SmartSaving

Sistema de Gestão de Finanças Pessoais

Daniel Silva, 90120

Licenciatura de Engenharia Informática

Tecnologias da Internet III

João Rebelo

Porto, 2025

Índice de Figuras

Índice

1. Problema

Muitas pessoas enfrentam dificuldades na gestão das suas finanças pessoais, perdendo o controlo sobre os gastos mensais, não conseguindo identificar padrões de consumo ou planear adequadamente o seu orçamento. A falta de uma ferramenta centralizada e intuitiva para registo e análise de transações financeiras resulta em decisões financeiras pouco informadas e dificuldades no cumprimento de objetivos financeiros.

2. Objetivos

2.1. Objetivo Geral

Desenvolver uma aplicação web que permita aos utilizadores gerir eficazmente as suas finanças pessoais através do registo e análise de transações financeiras.

2.2. Objetivos específicos

- **Registo de Transações:** Permitir o registo detalhado de ganhos e despesas
- **Categorização:** Organizar transações por categorias
- **Análise Financeira:** Gerar relatórios e gráficos sobre padrões de gastos
- **Gestão de Utilizadores:** Implementar sistema de autenticação com três níveis de acesso
- **Interface Responsiva:** Garantir acessibilidade em diferentes dispositivos

3. Estado da Arte

3.1. Soluções Existentes

- **Mint - Intuit:** Plataforma americana líder em gestão financeira pessoal
- **YNAB (You Need A Budget):** Focado em orçamentação e planeamento
- **PocketGuard:** Aplicação móvel para controlo de gastos

3.2. Análise Competitiva

As soluções existentes apresentam limitações como:

- Complexidade excessiva para utilizadores básicos
- Foco em mercados específicos (principalmente EUA)
- Custos elevados para funcionalidades avançadas
- Falta de personalização para contexto português

3.3. Proposta de Valor

O SmartSaving diferencia-se por:

- Simplicidade e foco nas funcionalidades essenciais
- Arquitetura moderna e escalável

4. Metodologia e Funcionalidades

4.1. Metodologia de Desenvolvimento

- **Abordagem:** Desenvolvimento iterativo com foco em MVP (Minimum Viable Product)
- **Padrão Arquitetural:** Model-View-Controller (MVC)
- **Gestão de Dados:** SQL Server com Stored Procedures

4.2. Funcionalidades Principais

4.2.1. Para Utilizadores Anónimos

- Visualização de página informativa sobre o sistema
- Acesso a demonstração limitada das funcionalidades
- Registo

4.2.2. Para Utilizadores Autores

- Autenticação no sistema
- Criação e visualização das suas próprias transações
- Edição e eliminação das suas próprias transações
- Categorização de transações
- Visualização de relatórios pessoais básicos

4.2.3. Para Utilizadores Editores

- Gestão de categorias do sistema

4.3. Entidade Principal (CRUD Completo)

Tracker será a entidade principal demonstrativa do sistema, representando o controlador financeiro mensal de cada utilizador. Esta entidade permite demonstrar eficazmente os três níveis de autorização:

4.3.1. Operações por Perfil:

Anónimo:

- Apenas visualização de informações gerais do sistema

User (Autor):

- **Create:** Criar novos trackers mensais
- **Read:** Listar e visualizar detalhes dos seus trackers
- **Update:** Atualizar os seus próprios trackers
- **Delete:** Eliminar os seus trackers

Admin (Editor):

- Gestão de utilizadores do sistema
- Gestão de categorias do sistema

4.3.2. Funcionalidades Associadas:

- Gestão de receitas (Incomes) e despesas (Expenses) por tracker
- Controlo mensal de orçamento pessoal
- Relatórios financeiros baseados nos dados do tracker

5. Estrutura do Projeto

5.1. Tecnologias Utilizadas

- **Backend:** ASP.NET Core 8.0 com C#
- **Frontend:** HTML5, CSS3, JavaScript, Razor Pages
- **Framework CSS:** Tailwind? ou CSS puro
- **Base de Dados:** SQL Server com Microsoft.Data.SqlClient
- **Gestão de Sessões:** ASP.NET Core Session State
- **Arquitetura:** MVC (Model-View-Controller)

5.2. Princípios e Boas Práticas

- **SoC (Separation of Concerns):** Clara separação entre camadas
- **DRY (Don't Repeat Yourself):** Reutilização de código
- **SOLID Principles:** Design orientado a objetos
- **Clean Code:** Código limpo e bem documentado
- **Responsive Design:** Interface adaptável a diferentes dispositivos

5.3. Abordagens Técnicas

- **Helper Classes Pattern:** Classes especializadas para operações de negócio
- **Session Management:** Gestão de estado do utilizador através de HttpContext.Session
- **Base Controller Pattern:** Controlador genérico para funcionalidades comuns
- **Stored Procedures:** Para todas as operações CRUD na base de dados
- **Configuration Management:** Configuração externa através de appsettings.json
- **JSON Serialization:** Para persistência de dados de sessão
- **GUID-based Identification:** Para identificação única de entidades

5.4. Diagrama de classes

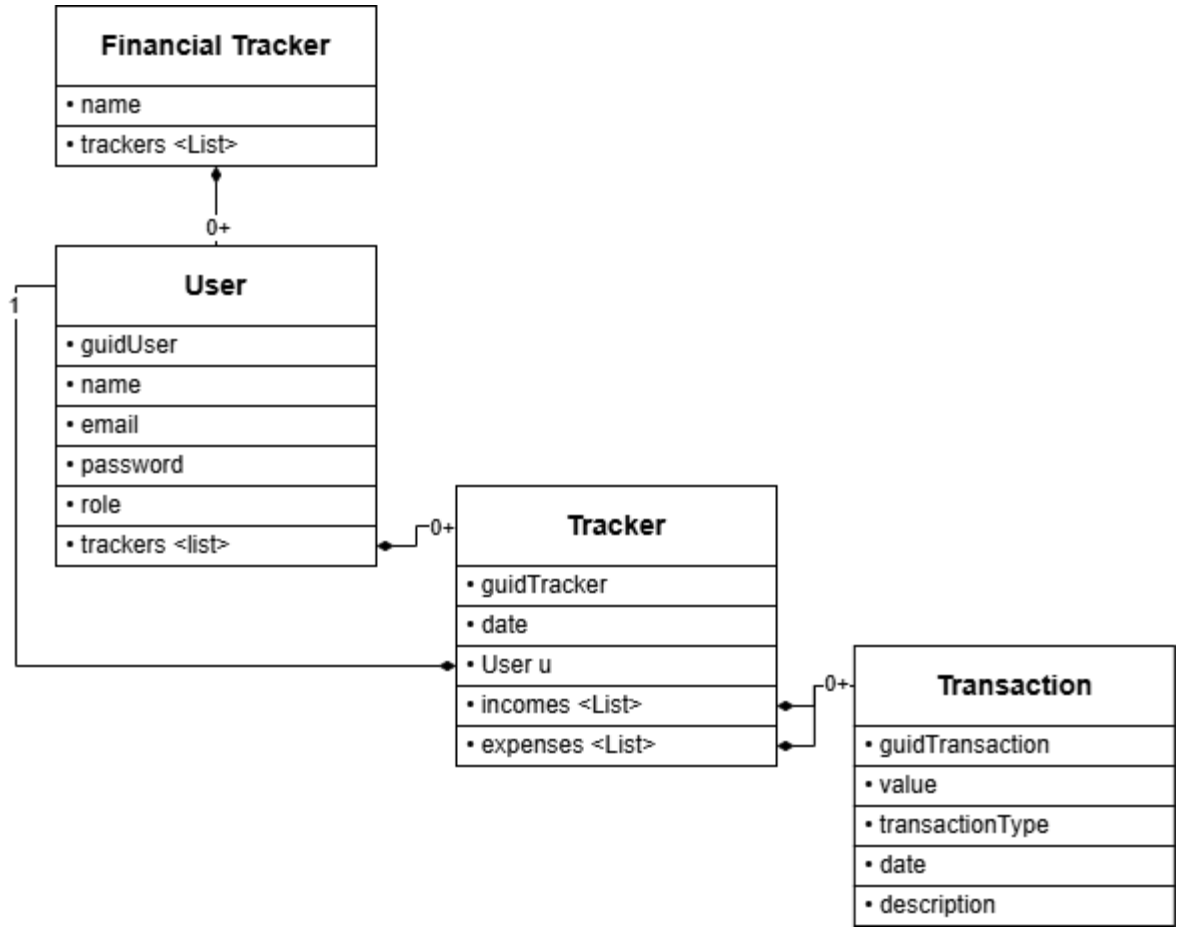

Figura 1- Diagrama de classes

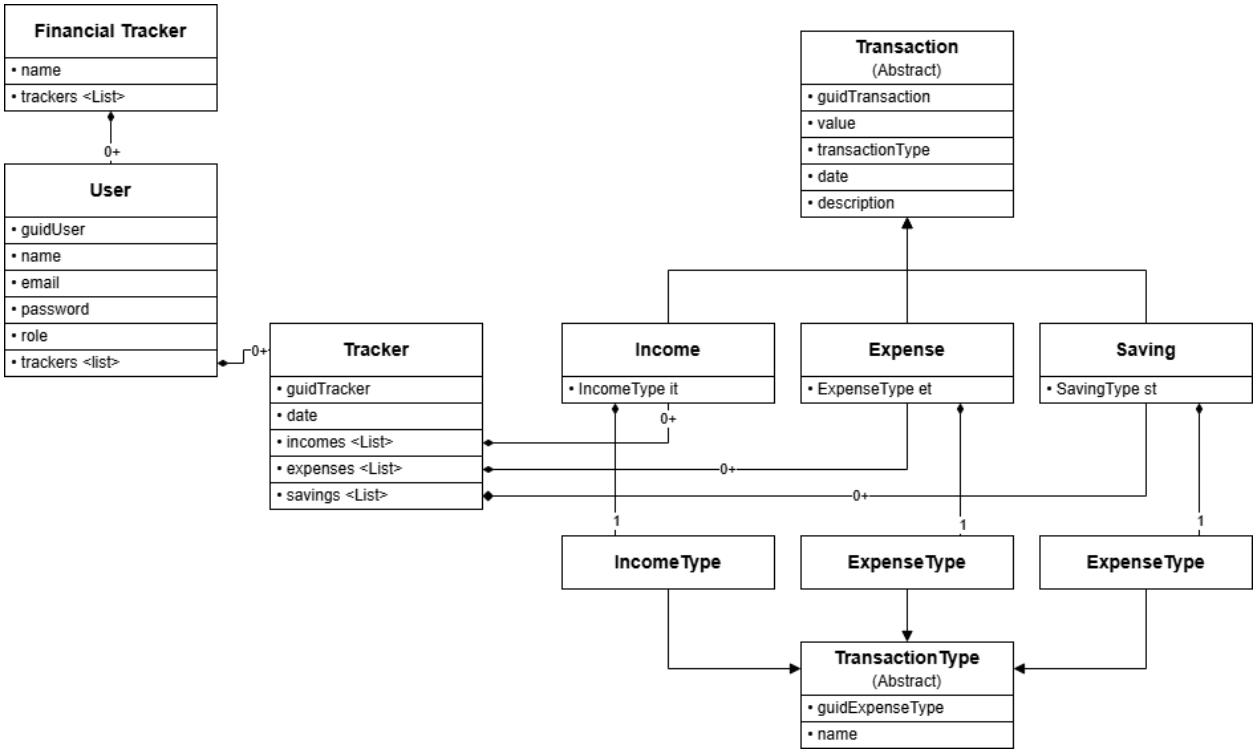

Figura 2 - Diagrama de clases (futuro)

5.5. Diagrama de use cases

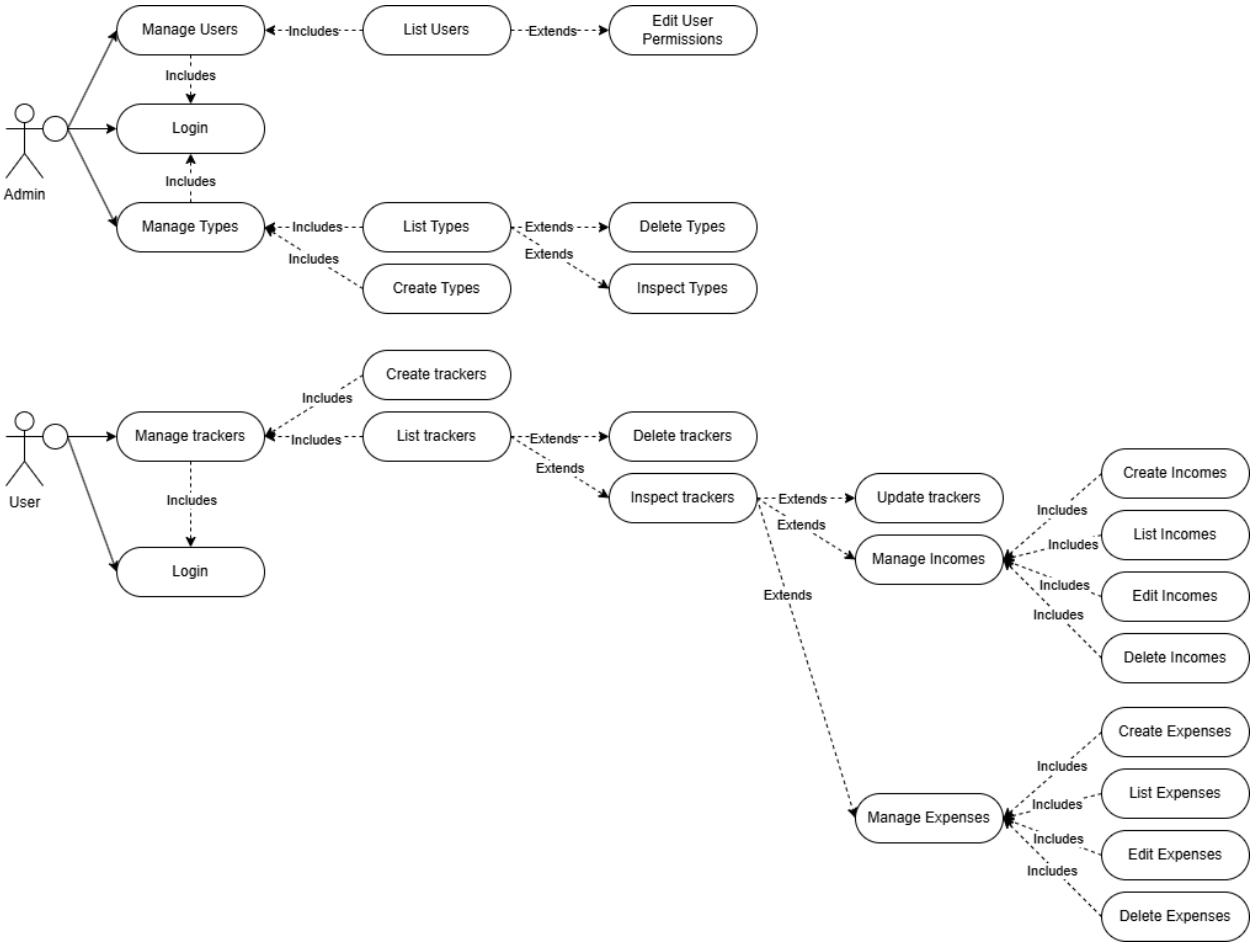

Figura 3- Diagrama de use cases

5.6. Diagrama Entidade Relacionamento

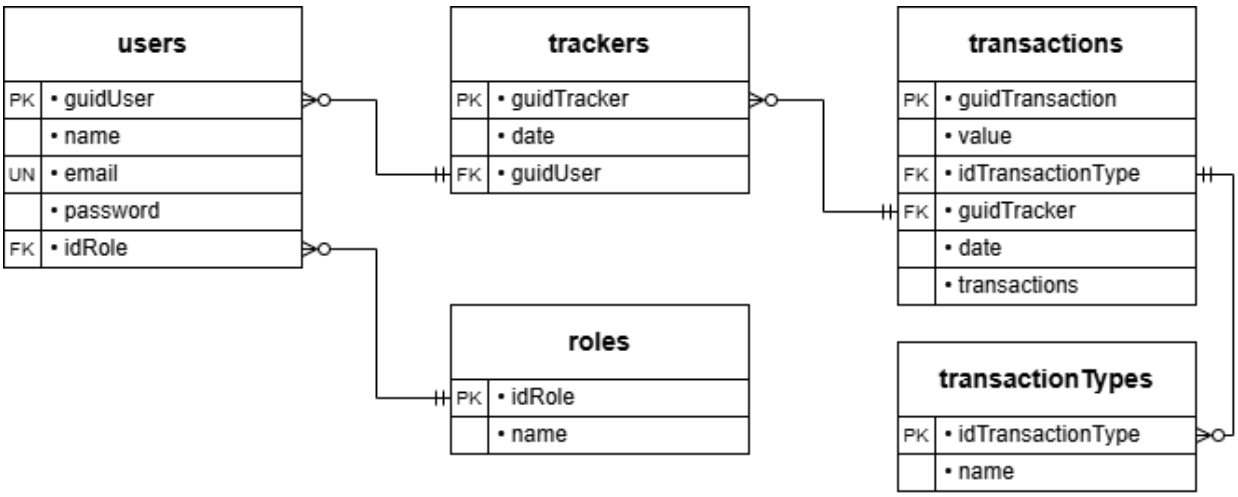

Figura 4 - Diagrama Entidade Relacionamento

6. Desenvolvimento

6.1. Arquitetura da Aplicação

A aplicação SmartSaving foi desenvolvida com base no padrão Model-View-Controller (MVC), permitindo uma clara separação de responsabilidades e uma maior escalabilidade do sistema. Esta abordagem estruturada divide o código em três grandes componentes: a camada de dados (Model), a camada de controlo (Controller) e a camada de apresentação (View).

Na camada Model, encontram-se definidas as principais entidades da aplicação, tais como o **User**, que representa os utilizadores com diferentes níveis de acesso (User, Admin e Owner), o **Tracker**, que centraliza os controladores financeiros mensais, a **Transaction**, responsável pela gestão individual de receitas e despesas, e a entidade **Configuration**, que armazena as definições gerais da aplicação.

A camada de acesso a dados, designada por Helper, abstrai as operações sobre a base de dados através de classes especializadas como **UserHelper**, **TrackerHelper** e **TransactionHelper**. Estas classes são responsáveis pelas operações CRUD e pelos cálculos associados, como o balanço financeiro. A **BaseHelper** fornece a ligação base à base de dados, centralizando a string de conexão.

A camada Controller é composta por vários controladores, como o **AuthController**, que gere a autenticação e o registo de utilizadores; o **HomeController**, que apresenta o dashboard principal com estatísticas; e o **TrackerController**, que permite a gestão completa de trackers e das suas transações associadas. Um controlador base (**GenericBaseController**) garante funcionalidades comuns de autenticação e autorização.

A interface da aplicação foi construída com Tailwind CSS, focando-se numa experiência responsiva e intuitiva. Estão incluídas vistas para autenticação (login, registo e perfil), bem como um dashboard interativo, formulários de gestão de transações e páginas dedicadas aos trackers financeiros.

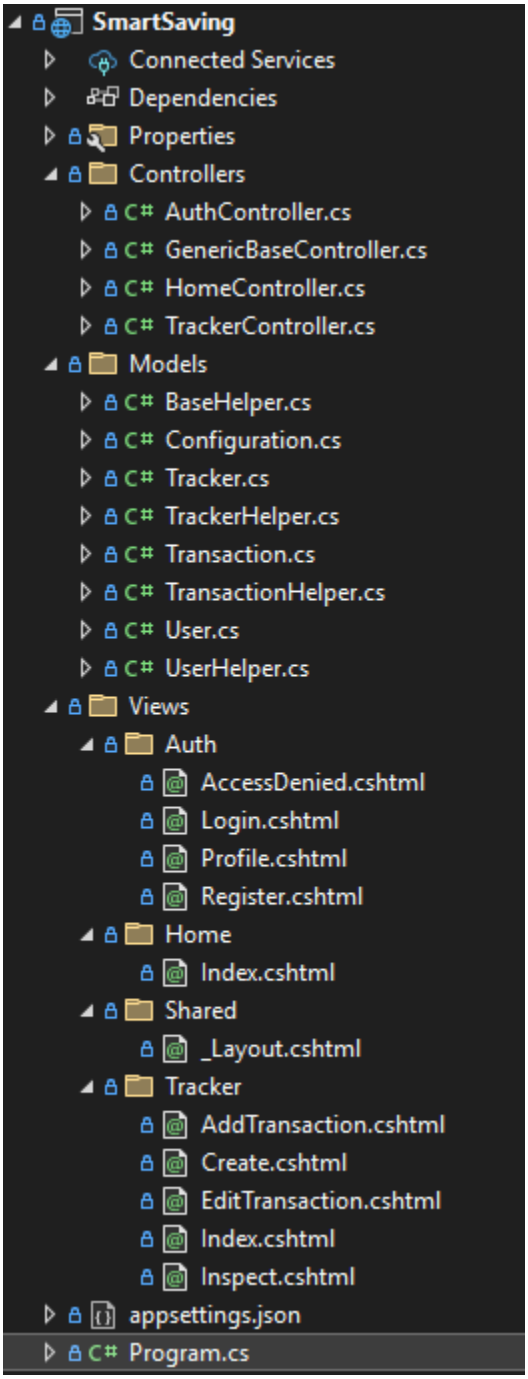

Figura 5- Sistema de Ficheiros

6.2. Funcionalidades Principais

O sistema de autenticação implementado garante a segurança dos dados através da encriptação de passwords com SHA256, mantendo sessões seguras e um modelo hierárquico de permissões. É ainda possível aceder à aplicação de forma anónima através de um utilizador convidado, gerido automaticamente.

A gestão de trackers financeiros permite criar controladores mensais, visualizar estatísticas em tempo real, eliminar dados com confirmação prévia e aceder a análises detalhadas das transações realizadas. Estas transações são automaticamente categorizadas como receitas ou despesas, com os balanços financeiros a serem calculados em tempo real com base no tipo e valor de cada transação. O sistema valida todos os dados introduzidos, garantindo a consistência da informação.

```
3 references
private string hashPassword(string password) {
    if (string.IsNullOrWhiteSpace(password)) {
        throw new ArgumentException("Password cannot be empty");
    }

    using (SHA256 sha256Hash = SHA256.Create()) {
        byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(password));
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < bytes.Length; i++) {
            builder.Append(bytes[i].ToString("x2"));
        }
        return builder.ToString();
    }
}
```

Figura 6 - Encriptação de Password

6.3. Aspetos Técnicos

A aplicação utiliza uma base de dados SQL Server, recorrendo a stored procedures otimizadas para a execução eficiente de todas as operações. Cada entidade é identificada de forma única através de GUIDs, e são garantidas relações de integridade entre as diferentes tabelas. As procedures implementadas, como QTracker_GetCountByUser ou QTransaction_GetTotalValueByTrackerAndType, asseguram a fiabilidade dos cálculos e o desempenho da aplicação.

Em termos de gestão de estado, a serialização e de-serialização de objetos de utilizador na sessão são feitas com recurso a JSON, permitindo manter o contexto de autenticação ao longo da navegação. O sistema inclui ainda validações rigorosas em todos os formulários, controlo de acesso baseado em roles, proteção contra CSRF e validação da propriedade dos recursos acedidos.

A interface desenvolvida com Tailwind CSS segue o princípio desktop-first, utilizando componentes reutilizáveis e oferecendo feedback visual imediato ao utilizador. A navegação foi pensada para ser fluída e intuitiva, independentemente do dispositivo utilizado.

```
6 references
public User? get(string guidUserToFind) {
    if (string.IsNullOrWhiteSpace(guidUserToFind)) {
        return null;
    }

    DataTable dt = new DataTable();
    SqlDataAdapter dataAdapter = new SqlDataAdapter();
    SqlCommand command = new SqlCommand();
    SqlConnection connection = new SqlConnection(Conector);

    command.CommandType = CommandType.StoredProcedure;
    command.Connection = connection;
    command.CommandText = "QUser_Get";
    command.Parameters.AddWithValue("@Guid", guidUserToFind);

    dataAdapter.SelectCommand = command;
    dataAdapter.Fill(dt);

    if (dt.Rows.Count == 1) {
        DataRow row = dt.Rows[0];
        User user = new User(row["guidUser"].ToString());
        user.Name = row["name"].ToString();
        user.Email = row["email"].ToString();
        user.Password = row["password"].ToString();
        user.Role = (User.Roles)Convert.ToByte(row["idRole"]);
        return user;
    }

    return null;
}
```

Figura 7 - Exemplo de Stored Procedure

6.4. Padrões e boas práticas

A aplicação seguiu rigorosamente os princípios de Separation of Concerns (SoC), com separação clara entre a lógica de negócio, o acesso a dados e a apresentação. Foram aplicados os princípios DRY (Don't Repeat Yourself) através da reutilização de código em helpers e componentes de interface, bem como da existência de uma classe base para os controladores.

Adicionalmente, o código foi desenvolvido segundo as boas práticas de Clean Code, com nomes descritivos, métodos com responsabilidade única e comentários apenas onde indispensáveis. Em termos de tratamento de erros, foi adotada uma abordagem robusta, com blocos try-catch-finally que garantem a fiabilidade das operações e a libertação dos recursos utilizados.

```
1 reference
public Boolean create(User userToStore) {
    if (userToStore == null) {
        return false;
    }

    Boolean result = false;
    SqlCommand command = new SqlCommand();
    SqlConnection connection = new SqlConnection(Conector);

    command.CommandType = CommandType.StoredProcedure;
    command.Connection = connection;
    command.CommandText = "QUser_Create";

    command.Parameters.AddWithValue("@Name", userToStore.Name);
    command.Parameters.AddWithValue("@Email", userToStore.Email);
    command.Parameters.AddWithValue("@Password", hashPassword(userToStore.Password));
    command.Parameters.AddWithValue("@IdRole", (byte)userToStore.Role);

    try {
        connection.Open();
        command.ExecuteNonQuery();
        result = true;
    }
    catch (Exception ex) {
        throw new Exception("Error creating user: " + ex.Message);
    }
    finally {
        connection.Close();
        connection.Dispose();
    }

    return result;
}
```

Figura 8 - Exemplo Try-Catch-Finally

7. Resultados Obtidos

O SmartSaving representa uma evolução significativa face ao projeto anterior, o DemoGifts. A nível de segurança, substituiu-se o armazenamento de passwords em texto simples por hashes encriptados. A validação dos dados foi tornada mais robusta e a interface foi redesenhada com uma abordagem moderna utilizando Tailwind CSS. Além disso, as funcionalidades evoluíram de um CRUD básico para um sistema completo de gestão financeira, demonstrando um claro amadurecimento técnico e uma aplicação prática dos conhecimentos adquiridos ao longo do desenvolvimento.

8. Conclusão

O desenvolvimento do SmartSaving proporcionou uma experiência prática valiosa na aplicação de tecnologias web modernas, demonstrando a evolução desde conceitos básicos até à implementação de uma solução completa e funcional.

O projeto evidencia não apenas competências técnicas em ASP.NET Core, mas também capacidades de design de sistema, análise de requisitos e implementação de boas práticas de desenvolvimento. A progressão desde o DemoGifts até ao SmartSaving ilustra claramente a curva de aprendizagem e a maturidade técnica adquirida.

Esta experiência consolida conhecimentos fundamentais em desenvolvimento web e prepara o terreno para projetos futuros de maior complexidade, demonstrando que a combinação de tecnologia moderna, design centrado no utilizador e implementação robusta pode resultar em soluções digitais verdadeiramente úteis e impactantes.

O SmartSaving não é apenas um projeto académico, mas sim uma demonstração tangível de como a tecnologia pode ser aplicada para resolver problemas reais, melhorando a qualidade de vida dos utilizadores através de ferramentas digitais bem concebidas e implementadas.

Limitações Atuais

- Sistema de categorias fixo (Income/Expense)
- Relatórios limitados a estatísticas básicas
- Ausência de funcionalidades de exportação de dados
- Integração limitada com sistemas bancários

Propostas para Desenvolvimento Futuro

- **Categorias Personalizáveis:** Permitir aos utilizadores criar categorias customizadas
- **Relatórios Avançados:** Implementar gráficos interativos e análises temporais
- **API REST:** Desenvolver API para integração com aplicações móveis
- **Machine Learning:** Análise preditiva de padrões de gastos
- **Integração Bancária:** Importação automática de transações
- **Funcionalidades Colaborativas:** Gestão familiar de orçamentos

Referências

Feasible Creative. (2024, outubro). *YNAB Budget App: What I Wish I Knew Before Starting* [Video]. YouTube. https://www.youtube.com/watch?v=mqB_Vu2_le0

Brittany Flammer. (2025, janeiro). PocketGuard Review // Updated for 2025 [Video]. YouTube. https://www.youtube.com/watch?v=Ar8GVKVUkb4

Ryan McGregor. (2022). *Mint Budgeting App Review (UPDATED Features)* [Video]. YouTube. https://www.youtube.com/watch?v=rQ_5v3BUBqQ