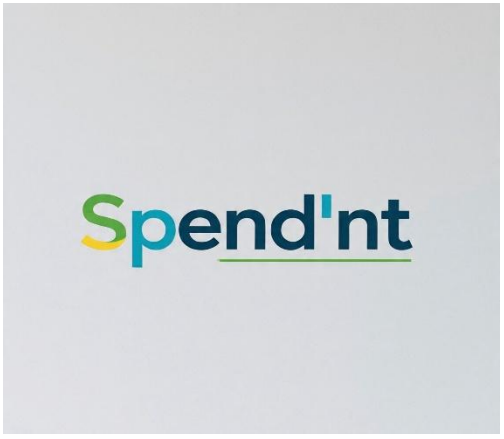

Programación de Software - Práctica Final

Spend’nt: Aplicación de Control de Gastos Personales

Steven Osorio Araque

Felipe Franco

Felipe León

Grupo X Orlando Alarcón Pérez (Docente)

Mayo 28, 2025

Resumen

Este documento presenta el desarrollo de "Spend'nt", una aplicación web progresiva para el control de gastos personales, como proyecto final de la asignatura Programación de Software.
La aplicación permite a los usuarios registrar ingresos y egresos, categorizarlos, visualizar su saldo, establecer metas de ahorro, gestionar recordatorios de gastos y visualizar su actividad financiera en un calendario.

Funcionalidades principales:

- **Gestión de Usuarios Segura:** Registro y autenticación robusta con ASP.NET Core Identity y tokens JWT.
- **Registro de Transacciones:** Ingresos y egresos con categorías personalizables.
- **Visualización del Saldo:** Cálculo automático y ajuste manual.
- **Metas de Ahorro:** Definición y seguimiento de objetivos financieros.
- **Recordatorios de Gasto:** Notificaciones para pagos futuros o recurrentes.
- **Calendario Financiero:** Vista consolidada de ingresos, egresos y metas.
- **Perfil:** Edición de información personal (nombre, apellido, foto de perfil) y guardado de cambios.

Tecnologías utilizadas:

- Backend: API RESTful con ASP.NET Core.
- Frontend: Blazor WebAssembly.
- Base de datos: SQL Server con Entity Framework Core.
- Autenticación: JWT y ASP.NET Core Identity.

Palabras Clave:
Control de Gastos, Finanzas Personales, Blazor WebAssembly, ASP.NET Core Web API, Entity Framework Core, JWT.

Índice

1. Introducción y Descripción de la Aplicación

En el contexto actual, la gestión eficiente de las finanzas personales se ha convertido en una habilidad crucial para individuos y familias que buscan estabilidad y crecimiento económico. La falta de herramientas adecuadas o la complejidad de las existentes a menudo desalientan a las personas a llevar un control riguroso de sus ingresos y gastos. "Spend'nt"surge como una solución moderna, intuitiva y accesible, diseñada para empo- derar a los usuarios en la administración de su dinero.

El propósito fundamental de Spend'nt es proporcionar una plataforma centralizada donde los usuarios puedan registrar, categorizar y analizar sus flujos financieros de manera sencilla. La aplicación busca resolver el problema común de la dispersión de información financiera y la dificultad para identificar patrones de gasto, establecer presupuestos realistas y trabajar hacia objetivos de ahorro concretos. A diferencia de hojas de cálculo manuales o aplicaciones overly-complejas, Spend'nt se enfoca en la simplicidad y la eficiencia, ofreciendo las funcionalidades esenciales sin abrumar al usuario.

Las funcionalidades principales implementadas en Spend'nt incluyen:

- **Gestión de Usuarios Segura:** Registro de nuevos usuarios y autenticación robusta mediante correo electrónico y contraseña, utilizando ASP.NET Core Identity y tokens JWT para proteger el acceso a los datos personales.
- **Registro de Transacciones:** Los usuarios pueden registrar fácilmente sus ingresos (sueldos, ventas, otros) y egresos (alimentación, transporte, ocio, etc.), asignándoles una fecha y una categoría predefinida o personalizada.
- **Categorización Personalizable:** Aunque se proveen categorías comunes, el sistema está diseñado para que (en futuras versiones o si el administrador lo permite) se puedan gestionar categorías, permitiendo una clasificación detallada y adaptada a las necesidades de cada usuario.
- **Visualización del Saldo:** Cálculo automático y presentación clara del saldo actual, mostrando el total de ingresos, el total de egresos y el balance resultante. Se contempla la posibilidad de un saldo manual para ajustes.
- **Metas de Ahorro:** Los usuarios pueden definir metas de ahorro específicas (ej. "Vacaciones", "Nuevo Portátil"), establecer un monto objetivo, una fecha límite (opcional), y registrar contribuciones para seguir su progreso.
- **Recordatorios de Gasto:** Funcionalidad para crear recordatorios de gastos futuros o recurrentes (ej. pago de alquiler, suscripciones), ayudando a los usuarios a no olvidar sus compromisos financieros.
- **Calendario Financiero:** Una vista de calendario que consolida y muestra los ingresos, egresos, fechas objetivo de metas de ahorro y recordatorios de gasto, ofreciendo una perspectiva temporal de la actividad financiera.
- **Historial de Transacciones:** Un registro detallado de todos los movimientos financieros realizados, permitiendo consultas y revisiones.

La aplicación ha sido desarrollada siguiendo una arquitectura distribuida moderna. El backend es una API RESTful construida con ASP.NET Core 7 (o la versión correspondiente), utilizando Entity Framework Core para la interacción con una base de datos SQL Server. La seguridad de la API se gestiona mediante tokens JWT generados tras la au- tenticación con ASP.NET Core Identity. El frontend es una Aplicación de Página Única (SPA) desarrollada con Blazor WebAssembly, lo que permite una experiencia de usuario rica e interactiva que se ejecuta directamente en el navegador del cliente. La comunicación entre el frontend y el backend se realiza a través de solicitudes HTTP a los endpoints de la API. Finalmente, un proyecto compartido ('Spendnt.Shared') contiene las entidades del dominio (como 'User', 'Saldo', 'Ingreso', 'Egreso', 'MetaAhorro', etc.) y los DTOs (Data Transfer Objects) utilizados para la comunicación, asegurando la consistencia y reduciendo la duplicación de código. Spend'nt no solo busca ser una herramienta de registro, sino un compañero en la toma de decisiones financieras, fomentando hábitos saludables de gasto y ahorro mediante una interfaz amigable y funcionalidades prácticas.

2. Arquitectura del Sistema

Se ha diseñado siguiendo una arquitectura de tres capas distribuida, lo que promueve la separación de responsabilidades, la escalabilidad y la mantenibilidad. Estas capas son:

- **Capa de Presentación (Frontend - Spendnt.WEB):** Desarrollada con Blazor WebAssembly, esta capa es responsable de toda la interacción con el usuario. Se ejecuta completamente en el navegador del cliente después de la carga inicial. Ges- tiona la renderización de la interfaz gráfica, la captura de entradas del usuario, la validación del lado del cliente y la comunicación con la capa de API a través de solicitudes HTTP para obtener y enviar datos. Se utilizan componentes Razor para construir una interfaz de usuario modular y reutilizable.
- **Capa de Aplicación/Servicio (Backend - Spendnt.API):** Construida como una API RESTful utilizando ASP.NET Core. Esta capa contiene la lógica de negocio principal de la aplicación. Expone endpoints seguros que el frontend consume. Sus responsabilidades incluyen:
  - Procesar las solicitudes HTTP del cliente.
  - Validar los datos de entrada.
  - Interactuar con la capa de acceso a datos para persistir y recuperar información.
  - Implementar la lógica de autenticación y autorización utilizando ASP.NET Core Identity y tokens JWT.
  - Orquestar las operaciones y aplicar las reglas de negocio.
- **Capa de Datos y Dominio (Spendnt.Shared y DataContext en API):**

Spendnt.Shared: Es una librería de clases .NET compartida entre el frontend y el backend. Contiene las definiciones de las entidades del dominio (ej. 'User',actual ahorrado, una fecha objetivo opcional y está vinculada a un 'User'

3. Modelo de Dominio y Base de Datos

El modelo de dominio de Spend'nt está diseñado para capturar la información esencial requerida para una gestión eficaz de las finanzas personales. A continuación, se describen las entidades principales y sus relaciones:

3.1. Entidades Principales

User: (Mapea a la tabla 'AspNetUsers' de Identity) Representa a un usuario registrado en la aplicación. Almacena información de autenticación gestionada por ASP.NET Core Identity (como 'Id', 'UserName', 'Email', 'PasswordHash') y propiedades personalizadas como 'FirstName' y 'LastName'. Cada usuario es el propietario de sus datos financieros.

Saldo: Entidad central que representa el balance financiero principal de un usuario. Cada usuario tiene un único 'SaldoPrincipal'. Este saldo se calcula a partir de los ingresos y
egresos asociados, aunque también permite un ajuste manual ('*totalSaldoManual*').*Mantienecol*

IngresEgsr:esos: Registra todas las salidas de dinero de un usuario. Similar a los ingresos, cada egreso tiene un monto, fecha, está asociado a un 'Saldo' y se clasifica mediante una 'Categoria'.

Categoria: Define las categorías utilizadas para clasificar tanto los ingresos como los egresos (ej. "Sueldo", .Alimentación", "Transporte", .Ocio"). En la implementación actual, las categorías son globales para todos los usuarios.

MetaAhorro: Permite a los usuarios definir objetivos financieros específicos, como ahorrar para unas vacaciones o un bien. Incluye un nombre, descripción, monto objetivo, monto

RecordatorioGasto: Ayuda a los usuarios a recordar pagos futuros o recurrentes. Contiene un título, monto estimado, fecha programada, notas y está vinculado a un 'User'.

Historial: Registra un log de las transacciones (ingresos y egresos) que afectan a un 'Saldo'. Cada entrada de historial incluye fecha, monto, tipo (ingreso/egreso), descripción y la categoría asociada.

TransaccionAhorro: (Opcional, si se implementa para un seguimiento detallado) Registraría cada contribución o retiro individual realizado hacia una 'MetaAhorro' específica.

Tablas de Identity: ASP.NET Core Identity crea automáticamente tablas adicionales para gestionar roles ('AspNetRoles'), la asignación de roles a usuarios ('AspNetUserRoles'), claims ('AspNetUserClaims', 'AspNetRoleClaims'), logins externos ('AspNetUserLogins') y tokens ('AspNetUA

3.2. Diagrama Entidad-Relación

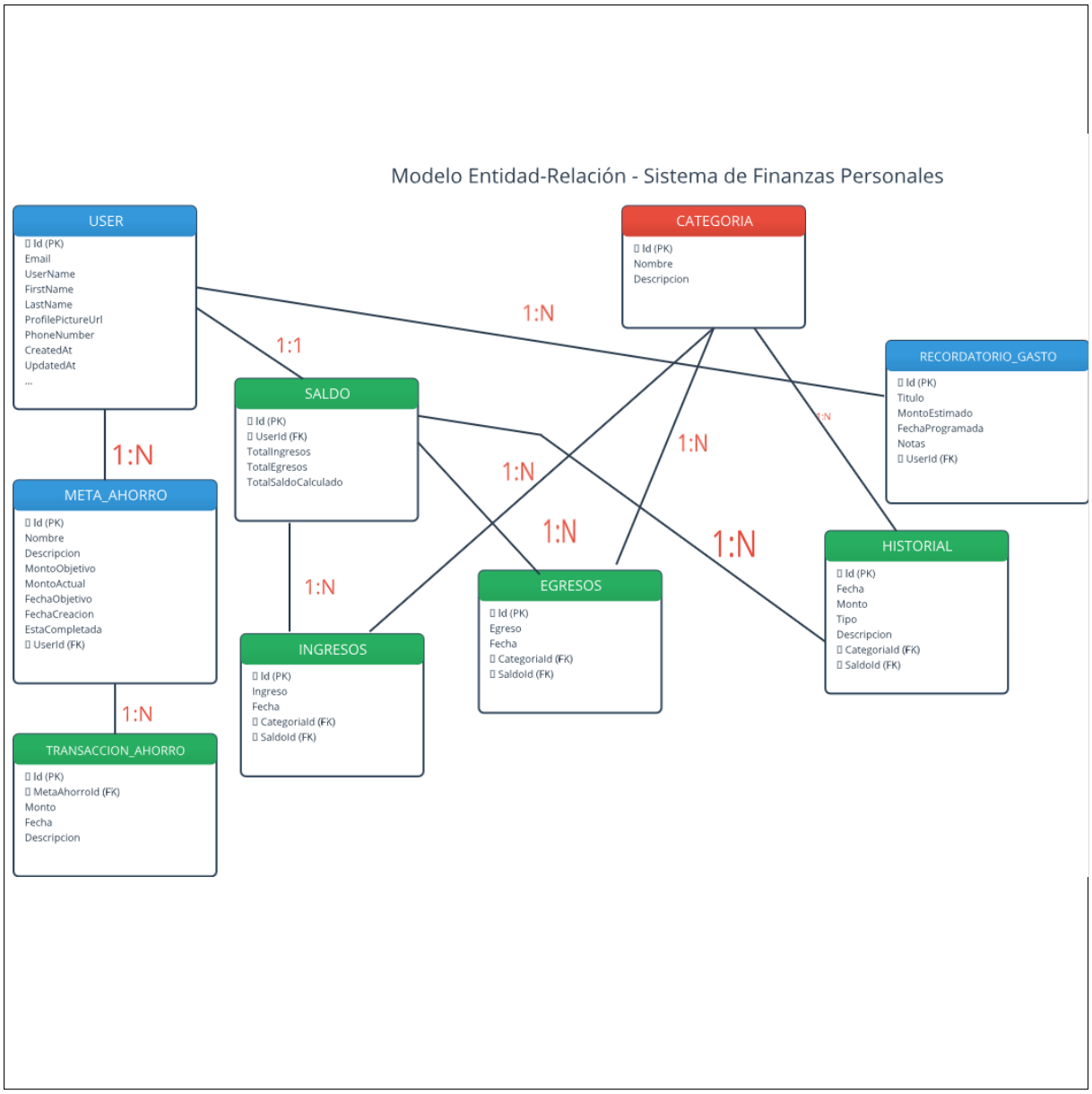

Figura 1: Diagrama Entidad-Relación de la base de datos de Spend'nt.

4. Funcionalidades Implementadas (Matriz de Funcio- nalidad)

4.1. API: Controladores y Métodos Implementados

La API de Spend'nt cuenta con controladores dedicados para cada entidad principal, gestionando las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y la lógica de negocio asociada. Todos los endpoints que manejan datos de usuario están protegidos y filtran la información basándose en el 'UserId' del usuario autenticado.

- **AuthController:** Maneja el registro de nuevos usuarios y el proceso de inicio de sesión, generando tokens JWT para la autenticación.
- **SaldoController:** Permite obtener el saldo actual del usuario (con sus ingresos y egresos asociados para el cálculo) y actualizar el saldo manual.
- **IngresosController y EgresosController:** Gestionan las operaciones CRUD para los registros de ingresos y egresos del usuario, asegurando que cada transacción se asocie al saldo del usuario correcto.
- **CategoriasController:** Administra las categorías globales (actualmente, permite listarlas; la creación/edición podría restringirse a roles de administrador).
- **MetasAhorroController:** Permite a los usuarios crear, leer, actualizar y eliminar sus metas de ahorro, además de registrar contribuciones.
- **RecordatoriosGastoController:** Gestiona los recordatorios de gasto del usuario.
- **HistorialesController:** Proporciona acceso al historial de transacciones del usuario, derivado de sus ingresos y egresos.
- **CalendarioEventosController:** Agrega datos de ingresos, egresos, metas y recordatorios para presentarlos en una vista de calendario.

Todas las operaciones que modifican o acceden a datos sensibles están protegidas por el atributo '[Authorize]' y la lógica interna de los controladores verifica la pertenencia de los datos al usuario autenticado.

4.2. Configuración de SeedDb

Se ha implementado una clase 'SeedDB' que se ejecuta al iniciar la API. Esta clase es responsable de:

1. Asegurar la creación del esquema de la base de datos si no existe.
2. Crear roles básicos del sistema (ej. .Admin", Üser") utilizando 'RoleManager'.
3. Crear uno o más usuarios de prueba (incluyendo un administrador) con contraseñas predefinidas (hasheadas) utilizando 'UserManager'.
4. Asignar roles a estos usuarios de prueba.
5. Poblar la tabla de 'Categorias' con un conjunto inicial de categorías comunes para ingresos y egresos.
6. Crear una entidad 'Saldo' inicial para cada usuario de prueba.
7. Generar datos de ejemplo para 'Ingresos', 'Egresos', 'MetasAhorro', 'RecordatoriosGasto' e 'Historiales', asociándolos correctamente con los 'UserId' y 'SaldoId' correspondientes.

Esto facilita las pruebas y el desarrollo inicial al proporcionar un entorno con datos coherentes.

4.3. Protección de API por Token (JWT)

La seguridad de la API se basa en la autenticación mediante JSON Web Tokens (JWT). La configuración se realiza en 'Program.cs' de la API:

- Se utiliza 'builder.Services.AddAuthentication().AddJwtBearer()' para configurar el middleware de autenticación JWT.
- Los 'TokenValidationParameters' se establecen para validar el emisor ('ValidIssuer'), la audiencia ('ValidAudience'), el tiempo de vida del token ('ValidateLifetime') y la firma del token ('ValidateIssuerSigningKey').
- La clave secreta ('JWT:Secret'), el emisor y la audiencia se leen desde 'appsettings.json', promoviendo una configuración segura y flexible.
- Los controladores y/o acciones que requieren autenticación están decorados con el atributo '[Authorize]'. Swagger UI también ha sido configurado para permitir la inclusión de tokens Bearer para probar los endpoints protegidos.

4.4. Personalización Visual y Experiencia de Usuario (UX) en Blazor WEB

Se ha puesto énfasis en crear una interfaz de usuario limpia, intuitiva y responsiva utilizando Blazor WebAssembly y el framework de diseño Bootstrap 5.

- **Diseño Consistente:** Se utiliza un layout principal ('MainLayout.razor') con una barra de navegación lateral ('NavMenu.razor') para una navegación consistente a través de la aplicación.
- **Componentes Reutilizables:** Se han creado componentes de formulario (ej. 'IngresosForm.razor', 'MetasAhorroForm.razor') para encapsular la lógica de entrada de datos y validación, promoviendo la reutilización de código.
- **Feedback al Usuario:** Se utiliza la librería 'SweetAlert2' para mostrar mensajes de éxito, error o confirmación de forma atractiva y no intrusiva. Los botones y operaciones que implican llamadas a la API muestran indicadores de carga (spinners) para informar al usuario que una acción está en progreso.
- **Iconografía:** Se utilizan iconos de Bootstrap Icons para mejorar la comprensión visual de las acciones y elementos de la interfaz.
- **Responsividad Básica:** Gracias a Bootstrap, la aplicación tiene un grado de responsividad para adaptarse a diferentes tamaños de pantalla, aunque se podrían realizar mejoras específicas para móviles.
- **Colores Personalizados (Limitado):** Aunque no se ha realizado una personalización exhaustiva de colores, se han utilizado las clases de Bootstrap (ej. 'bg-primary', 'text-success', 'text-danger') para dar significado visual a ciertos elementos, como los montos de ingresos y egresos,cabeceras de las tarjetas.

4.5. Registro Público de Usuarios y Autenticación

La aplicación permite el registro de nuevos usuarios y la autenticación de usuarios existentes.

- **Registro ('RegisterPage.razor'):** Un formulario permite a los nuevos usuarios ingresar su nombre, apellido, nombre de usuario, correo electrónico y contraseña (con confirmación). Los datos se envían al endpoint '/api/Auth/register'. La API valida los datos (ej. unicidad de email/username, complejidad de contraseña según las reglas de Identity) y crea el nuevo usuario. También se crea automáticamente un 'Saldo' principal para el nuevo usuario.
- **Inicio de Sesión ('LoginPage.razor'):** Los usuarios existentes pueden iniciar sesión con su correo electrónico y contraseña. Los datos se envían al endpoint '/api/Auth/login'. Si las credenciales son válidas, la API devuelve un token JWT.
- **Gestión de Sesión en Blazor ('JwtAuthenticationService.cs'):** Un servicio personalizado ('JwtAuthenticationService') que implementa 'AuthenticationStateProvider' e 'ILoginService' se encarga de:
  - Almacenar el token JWT en 'localStorage' después de un login exitoso.
  - Leer el token desde 'localStorage' al cargar la aplicación para determinar el estado de autenticación.
  - Parsear los claims del token para construir el 'ClaimsPrincipal' del usuario.
  - Notificar a Blazor los cambios en el estado de autenticación.
  - Eliminar el token de 'localStorage' durante el logout.
  - Configurar la cabecera 'Authorization' por defecto en 'HttpClient' para las llamadas a la API.
- **Protección de Rutas y Contenido:** Se utiliza el componente '<AuthorizeView>' y el atributo '[Authorize]' para restringir el acceso a páginas y mostrar contenido condicionalmente basado en el estado de autenticación y los roles del usuario.

La funcionalidad de "foto de perfil ҫconfirmación de cuenta por email"no se han implementado en la versión actual, pero la estructura de ASP.NET Core Identity facilita su adición futura.

4.6. Administración de Entidades (CRUD)

La aplicación proporciona interfaces para que los usuarios gestionen sus datos financieros:

- **Ingresos y Egresos:** Los usuarios pueden crear, ver (listar), editar y eliminar sus registros de ingresos y egresos. Cada operación está vinculada a su 'Saldo' y, por ende, a su 'UserId'.

- **Saldo:** Los usuarios pueden ver su saldo actual (calculado y manual si existe). La edición se limita a establecer un valor para el saldo manual. La creación de la entidad 'Saldo' principal es automática al registrarse el usuario.
- **Metas de Ahorro:** CRUD completo para las metas de ahorro personales, incluyendo la posibilidad de registrar contribuciones.
- **Recordatorios de Gasto:** CRUD completo para los recordatorios de gasto personales.
- **Categorías:** Actualmente, las categorías se listan y se pueden crear/editar (potencialmente restringido a un rol Admin). No se implementó la eliminación para evitar problemas de integridad referencial con transacciones existentes.
- **Historial:** Se lista el historial de transacciones. La creación es automática a partir de ingresos/egresos. Se implementó la edición y eliminación manual del historial, aunque en un sistema real esto podría estar restringido o ser solo de lectura.

Todas estas operaciones CRUD en la API están protegidas y aseguran que un usuario solo pueda operar sobre sus propios datos.

4.7. Repositorio de Código y Colaboración

El código fuente de la aplicación se encuentra alojado en un repositorio privado en GitHub. Se ha compartido el acceso de colaborador con el docente (orlapez@gmail.com) según los lineamientos. El archivo 'README.md' en la raíz del repositorio contiene los nombres completos de los integrantes del equipo y el nombre de la aplicación. La historia de commits refleja la participación de los miembros en el desarrollo.

5. Conclusiones

El desarrollo de la aplicación "Spend'nt"ha permitido aplicar de manera práctica los conceptos fundamentales de la programación de software en un entorno distribuido utilizando tecnologías modernas del ecosistema .NET. La implementación de una API RESTful con ASP.NET Core, un frontend interactivo con Blazor WebAssembly, y la gestión de datos con Entity Framework Core, junto con la seguridad proporcionada por ASP.NET Core Identity y JWT, ha constituido una experiencia de aprendizaje integral.

Incluyendo la arqui- tectura de tres capas, el modelo de dominio con el número de tablas solicitado, y la implementación de la mayoría de las funcionalidades especificadas en la matriz, como el CRUD completo para las entidades financieras principales, la autenticación de usuarios, y la presentación de datos de forma organizada.Los principales aprendizajes incluyen la importancia de un diseño de API bien definido, la gestión del estado en aplicaciones Blazor WebAssembly, los desafíos de la autenticación y autorización en sistemas distribuidos, y las mejores prácticas para la interacción con bases.

de datos mediante un ORM. La separación de responsabilidades entre el frontend, el backend y la librería compartida demostró ser crucial para la organización y mantenibilidad del código.

Como trabajos futuros o mejoras potenciales para Spend'nt, se podrían considerar:

- Implementación de la subida de fotos de perfil para usuarios.
- Flujo de confirmación de cuenta por correo electrónico.
- Generación de reportes y gráficos financieros más avanzados.
- Presupuestos y alertas de gasto.
- Notificaciones push para recordatorios de gasto.
- Internacionalización y localización.
- Pruebas unitarias y de integración más exhaustivas.
- Despliegue en un entorno de nube como Azure.

En general, el proyecto ha sido un ejercicio valioso para consolidar conocimientos y desarrollar una aplicación funcional y completa desde cero.