servicios no cumple este principio. La implementación directa de

FintracContext en nuestros servicios crea una dependencia concreta en lugar de una abstracción.

Patrones GRASP

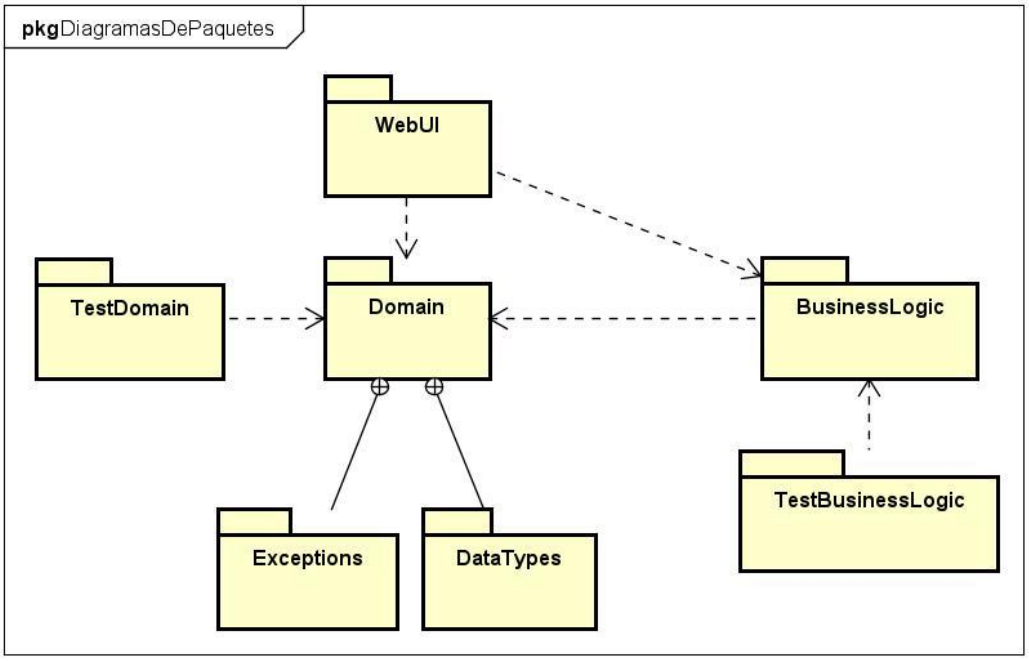
- Controlador: Los servicios de BusinessLogic funcionan como controladores, manejando la lógica de las operaciones del usuario y la comunicación con FintracContext.
- Creador: Hemos seguido este patrón al permitir que los servicios creen instancias de entidades para operaciones como Add y Create.
- Alta Cohesión: Al mantener las operaciones relacionadas dentro de servicios dedicados, hemos asegurado una alta cohesión.
- Bajo Acoplamiento: A pesar de que la falta de interfaces aumenta el acoplamiento, cada servicio se mantiene relativamente desacoplado de otros servicios, limitando las interacciones a través de FintracContext.
- Indireccion: No se ha implementado una capa de indirección completa debido a la interacción directa con FintracContext, lo que era parte de nuestro objetivo inicial de diseño.

Reflexiones y Mejoras:

- Se planeó una interfaz para desacoplar la base de datos, pero las limitaciones de tiempo impidieron su implementación.
- A pesar de no seguir completamente los principios SOLID, especialmente en términos de OCP y DIP, nuestra arquitectura actual ha funcionado adecuadamente para los requerimientos actuales del proyecto.

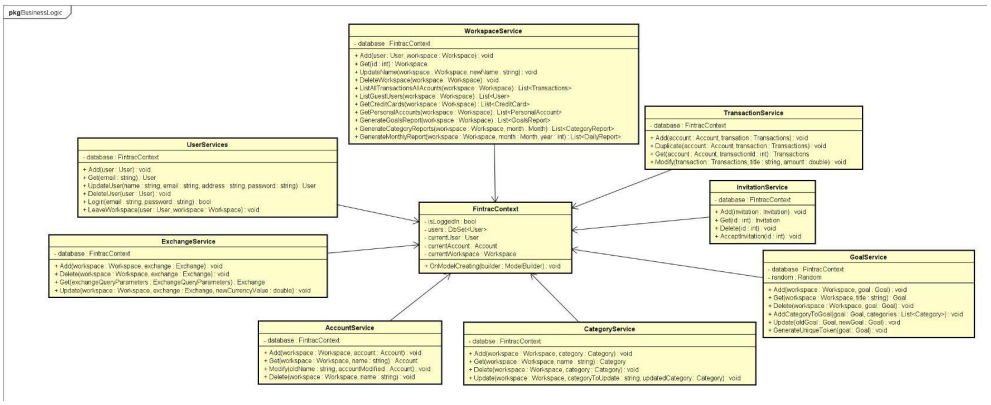
3.1. Diseño de Paquetes

A continuación presentamos el diseño para el diagrama de paquetes. En este diagrama se ilustra la arquitectura de la aplicación, destacando la clara separación entre la interfaz de usuario (WebUI), la lógica de negocio (BusinessLogic), y las clases de dominio (Domain). Se observa que cada paquete tiene una función específica, lo que facilita el mantenimiento y la escalabilidad. Las pruebas unitarias están representadas por TestDomain y TestBusinessLogic, lo que indica un enfoque en la calidad del código. Los paquetes adicionales para excepciones y tipos de datos sugieren una atención al manejo de errores y a la definición precisa de la información manejada.

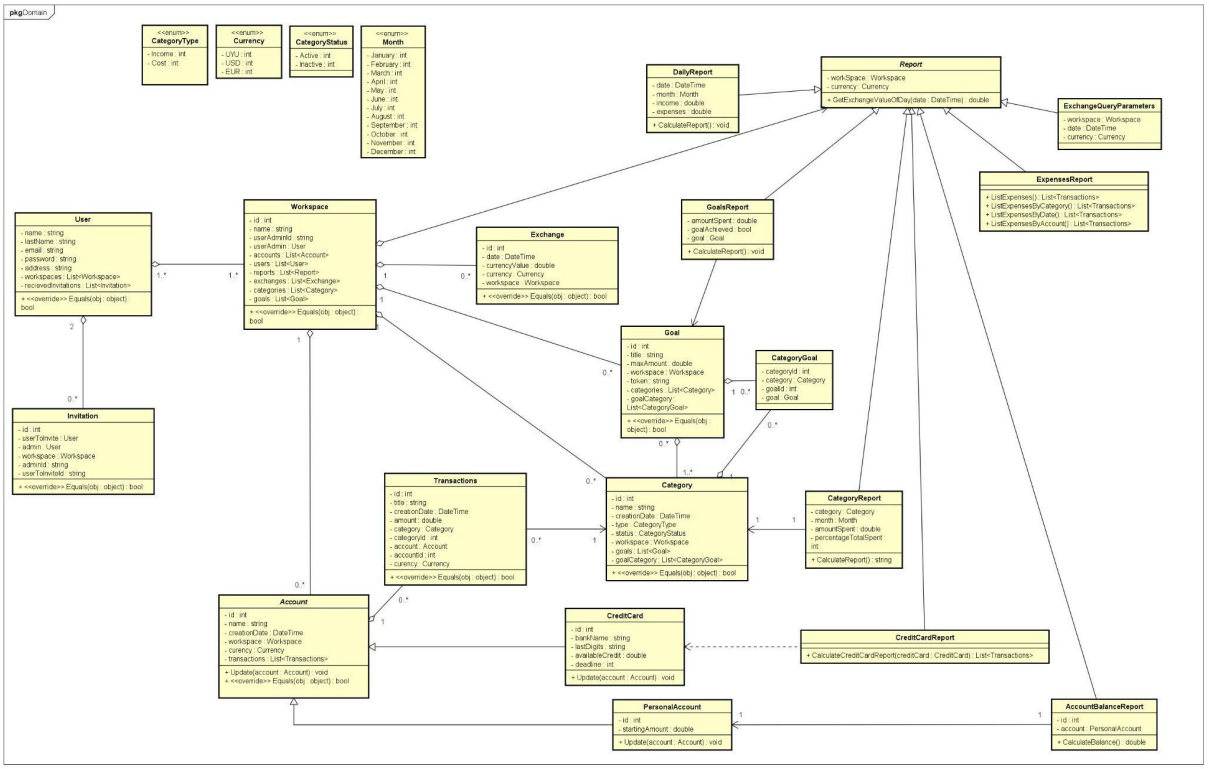


3.2. Diagramas de clases

Presentamos también el diagrama de la lógica de negocio, que ilustra cómo nuestras clases de servicio, como UserService, WorkspaceService, y AccountService, interactúan con la base de datos a través del contexto de Entity Framework FintracContext. Cada servicio encapsula la lógica específica para su dominio y utiliza FintracContext para realizar operaciones de persistencia de datos, como agregar, modificar y eliminar registros. Este diseño garantiza un acoplamiento adecuado y una clara separación de responsabilidades, facilitando la mantenibilidad y la escalabilidad del sistema



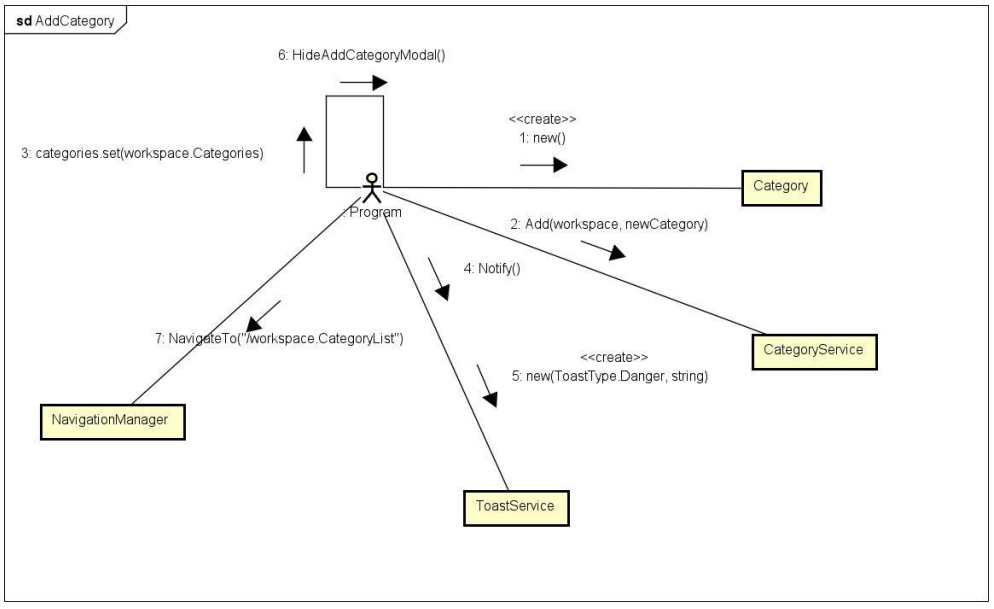
El siguiente diagrama que presentamos es el Diagrama de Clases.
Para esta entrega, hemos perfeccionado el diagrama de clases, enfocándonos en corregir y detallar las cardinalidades y relaciones entre entidades. Estos ajustes reflejan mejor las reglas de negocio y las interacciones dentro de FinTrac, proporcionando una comprensión clara y precisa de la estructura del sistema.



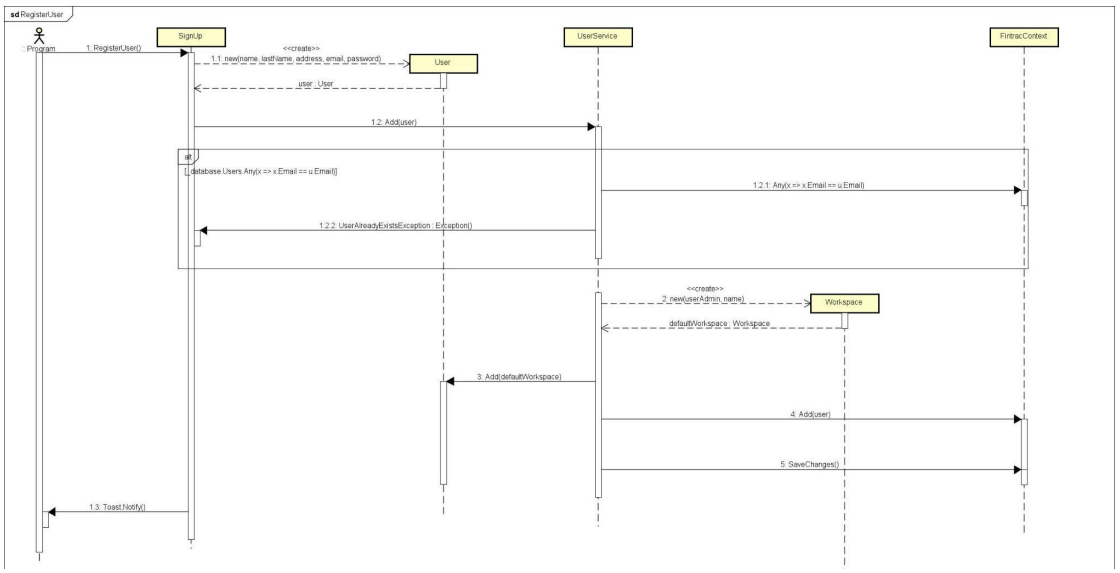
Nota: A su vez, estos diagramas también serán entregados junto con la documentación para su mayor legibilidad.

3.3. Diagramas de Interacción

Para esta sección hemos incluido solo 2 diagramas de Interacción para ilustrar comportamientos críticos del sistema.
El primero es un Diagrama de Comunicación que describe el proceso de agregar una categoría, enfatizando la interacción entre los componentes involucrados.



El segundo es un Diagrama de Secuencia que detalla los pasos involucrados en el registro de un usuario, ofreciendo una visión detallada del flujo de información.



Ambos diagramas reflejan funcionalidades clave y resaltar la interacción del sistema. A pesar de no incluir un tercer diagrama de interacción, hemos invertido un esfuerzo considerable en asegurar que los dos proporcionados demuestren con precisión y claridad las operaciones esenciales de nuestra aplicación.

3.4. Modelo de tablas

Implementación de Persistencia con Entity Framework Core y SQL Server Express 2019

En nuestro proyecto, utilizamos Microsoft SQL Server Express 2019 como sistema de gestión de bases de datos y Entity Framework Core como ORM para la persistencia de datos.

Entity Framework Core facilita la interacción con la base de datos a través de un contexto, denominado FintracContext, que actúa como un puente entre nuestro dominio de clases y la base de datos. Este contexto define conjuntos de entidades (DbSet<T>) para cada tipo de entidad de dominio que necesitamos persistir, como User, Workspace, Account, Transaction, entre otros.

Para definir la estructura y las relaciones de nuestra base de datos, utilizamos el método OnModelCreating en FintracContext. Aquí especificamos configuraciones como las estrategias de herencia de tabla (Table-per-Type, TPT) para las entidades Account, CreditCard, y PersonalAccount, así como las relaciones entre las diferentes entidades, incluyendo las claves foráneas y el comportamiento en caso de eliminación (OnDelete).

Las migraciones de Entity Framework Core proporcionaron un medio para aplicar cambios incrementales en la base de datos. La primera migración, InitialMigration, creó las tablas

necesarias en la base de datos basándose en el modelo definido en FintracContext.

Diseño de la Base de Datos y Lógica de Persistencia

El diseño de la base de datos refleja la estructura de nuestras clases de dominio. Sin embargo, hubo decisiones particulares tomadas durante el modelado que afectaron la estructura de las tablas:

- Herencia de Cuenta: Para las entidades CreditCard y PersonalAccount, que heredan de Account, optamos por una estrategia TPT, lo que resultó en la creación de tablas separadas para cada subclase, manteniendo una jerarquía de herencia clara y evitando la duplicación de columnas.

- Relaciones y Restricciones: Las restricciones de integridad, como las relaciones de uno-a-muchos y muchos-a-muchos, se mapearon cuidadosamente para asegurar la coherencia de los datos. Por ejemplo, las invitaciones (Invitation) y las transacciones (Transaction) están vinculadas a usuarios (User) y workspaces (Workspace) de tal manera que reflejan las relaciones del dominio de la aplicación.

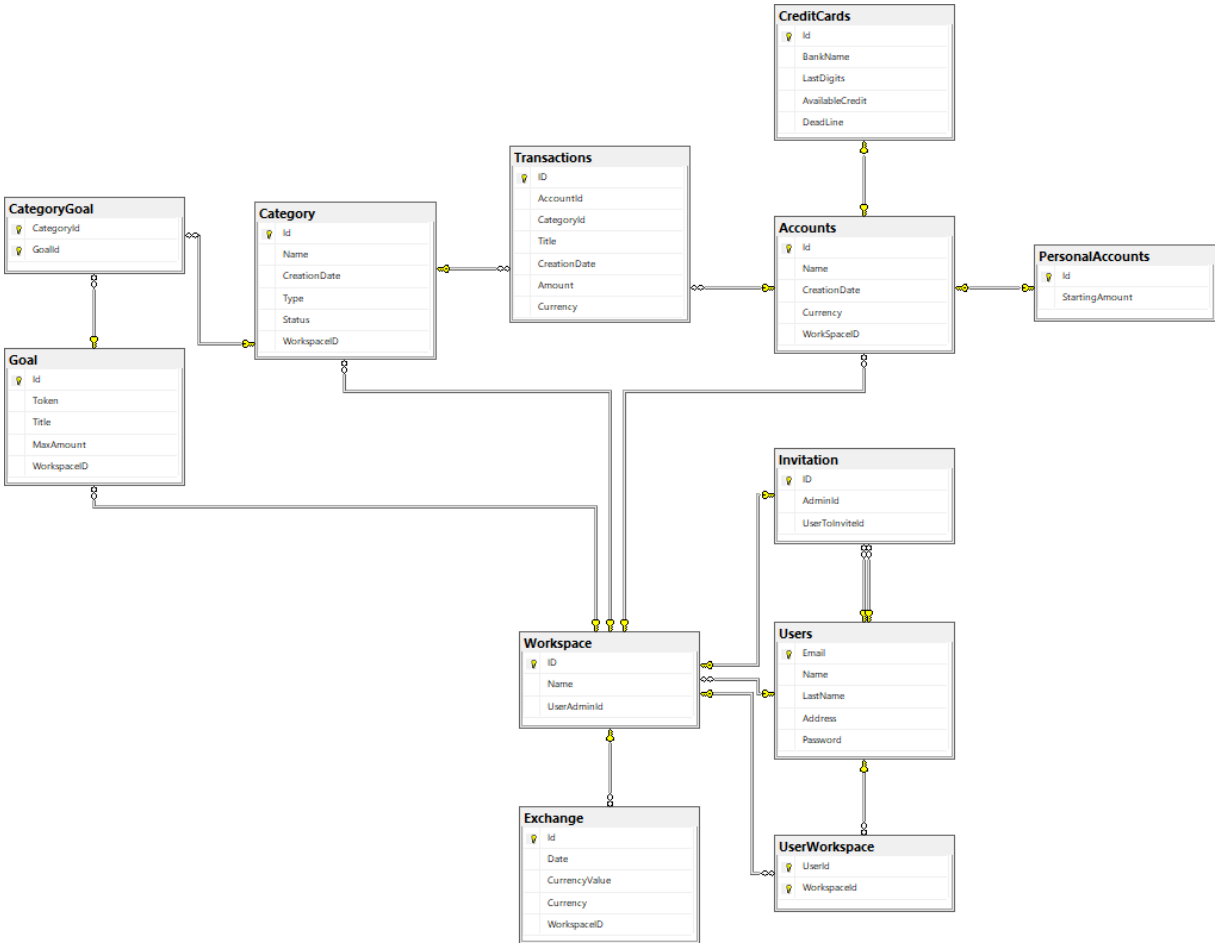
- Eliminación en Cascada y Restricción: Determinadas relaciones requerían un tratamiento especial al eliminar registros. Por ejemplo, al eliminar un Workspace, todas sus cuentas asociadas (Account) también se eliminan, pero al eliminar un User, las invitaciones (Invitation) que ha enviado no se eliminan para mantener la integridad de las invitaciones recibidas por otros usuarios.

- Tablas de Unión para Relaciones Muchos-a-Muchos: En el caso de las relaciones muchos-a-muchos, como entre Category y Goal, y entre User y Workspace, se introdujeron clases de asociación, CategoryGoal y UserWorkspace. Estas clases no son parte del dominio de lógica de negocio, pero son cruciales para el mapeo de relaciones en la base de datos. Estas entidades intermedias actúan como tablas de unión, permitiendo representar estas complejas relaciones.

- Estrategia Eager Loading: Para mejorar la eficiencia en la recuperación de datos y reducir el número de consultas a la base de datos, implementamos la estrategia de Eager Loading en todas las tablas. Esto significa que cuando se consulta una entidad principal, todas sus entidades relacionadas se cargan simultáneamente. Esta estrategia es especialmente útil en situaciones donde se sabe que se necesitarán los datos relacionados, ayudando a evitar múltiples viajes ida y vuelta a la base de datos y mejorando el rendimiento general de la aplicación.

3.4.1. Diagrama del modelo de tablas

El diagrama del modelo de tablas adjunto representa la estructura relacional de nuestra base de datos como se define en SQL Server Management Studio.



Este modelo de base de datos fue generado automáticamente por Entity Framework Core a través de migraciones y refleja fielmente la estructura y relaciones definidas en nuestro contexto FintracContext. La selección de tipos de relaciones y las configuraciones de herencia han sido esenciales para mantener la coherencia y la integridad referencial a través de la aplicación.

3.4.2. Discusiones

Discusión sobre Discrepancias:
Hemos identificado y abordado las discrepancias entre el diseño de clases y las tablas de la base de datos. Estas discrepancias surgieron principalmente debido a optimizaciones en la persistencia de datos y requisitos de integridad referencial. Cada cambio ha sido justificado para garantizar que el diseño final sea el más adecuado para nuestras necesidades.

3.5. Mecanismos generales

Para la explicación de los mecanismos generales y descripción de las principales decisiones de diseño tomadas ya han sido redactadas en la Sección Diseño y Arquitectura.

Pero a los efectos de explicar algo breve en esta sección, en nuestro diseño, la interfaz de usuario se comunica directamente con los servicios correspondientes a cada clase del dominio. Cada servicio encapsula la lógica de negocio específica y actúa como el enlace directo con nuestra capa de persistencia.

Los servicios específicos del dominio interactúan con la base de datos a través del contexto de Entity Framework Core, lo que nos permite gestionar las operaciones de datos de manera eficiente. Al centralizar la interacción con la base de datos en estos servicios, garantizamos que cualquier acción realizada en la interfaz de usuario se refleje de manera coherente y segura en el almacenamiento de datos.

Para el manejo de errores, hemos implementado excepciones personalizadas que son lanzadas por los servicios cuando se encuentran con condiciones inesperadas. Estas excepciones son luego capturadas en la capa de UI, donde se proporciona la información adecuada al usuario sobre el error ocurrido. Este sistema de manejo de errores nos permite mantener la usabilidad de la aplicación y facilita la depuración durante el desarrollo y el mantenimiento.

3.6. Asignación de responsabilidades

Enfoque Inicial en la Primera Entrega:
Inicialmente, el equipo comenzó el proyecto con un enfoque colaborativo, realizando análisis conjuntos y entrando en el tema del obligatorio. Cada miembro del equipo asumió la responsabilidad de desarrollar clases específicas utilizando TDD, lo que permitió una sólida comprensión y aplicación de esta metodología desde el inicio.
Evolución y Especialización de Responsabilidades:

- Análisis y Diseño Conjunto: Todos los miembros del equipo participamos activamente en el análisis de los requisitos y en la conceptualización de la solución.
- Inicio con TDD: Repartimos clases entre los tres miembros del equipo para iniciar la implementación mediante TDD, asegurando que todos adquiriéramos experiencia práctica con esta metodología.
- Desarrollo Progresivo: Desarrollamos de manera incremental las clases, la lógica de negocio y la interfaz de usuario, comenzando con la persistencia de datos en memoria.

A medida que avanzamos, nos dimos cuenta de la necesidad de optimizar nuestra distribución de tareas para cumplir con los plazos del proyecto:

División de Tareas Específicas:

- Sebastián y Emiliano se concentraron en funcionalidades más complejas a través de pair programing, abordando conjuntamente los desafíos más grandes.
- Agustina adelantó el desarrollo de la interfaz de usuario y la integración con la lógica de negocio, verificando la correcta funcionalidad del sistema.

Estrategia de Trabajo en la Segunda Entrega:
Para la segunda entrega, debimos ajustar nuestra estrategia, y la separación de responsabilidades se hizo aún más evidente:

- Agustina se encargó de finalizar el desarrollo del frontend y de implementar nuevas funcionalidades en el backend según fuera necesario.
- Sebastián se centró en la finalización de los reportes y en la revisión de la lógica de negocio en conjunto con Emiliano.

- Integración de Entity Framework: Una vez completados los puntos pendientes de la entrega anterior, Sebastián lideró la integración de Entity Framework.
- Refactorizaciones y Nuevos Requerimientos: Emiliano y Agustina se enfocaron en las refactorizaciones y en los nuevos requerimientos.
- Persistencia de Datos: Hacia la fase final, Sebastián y Emiliano continuaron concentrándose en la persistencia de los datos.
- Mejora de la Documentación: Agustina asumió la responsabilidad de mejorar la documentación, reconociendo su importancia crítica para el proyecto.
- Práctica de Clean Code: Todos los miembros del equipo se esforzaron en aplicar los principios de Clean Code, mejorando así la legibilidad y mantenibilidad del código que entregamos.
- Además, Sebastián se dedicó a la elaboración y ajuste de los diagramas UML, mejorando cardinalidades y detalles pendientes de la primera entrega.

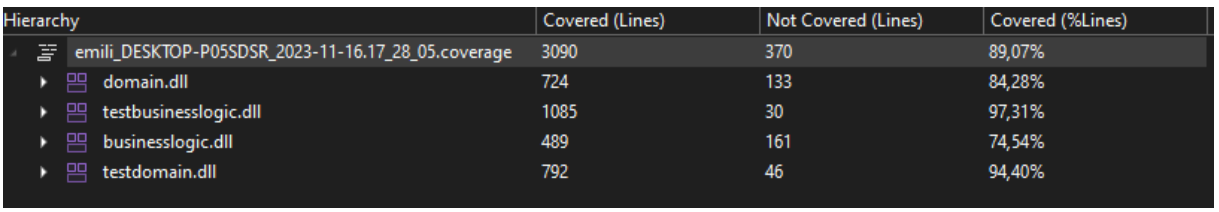
Reflexión:
La asignación y adaptación de responsabilidades ha sido clave en nuestro proceso de desarrollo. Esta estrategia no solo ha permitido maximizar la eficiencia del equipo, sino que también ha garantizado que cada área del proyecto reciba la atención y el expertise necesarios. Estamos orgullosos de lo que hemos logrado como equipo y confiamos en que estas bases sólidas serán fundamentales para el éxito continuo.

4. Pruebas Unitarias y Cobertura

Desde la primera entrega del desarrollo de FinTrac, adoptamos el enfoque de Desarrollo Guiado por Pruebas (TDD) como metodología de trabajo. Esta decisión fue fundamental para asegurar una base sólida y confiable para nuestro software. Durante la primera entrega, todas las características y funcionalidades se desarrollaron, comenzando con la creación de pruebas unitarias. Esto no solo mejoró la calidad del código, sino que también nos permitió diseñar un sistema más mantenible.

En esta segunda entrega del desarrollo, hemos mantenido el uso de TDD. Al enfrentarnos a desafíos más complejos, como la implementación de nuevas funcionalidades, el TDD ha sido crucial para asegurar que cada incremento del sistema se construya sobre una base probada y fiable.

Nuestro objetivo ha sido mantener o superar el estándar que establecimos en la primera entrega, donde alcanzamos una cobertura del 92% de las líneas de código. Sin embargo, debido a las limitaciones de tiempo y la complejidad creciente de las funcionalidades, reconocemos que la cobertura de pruebas ha disminuido ligeramente.



Hierarchy	Covered (Lines)	Not Covered (Lines)	Covered (%Lines)
emili_DESKTOP-P05SDSR_2023-11-16.17_28_05.coverage	3090	370	89,07%
domain.dll	724	133	84,28%
testbusinesslogic.dll	1085	30	97,31%
businesslogic.dll	489	161	74,54%
testdomain.dll	792	46	94,40%

La cobertura total de 89.07% es testimonio de nuestro esfuerzo constante por la calidad y la fiabilidad, aunque no alcanzamos el umbral previamente establecido. A pesar de no seguir TDD en algunas funciones más recientes, continuamos valorando el TDD como un estándar de calidad.

Beneficios Observados:

- Reducción de Bugs: El uso continuo de TDD ha resultado en una notable disminución de errores y problemas en el código, lo que se refleja en la estabilidad y confiabilidad del sistema.
- Facilidad en la Integración de Nuevas Funciones: TDD nos ha permitido integrar nuevas funcionalidades con mayor confianza, sabiendo que cualquier impacto negativo sería detectado rápidamente por nuestras pruebas existentes.
- Mejora Continua del Código: El refactoring constante, un componente clave del TDD, ha permitido que nuestro código evolucione y mejore continuamente en términos de claridad, eficiencia y mantenibilidad.

Desafíos y Estrategias:

A pesar de los beneficios, mantener el TDD a lo largo de la segunda entrega ha presentado sus desafíos, especialmente en lo que respecta a:

- Pruebas de Interfaz de Usuario y Persistencia de Datos: Estas áreas han requerido un enfoque más creativo y herramientas especializadas para asegurar que se mantengan dentro del marco del TDD.
- Alta Cobertura de Pruebas: Hemos tenido que ser meticulosos en garantizar que la cobertura de pruebas se mantenga alta, incluso cuando se añaden nuevas funcionalidades complejas.

Conclusión:

- El TDD ha sido y seguirá siendo un componente integral de nuestro proceso de desarrollo. Creemos firmemente que esta metodología no solo mejora la calidad técnica del producto, sino que también alinea nuestro desarrollo con las mejores prácticas de la ingeniería de software. Estamos comprometidos a seguir este enfoque en futuras fases de desarrollo de FinTrac.

5. Instalación y Configuración

Esta sección proporciona una guía detallada para la instalación y configuración de FinTrac, asegurando un proceso de puesta en marcha eficiente y sin problemas.

Requisitos Previos

Antes de comenzar la instalación, asegúrese de tener:

1. Microsoft SQL Server Express 2019 instalado y en ejecución.
2. .NET Core SDK versión 6.0 instalado en su máquina.

Pasos para la Instalación
Descarga del Código Fuente:

1. Acceder al repositorio en GitHub: https://github.com/IngSoft-DA1-2023-2/187884_303433_303804
2. Descargue el código fuente correspondiente al último release etiquetado para la entrega actual.

Configuración de la Base de Datos:

1. Localizar el archivo ZIP proporcionado (en la entrega por Gestión) que contiene los backups de la base de datos.
2. Restaure la base de datos utilizando el archivo .bak.

Configuración del Proyecto:

1. Localice la carpeta 'publish' que se ha generado y subido al repositorio tras la compilación y publicación desde Visual Studio.

Configuración de Strings de Conexión:

1. En la base del directorio WebUI, cree un nuevo archivo `appsettings.json` con la siguiente estructura:

Por ejemplo:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data
Source=<DESKTOP-XXXXX>\\SQLEXPRESS;User
ID=<usuario>;Password=<password>;Database=<database-name>; Connect
Timeout=10;TrustServerCertificate=true"
  },

  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Se debe editar el nombre del equipo, también el nombre del usuario y la contraseña.

2. Asegúrese de reemplazar los valores **<DESKTOP-XXXXX>** por el nombre de su equipo, **<usuario>** por el nombre de usuario creado en SQL Server, **<password>** por la contraseña del usuario de SQL Server, y **<database-name>** por el nombre de la base de datos que haya creado en SQL Management Studio.

Ejecución del Proyecto:

1. Desplegar la carpeta publish o ejecutar la aplicación directamente.

Verificación Post-Instalación

- Una vez iniciada la aplicación, abra su navegador y navegue a http://localhost:[puerto]/ para acceder a FinTrac.
- Verifique que la aplicación se conecte correctamente a la base de datos y que todas las funcionalidades estén operativas.

6. Conclusión y reflexión final

Logros del Proyecto

Al concluir esta fase del desarrollo de FinTrac, nos complace destacar varios logros significativos. Hemos creado una buena solución para la gestión de finanzas personales, que no solo cumple con los requisitos establecidos, sino que también ofrece una experiencia de usuario intuitiva y eficiente. La integración de tecnologías avanzadas como Entity Framework y el enfoque en un diseño extensible han sido fundamentales en estos logros.

Desafíos y Aprendizaje

El proyecto presentó desafíos únicos, especialmente en la transición a una persistencia de datos y en la adaptación a cambios de requisitos. A través de estos desafíos, hemos aprendido la importancia de una planificación flexible y la adaptabilidad en el desarrollo de software. La aplicación del TDD y los principios de Clean Code no solo mejoraron la calidad del código, sino que también facilitaron la gestión de estos desafíos.

Reflexiones sobre el Trabajo en Equipo

Cada miembro aportó sus fortalezas individuales, lo que permitió abordar una amplia gama de tareas técnicas y asegurar un progreso constante. Esta experiencia ha reforzado nuestra comprensión de la importancia de la comunicación y la cooperación en proyectos de desarrollo de software.