Desenvolvimento Full-Stack Avançado com ASP.NET Core

Proposta de Projeto

1. Título do Projeto:

Plataforma de Controle Financeiro Pessoal

2. Objetivo:

- Desenvolver uma aplicação full-stack que permita aos usuários gerenciar suas finanças pessoais. A plataforma deve oferecer funcionalidades como criação de categorias financeiras, registro de transações (entradas e saídas), relatórios dinâmicos e gráficos interativos para análise de dados financeiros.

3. Descrição Geral e Requisitos Funcionais:

Descrição Geral

A Plataforma de Controle Financeiro Pessoal é uma aplicação web full-stack projetada para ajudar usuários a gerenciar suas finanças de forma eficiente e organizada. A solução deve oferecer um painel integrado para registro de transações financeiras, relatórios interativos, e ferramentas de planejamento financeiro, garantindo segurança, usabilidade e escalabilidade.

O sistema será dividido em três camadas principais:

1. Frontend (Interface do Usuário): Desenvolvido em Angular ou Blazor para criar uma SPA (Single Page Application) responsiva e interativa.
2. Backend (API RESTful): Desenvolvido com ASP.NET Core WebAPI, responsável pelo processamento de dados e lógica de negócios.
3. Banco de Dados: SQL Server com EF Core para persistência e gerenciamento de dados,
    - **Deve haver suporte para SQLite** para validação sem dependência de infra.

Requisitos Funcionais

1. Cadastro e Autenticação de Usuários

- **Cadastro de Usuários**:
    - Permitir registro de novos usuários com dados como nome, e-mail, senha.
    - Validação de campos obrigatórios com feedback claro ao usuário.
- **Autenticação**:
    - Implementação de login seguro utilizando ASP.NET Core Identity e JWT.
    - Sessões protegidas com expiração configurável do token JWT.
    - O user do Identity deve co-existir com a entidade Usuário, ambos compartilham o mesmo ID e devem ser criados no mesmo processo.

2. Gerenciamento de Transações

- **Registro de Transações**:
    - Adicionar transações com dados como valor, descrição, categoria, tipo (entrada/saída)
    - Permitir edição e exclusão de transações.
- **Filtro e Busca de Transações**:
    - Implementar filtros por data, categoria e tipo.

3. Gestão de Categorias

- **Categorias**:
    - CRUD completo para categorias de transações financeiras.
    - Categorias padrão criadas automaticamente (ex.: Alimentação, Transporte).
    - Proibir a exclusão de categorias que estão sendo utilizadas em transações já lançadas.

4. Relatórios e Dashboards

- **Relatórios Interativos**:
    - Geração de relatórios financeiros detalhados por categoria e período.

    - Exibição de gráficos dinâmicos (ex.: pizza, barras) usando bibliotecas como Chart.js ou similares
    - DESEJÁVEL: Relatórios customizáveis com opções de filtros e exportação (PDF, Excel).
- **Painel Resumido**:
    - Visão geral das finanças: saldo total, total de receitas e despesas por período.

5. Planejamento e Orçamentos

- **Definição de Orçamentos**:
    - Configurar limites de gastos por categoria e alertar quando ultrapassados.
- **Alertas Automáticos**:
    - Notificar o usuário sobre orçamento ultrapassado ou saldo baixo.

6. Segurança e Privacidade

- Implementar autenticação e autorização robustas usando JWT e roles do Identity.
- Proteção contra ataques comuns, como SQL Injection e Cross-Site Scripting (XSS).
- Configuração de HTTPS obrigatório para todas as comunicações.
- Os dados de um usuário nunca devem ser acessíveis por outro.

7. Usabilidade

- Interface intuitiva e responsiva (desejavel suporte a dispositivos móveis).
- Feedback claro em caso de erros ou validações incorretas.
- Experiência aprimorada com design moderno e navegação fluida.

Regras de Negócio

1. **Restrições de Transações**:
    - Nenhuma transação pode ser registrada com valores negativos.
2. **Limitações de Orçamentos**:
    - Não é permitido definir um orçamento negativo ou exceder os recursos disponíveis.
3. **Exclusão Condicional**:
    - Transações associadas a categorias excluídas devem ser previamente realocadas.
4. **Categorias Padrão**:
    - Categorias padrão do sistema são imutáveis (não podem ser alteradas ou excluídas).

4. Requisitos Técnicos

- **Linguagem de Programação:** C#
- **Backend**:
    - ASP.NET Core WebAPI para lógica de negócios.
    - Acesso a dados com Entity Framework Core (EF Core), SQL Server e SQLite (o uso de SQLite deve estar sempre configurado com o Seed para que qualquer participante do projeto possa executar sem a infra do BD).
    - Autenticação e autorização via Identity e JWT.
- **Frontend**:
    - Angular ou Blazor para a interface do usuário com SPA (Single Page Application).
    - Gráficos interativos utilizando bibliotecas como Chart.js, D3.js ou similares.
- **Documentação**:
    - Swagger para documentação da API.

- **Versionamento:**
    - Github para controle de versão, com o código sendo hospedado em um repositório publico e dentro dos padrões especificados em: https://github.com/desenvolvedor-io/template-repositorio-mba

5. Critérios de Sucesso:

- **Funcionalidade Completa:**
    - Implementação de todas as funcionalidades descritas no escopo.
- **Qualidade do Código:**
    - Código bem estruturado, documentado e de fácil entendimento.
- **Segurança:**
    - Autenticação e autorização robustas, proteção contra SQL Injection e outros riscos.
- **Interface:**
    - Interface amigável, intuitiva e funcional, priorizando a experiência do usuário.
- **Apresentação:**
    - Apresentação clara e objetiva, destacando as principais funcionalidades e desafios enfrentados.
- **Configuração (OBRIGATÓRIO):**
    - O projeto deve rodar com a menor configuração de infra possível, para isso utilize a prática ensinada no vídeo a seguir: https://desenvolvedor.io/plus/criando-e-populando-automaticamente-qualquer-banco-de-dados

6. Prazos e Grupos:

- **Início do Projeto:** 16/12/2024
- **Fechamento dos grupos:** 23/11/2025
- **Primeira entrega (avaliação):** 03/02/2025
- **Segunda entrega (final):** 17/02/2025
- **Apresentação:** 24/02/2025 à 28/02/2025

Formação de Grupos e Regras de Trabalho

Para o desenvolvimento deste projeto, deverão ser formados **apenas cinco turmas**. Cada grupo será responsável por definir seus integrantes e registrar formalmente a composição do grupo. Assim que os grupos forem definidos, será enviado um documento para preenchimento, onde deverão constar os **nomes completos de todos os participantes** e o **nome do grupo**.

Responsabilidades do Grupo

- **Entrega do Projeto**: O projeto será entregue por grupo, em um **repositório público no GitHub**, devidamente organizado e seguindo os padrões exigidos.
- **Responsabilidade Compartilhada**: Todos os integrantes do grupo são igualmente responsáveis por **todas as funcionalidades** do projeto. A divisão de tarefas entre os membros é permitida e incentivada, mas o **resultado final do projeto refletirá na nota individual de cada integrante**.
    - Isso significa que, independentemente de um membro estar focado em uma funcionalidade específica, o grupo como um todo será avaliado pelo sucesso do projeto.
- **Qualidade e Cooperação**: O trabalho em equipe e a colaboração são elementos cruciais para o sucesso do projeto. Todos os membros devem estar alinhados e contribuir de forma ativa.

Apoio e Suporte

- **Instrutor Disponível**: O instrutor estará à disposição durante todo o período do projeto para esclarecer dúvidas e orientar decisões de arquitetura.
- **Sessões de Suporte**: Caso necessário, o grupo poderá solicitar uma reunião por chamada de vídeo com o instrutor, para discutir detalhes do projeto e receber orientações específicas.

O objetivo é garantir que todos os alunos tenham suporte adequado para superar desafios técnicos ou de planejamento, promovendo aprendizado colaborativo e resultados de alta qualidade.

7. Entrega:

- **Repositório no GitHub:**
    - O código deve ser versionado e entregue através de um repositório público no Github.
- **Documentação:**
    - O README.md deve seguir as diretivas e padrões informados na documentação do projeto referência.
    - Incluir um arquivo FEEDBACK.md no repositório onde os feedbacks serão consolidados, o instrutor fará um PR no repositório atualizando este arquivo.

8. Matriz de avaliação:

- Os projetos serão avaliados e receberão uma nota de 0 até 10 considerando os critérios a seguir:

Critério	**Peso**	**Comentários**
Funcionalidade	30%	Avalie se o projeto atende a todos os requisitos funcionais definidos.
Qualidade do Código	20%	Considere clareza, organização, uso de padrões de codificação.
Eficiência	10%	Avalie o desempenho e a eficiência das soluções implementadas.
Inovação	10%	Considere a criatividade e inovação na solução proposta.
Documentação	10%	Verifique a qualidade e completude da documentação, incluindo README.md.
Apresentação	10%	Avalie a clareza, organização e impacto da apresentação ao vivo.
Resolução de Feedbacks	10%	Considere como o aluno ou grupo abordou os feedbacks da revisão de código.

9. Casos de Uso

Caso de Uso 1: Cadastro de Usuário

Ator Principal: Usuário
Descrição: O sistema permite que novos usuários se cadastrem, criando suas contas pessoais.
Fluxo Principal:

1. O usuário acessa a página de cadastro.
2. Preenche os campos obrigatórios: nome, e-mail e senha.
3. O sistema valida os dados e cria a conta do usuário.

- E-mail único para cada conta.
- Senhas devem atender critérios de segurança (ex.: mínimo 8 caracteres, letras maiúsculas, números e símbolos).

Caso de Uso 2: Autenticação de Usuário

Ator Principal: Usuário
Descrição: O sistema autentica o usuário para permitir o acesso às funcionalidades.
Fluxo Principal:

1. O usuário acessa a página de login.
2. Insere o e-mail e a senha.
3. O SPA faz via API a validação das credenciais e retorna um token JWT que deve ser mantido em sessão do browser.

- O usuário é redirecionado ao painel de controle financeiro.
- Se as credenciais forem inválidas, o sistema exibe uma mensagem de erro.

Caso de Uso 3: Registro de Transações

Ator Principal: Usuário autenticado
Descrição: Permite ao usuário registrar entradas e saídas financeiras.
Fluxo Principal:

1. O usuário acessa o painel de transações.
2. Seleciona “Adicionar Transação”.
3. Insere os dados: tipo (entrada/saída), categoria, valor, data e descrição.
4. O sistema valida os dados e salva a transação no banco de dados.
   Regras de Negócio:

- O valor da transação não pode ser negativo.
- Categorias precisam estar cadastradas antes de associar uma transação.

Caso de Uso 4: Gerenciamento de Categorias

Ator Principal: Usuário autenticado
Descrição: O sistema permite criar, editar e excluir categorias financeiras.
Fluxo Principal:

1. O usuário acessa a seção de categorias.
2. Clica em “Adicionar Categoria”.
3. Insere o nome e a descrição da categoria.
4. O sistema salva os dados e exibe a categoria na lista.
   Fluxos Alternativos:

- O usuário pode editar ou excluir categorias existentes.
  Regras de Negócio:

- Categorias padrão (ex.: alimentação, transporte) não podem ser excluídas.

Caso de Uso 5: Relatórios Financeiros

Ator Principal: Usuário autenticado
Descrição: O sistema gera relatórios interativos com base nas transações do usuário.
Fluxo Principal:

1. O usuário acessa a página de relatórios.
2. Seleciona o tipo de relatório desejado (ex.: por categoria, período).
3. O sistema gera gráficos e tabelas com os dados financeiros.
4. DESJÁVEL: O usuário pode exportar o relatório em formatos como PDF ou Excel.
   Regras de Negócio:

- Apenas transações do usuário logado podem ser incluídas nos relatórios.

Caso de Uso 6: Configuração de Orçamentos

Ator Principal: Usuário autenticado
Descrição: Permite definir limites de gastos por categoria ou geral.
Fluxo Principal:

1. O usuário acessa a seção de orçamentos.
2. Define um limite mensal por categoria ou geral.
3. O sistema salva as configurações e alerta caso o orçamento seja excedido.
   Regras de Negócio:

- Limites não podem ser negativos.