MiBolsillo

**Concepto General:**

Este proyecto consiste en desarrollar una aplicación web robusta y fácil de usar, construida con tecnología .NET, que permita a individuos o familias llevar un control exhaustivo de sus ingresos y gastos. El objetivo principal es ofrecer una herramienta digital para visualizar de manera clara dónde se está yendo el dinero, identificar patrones de gasto, y tomar decisiones financieras más informadas para mejorar la salud económica personal o del hogar.

**¿Para quién es este proyecto?**

Está diseñado para cualquier persona que busque mejorar su gestión financiera, desde estudiantes y jóvenes profesionales que quieren empezar a ahorrar, hasta familias que desean presupuestar y controlar sus gastos mensuales de manera más eficiente.

**Funcionalidades Clave (MVP - Producto Mínimo Viable):**

1. **Registro de Usuarios y Autenticación Segura:**
   * Los usuarios podrán crear una cuenta personal (registrarse) con un email y contraseña.
   * Accederán a la aplicación mediante un inicio de sesión seguro.
   * Cada usuario tendrá su propia información financiera protegida y separada de otros usuarios.

**Integrantes**

* Brayan Palma Aboytes – Scrum Master
* Oscar Ulises Ramírez Cruz – Desarrollador FullStack

1. **Registro de Transacciones:**
   * Los usuarios podrán añadir nuevas transacciones, especificando:
     + **Tipo de Transacción:** Si es un *ingreso* o un *gasto*.
     + **Monto:** La cantidad de dinero de la transacción.
     + **Fecha:** El día en que ocurrió la transacción.
     + **Descripción:** Un texto breve que detalle la transacción.
     + **Categoría:** Asignar la transacción a una categoría predefinida o personalizada ("Alimentos", "Transporte", "Ocio", "Vivienda", "Salario", "Inversión"). Esto es crucial para el análisis posterior.
2. **Gestión de Categorías:**
   * Los usuarios podrán ver una lista de categorías de gastos e ingresos.
   * Podrán crear nuevas categorías personalizadas, editarlas o eliminarlas (si no tienen transacciones asociadas).
3. **Visualización del Historial de Transacciones:**
   * Mostrar un listado cronológico de todas las transacciones.
   * Funcionalidades de búsqueda y filtrado por fecha, tipo de transacción (ingreso/gasto), categoría o descripción.
   * Paginación para manejar grandes volúmenes de datos.
4. **Resumen y Estadísticas Básicas:**
   * Una vista o dashboard principal que muestre:
     + El saldo actual del usuario.
     + Resumen de ingresos y gastos por mes o por un período seleccionado.
     + Desglose de gastos por categoría.

**Metodología: *Scrum***

Para el proyecto "MiBolsillo", la metodología **Scrum** es una elección excelente y muy adecuada.

**Justificación para Scrum:**

* **Enfoque Iterativo e Incremental:** Scrum trabaja por Sprints, esto nos permite construir el proyecto por incrementos funcionales, en lugar de intentar hacer todo de golpe. Para un plazo ajustado, esto es crucial: podemos tener una versión mínima funcionando rápidamente (MVP) y luego añadir más funcionalidades en Sprints posteriores.
* **Flexibilidad y Adaptación:** Permite adaptarse a cambios o nuevas ideas que surjan durante el desarrollo sin descarrilar todo el proyecto. Si algo no funciona o se descubre una mejor manera de implementarlo, Scrum facilita ajustar el rumbo.
* **Transparencia y Colaboración:** Fomenta la comunicación constante y la transparencia sobre el progreso. Con solo dos personas, esto es aún más fácil de gestionar, asegurando que ambos estén siempre alineados y conscientes del trabajo del otro.
* **Gestión de Riesgos:** Al entregar funcionalidades en ciclos cortos, los problemas y riesgos se identifican y resuelven más temprano, reduciendo la posibilidad de sorpresas al final del proyecto.
* **Entrega de Valor Continuo:** Cada Sprint finaliza con un "incremento" de software potencialmente entregable, lo que significa que el proyecto siempre estará en un estado utilizable, aunque sea con funcionalidades limitadas al principio.

**2.1. Herramientas de Apoyo**

Para complementar Scrum, las herramientas de apoyo son esenciales para la colaboración y el seguimiento.

* **GitHub:**
  + **Control de Versiones:** Es fundamental para que el equipo trabaje en el mismo código sin sobrescribirse. Permite ramificación, fusiones, seguimiento de cambios y reversión a versiones anteriores si es necesario.
  + **Colaboración:** Permite la revisión de código a través de Pull Requests.
  + **Gestión de Issues:** Puede usarse para registrar bugs, tareas y mejoras.
* **Trello:**
  + **Tablero Kanban/Scrum Visual:** Trello es excelente para visualizar el flujo de trabajo de un Sprint, con columnas como Product Backlog, Sprint Backlog, Doing, Review/Testing y Done.
  + **Gestión de Tareas:** Cada tarjeta de Trello puede representar una historia de usuario o una tarea, con la posibilidad de asignar miembros, añadir descripciones, listas de verificación y fechas de vencimiento.
  + **Comunicación:** Los comentarios en las tarjetas permiten discusiones rápidas sobre tareas específicas.

[*Tablero Trello*](https://trello.com/invite/b/683515f13740a0533d8783d6/ATTI7f89978d9875aaba17590298099c59a44339EF11/desarrollo-web-integral)

**2.2. Herramienta para Registrar el Trabajo Realizado por Fecha**

La herramienta ideal para registrar el trabajo realizado por fecha, que además se integra muy bien con las metodologías ágiles y las herramientas mencionadas, es **Trello**, complementado con **GitHub**.

* **Trello:**
  + **Registro por Fecha:** Al mover una tarjeta a la columna "Done" o "Terminado", Trello registra la fecha y hora en que se movió.
  + **Historial de Actividad:** Cada tarjeta de Trello tiene un historial de actividad detallado que muestra quién hizo qué y cuándo.
  + **Checklists dentro de tarjetas:** Dentro de cada tarjeta, al marcar un elemento como completado, Trello registra quién lo hizo y cuándo.
* **GitHub:**
  + **Commits y Pull Requests:** Cada commit (envío de cambios al repositorio) y cada fusión de Pull Request tienen una marca de tiempo y el autor, lo que proporciona un historial técnico detallado de la contribución.
  + **GitHub Issues:** Registran la fecha de creación y los comentarios, sirviendo como seguimiento adicional para tareas o bugs específicos.

Esta combinación dará una visión clara del progreso del proyecto y de la contribución individual a lo largo del tiempo, facilitando el seguimiento y la colaboración en el proyecto .NET.

**Definir Arquitectura: *Arquitectura en Capas con Patrón MVC***

La arquitectura más adecuada para el proyecto "MiBolsillo" utilizando .NET con ASP.NET Web Forms es una Arquitectura en Capas (o N-Tier), implementando el patrón Model-View-Controller (MVC) de forma conceptual en la capa de presentación.

**Justificación de la Arquitectura:**

Esta combinación proporciona una estructura robusta, modular y mantenible, fundamental para el desarrollo de aplicaciones web.

**3.1. Arquitectura en Capas**

La arquitectura en capas divide la aplicación en distintas responsabilidades lógicas y físicas. Para este proyecto, se definirán al menos tres capas principales:

* **Capa de Presentación (UI Layer):**
  + **Responsabilidad:** Es la interfaz de usuario, lo que el usuario ve y con lo que interactúa. Se encarga de mostrar la información y capturar las entradas del usuario.
  + **Implementación en .NET:** Aquí es donde ASP.NET Web Forms juega un papel central. Contendrá las páginas .aspx (el marcado HTML y controles de UI) y sus archivos .aspx.cs.
  + **Justificación:** Separa la forma en que se presentan los datos de cómo se procesan, permitiendo que diseñadores y desarrolladores de frontend trabajen con relativa independencia.
* **Capa de Lógica de Negocio (Business Logic Layer - BLL / Application Layer):**
  + **Responsabilidad:** Contiene las reglas de negocio, validaciones y la lógica central de la aplicación. Aquí se procesan los datos, se aplican las reglas para los cálculos de gastos/ingresos, y se orquesta el flujo de las operaciones.
  + **Implementación en .NET:** Clases C# que encapsulan la lógica de la aplicación. Los code-behind de las páginas .aspx.cs interactuarán con esta capa. Se pueden usar servicios o manejadores para organizar la lógica de negocio.
  + **Justificación:** Aísla la lógica de negocio de la interfaz de usuario y del acceso a datos, haciendo que sea reusable, probada de forma independiente y más fácil de mantener. Cualquier cambio en una regla de negocio solo afectará a esta capa.
* **Capa de Acceso a Datos (Data Access Layer - DAL / Infrastructure Layer):**
  + **Responsabilidad:** Gestiona la comunicación con la base de datos. Se encarga de las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y de mapear los objetos del dominio a los registros de la base de datos.
  + **Implementación en .NET:** Se puede utilizar ADO.NET directamente, o un ORM como LINQ to SQL o versiones anteriores de Entity Framework. Esta capa contendrá las clases responsables de la conexión y las consultas a la base de datos.
  + **Justificación:** Abstrae los detalles de la base de datos de las otras capas. Esto significa que, en teoría, la base de datos subyacente podría cambiarse sin afectar a las capas superiores. También centraliza la lógica de persistencia.

**3.2. Patrón Model-View-Controller (MVC) de forma conceptual en la Capa de Presentación**

Aunque ASP.NET Web Forms no implementa el patrón MVC de manera estricta como ASP.NET MVC o Core, se puede aplicar su filosofía conceptualmente dentro de la capa de presentación para mantener una buena organización:

* **Modelo (Model):**
  + **Responsabilidad:** Representa los datos y la lógica de negocio asociada a esos datos. Incluye las clases que mapean a las tablas de la base de datos (POCOs o entidades de datos) y la lógica de validación de datos.
  + **Implementación en .NET:** Clases C# que representan las transacciones, categorías, usuarios, etc. Estas clases serán utilizadas por las capas de negocio y datos.
  + **Justificación:** Separa los datos de la interfaz de usuario, promoviendo la reutilización y la claridad.
* **Vista (View):**
  + **Responsabilidad:** Es la interfaz de usuario que se muestra al usuario. Se encarga únicamente de la presentación de los datos.
  + **Implementación en .NET:** Archivos **.aspx** que contienen el marcado HTML y los controles de servidor. Estos archivos se encargan de renderizar la interfaz.
  + **Justificación:** Permite que los diseñadores se enfoquen en la interfaz sin preocuparse por la lógica de negocio.
* **"Controlador" (Code-Behind):**
  + **Responsabilidad:** En ASP.NET Web Forms, el archivo **.aspx.cs** (el code-behind) actúa como un "controlador" en el sentido de que maneja los eventos de la UI, interactúa con la Capa de Lógica de Negocio para obtener o procesar datos, y actualiza la Vista.
  + **Implementación en .NET:** Los métodos dentro de los archivos .aspx.cs que responden a eventos de los controles de UI.
  + **Justificación:** Centraliza el manejo de los eventos de usuario y la orquestación entre la UI y la lógica de negocio, ayudando a mantener la organización del código.

**3.3. Justificación Adicional de la Arquitectura Propuesta:**

* **Separación de Preocupaciones:** Cada capa y, en la medida de lo posible, cada componente dentro del code-behind tienen una responsabilidad única, lo que facilita la comprensión, el desarrollo y la depuración del código.
* **Mantenibilidad:** Los cambios en una capa o componente tienen un impacto mínimo en otros, reduciendo el riesgo de introducir errores y facilitando la evolución de la aplicación.
* **Capacidad de Prueba:** Las capas de Lógica de Negocio y Acceso a Datos, al estar desacopladas de la UI, son más fáciles de probar de forma unitaria, lo que contribuye a la calidad del software.
* **Reutilización:** La lógica de negocio y el acceso a datos pueden ser reutilizados si se necesitara construir otra interfaz.
* **Familiaridad con Web Forms:** Utilizar ASP.NET Web Forms y esta estructura de capas se alinea con las prácticas comunes para este tipo de aplicaciones en .NET Framework.

Esta arquitectura proporcionará una base sólida y bien organizada para el desarrollo del proyecto "MiBolsillo" utilizando ASP.NET Web Forms.

