DO PLANEJAMENTO À IMPLANTAÇÃO: O GERENCIAMENTO DO DESENVOLVIMENTO DE UM APLICATIVO DE COMPARTILHAMENTO DE DESPESAS

Adair Locks Grassi[1], Ana Claudia Garcia Barbosa[2]

Resumo: Este trabalho apresenta o desenvolvimento de um sistema para controle e compartilhamento de despesas em grupos, visando auxiliar no planejamento financeiro coletivo. A aplicação, construída com tecnologias como Next.js, C# com .NET 8.0 e PostgreSQL, oferece funcionalidades como registro de gastos compartilhados, categorização de despesas, cálculo dinâmico de recorrências e visualização integrada do orçamento. A solução diferencia-se de ferramentas existentes ao focar especificamente na gestão colaborativa, abordando uma lacuna no mercado de aplicativos financeiros. A metodologia incluiu análise de requisitos, modelagem de banco de dados com otimização para recorrências, implementação de arquitetura em camadas no *backend* e desenvolvimento de interface *mobile-first*. Resultados demonstram que o sistema atende aos objetivos propostos, combinando segurança (via JWT), desempenho (com cálculos sob demanda) e usabilidade (PWA responsiva). O projeto reforça a importância da engenharia de software aplicada a problemas sociais, evidenciando o potencial da computação como ferramenta de educação financeira. Como contribuição, oferece uma base para futuras expansões, como integração com serviços bancários e análises preditivas.

Palavras-chave: controle financeiro; despesas compartilhadas; aplicação web; educação financeira; desenvolvimento de software.

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), adairlocksgrassi@gmail.com
[2]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), agb@unesc.net

ABSTRACT: This paper presents the development of a group expense-sharing system designed to assist in collective financial planning. The application, built with technologies such as Next.js, C#.NET 8.0, and PostgreSQL, offers features like shared expense tracking, expense categorization, dynamic recurrence calculations, and integrated budget visualization. The solution stands out from existing tools by specifically focusing on collaborative management, addressing a gap in the financial app market. The methodology included requirements analysis, database modeling optimized for recurrences, layered backend architecture implementation, and mobile-first interface development. Results demonstrate that the system meets the proposed objectives, combining security (via JWT), performance (with on-demand calculations), and usability (responsive PWA). The project highlights the importance of software engineering applied to social problems, showcasing computing's potential as a financial education tool. As a contribution, it provides a foundation for future expansions, such as banking service integrations and predictive analytics.

Keywords: financial control; shared expenses; web application; financial education; software development.

1 INTRODUÇÃO

A educação financeira é um processo essencial para a gestão eficiente de recursos e para o desenvolvimento econômico das famílias, adquirindo visão crítica sobre o uso do dinheiro (Cordeiro; Costa; Silva, 2018). Apesar de sua relevância, no cenário brasileiro, observa-se um desafio significativo, pois, de acordo com uma pesquisa realizada pela Confederação Nacional do Comércio (CNC), o índice de endividamento brasileiro aumentou em 9,7% entre 2010 e 2020, sendo que essa situação poderia ter sido minimizada com uma educação financeira mais eficiente (Melo; Pedro, 2023). Outrossim, como escrito por (Johri et al., 2023): o controle insuficiente dos gastos pode levar a saídas irregulares de caixa, prejudicando a poupança e os objetivos financeiros. Nesse contexto, a área da computação desempenha papel importante, oferecendo ferramentas e aplicações que podem contribuir para a resolução desses problemas financeiros.

A principal problemática reside no fato de que as ferramentas financeiras que estão atualmente disponíveis no mercado focam predominantemente no controle de despesas individuais, negligenciando a organização financeira em grupo. Essa modalidade de controle de despesas

é de relevante importância, pois conforme aponta Coella et al. (2014), o controle de orçamento doméstico requer uma abordagem coletiva, pois o alinhamento entre o planejamento financeiro familiar e individual é essencial para garantir o equilíbrio das finanças. Sendo assim, essa carência de ferramentas adequadas para o planejamento financeiro coletivo dificulta a gestão integrada do orçamento. Além disso, de acordo com Tamizhselvi, Anbu e Radhakrishnan (2022), ao identificar a categoria com maior volume de gastos, torna-se mais fácil economizar, pois é possível reduzir despesas específicas.

Do ponto de vista computacional-tecnológico, existe a necessidade de desenvolver uma aplicação que vá além do registro compartilhado de despesas, incorporando funcionalidades para a análise de despesas recorrentes. A escolha deste tema se justifica pela sua relevância tanto no campo da educação financeira quanto no desenvolvimento de soluções computacionais aplicadas. Em um país com altos índices de endividamento e baixa alfabetização financeira, o desenvolvimento de ferramentas que auxiliem no planejamento e controle das finanças é de grande importância. Além disso, o presente trabalho consolida conhecimentos técnicos e metodológicos adquiridos durante o curso, aplicando-os em uma solução real.

O objetivo geral desse trabalho é desenvolver uma aplicação para controle de despesas em grupo que permita o acompanhamento e compartilhamento de gastos, oferecendo visualização de despesas recorrentes e auxiliando no planejamento financeiro coletivo.

Os objetivos específicos deste trabalho consistem em aplicar metodologias de desenvolvimento de projetos para organizar e estruturar o progresso da aplicação, utilizando as etapas necessárias para o desenvolvimento de um software, além de desenvolver uma aplicação para controle de despesas em grupo, permitindo o acompanhamento e compartilhamento de gastos de forma colaborativa. Também busca-se publicar uma primeira versão oficial da aplicação que possa ser utilizada para qualquer dispositivo móvel ou navegador web. Por fim, deseja-se garantir a construção de uma arquitetura de software segura, fundamentada em boas práticas de engenharia, para assegurar a manutenibilidade e a escalabilidade do sistema a longo prazo.

A revisão de literatura apresenta trabalhos correlatos como o "Aplicativo para controle financeiro de estudantes universitários"(Andrade et al., 2024), "Gestão Financeira Familiar: aplicativo compartilhado para planejamento, monitoramento e controle do orçamento"(SANTOS; CRUZ et al.,

2023) e o "Smart Expense Tracker Application Using Naive Bayes"(Manekar et al.,). O primeiro foca no compartilhamento de despesas, mas para um grupo específico, que são os estudantes. O segundo tem uma abordagem apenas para o orçamento familiar, utilizando receitas e despesas e possibilitando ser participante de apenas um grupo. Já o terceiro utiliza *machine learning*, porém para a gestão de despesas individuais.

Para o desenvolvimento da aplicação, foram utilizados diversos conceitos, softwares, frameworks e bibliotecas. Para a definição do escopo total da aplicação, foram utilizadas técnicas de análise de requisitos, além da criação de um diagrama de entidade e relacionamento. Para a organização e gestão das tarefas, foi utilizada a metodologia ágil Kanban junto da ferramenta Trello. As ferramentas tecnológicas escolhidas incluem o banco de dados PostgreSQL de forma *serverless*, por meio da plataforma Neon, a linguagem C# com o Entity Framework para o *backend* e o Next.js como um *framework* para desenvolvimento de uma *Progressive Web Application* (PWA) no *frontend*.

A escolha do PostgreSQL deve-se à sua popularidade e confiabilidade, já o C# e o Entity Framework combinados, oferecem uma camada de acesso a dados eficiente e portável na plataforma .NET. Já o Next.js, um *framework* construído sobre React, foi selecionado por suas vantagens como renderização híbrida (SSR e SSG), roteamento otimizado, suporte nativo a TypeScript e facilidade na criação de PWAs. Essas características proporcionam melhor desempenho, SEO e uma experiência de usuário semelhante a um aplicativo nativo em qualquer plataforma *mobile*.

A aplicação permite aos usuários criar e participar de múltiplos grupos, onde podem registrar, visualizar e gerenciar despesas compartilhadas de forma organizada. Com um sistema de autenticação segura via Json Web Token (JWT), os usuários podem criar grupos, convidar outros membros por meio de links de convite e categorizar despesas dentro de cada grupo. Ademais, possui funcionalidades como cadastro de despesas recorrentes de forma personalizada por periodicidade, cálculos sob demanda para evitar armazenamento redundante e uma interface que inclui e prioriza a visualização de gastos mensais.

2 MATERIAIS E MÉTODOS

O desenvolvimento deste projeto tem como base tecnológica a criação de um software para o gerenciamento e compartilhamento de despesas, voltado para o auxílio da educação financeira, principalmente fami-

liar. A aplicação permite a criação de grupos de usuários, onde membros podem organizar, analisar e gerenciar de forma integrada.

Além de possuir categorização de despesas por grupo, criação de links de convites para outros usuários, participação de diversos grupos, uma parte essencial da aplicação é a possibilidade de criação de despesas recorrentes, como custos mensais fixos, com opções de personalização (diária, semanal, quinzenal ou intervalos customizados). Para a obtenção de uma boa performance na criação da recorrência das despesas, foram utilizados cálculos sob demanda, para evitar armazenamento redundante.

Para a realização da pesquisa, foram abordados diversos tópicos fundamentais relacionados ao desenvolvimento da aplicação. No que diz respeito aos conceitos de engenharia de software, para a definição do escopo do projeto e necessidades dos usuários, foi realizada uma análise de requisitos, dividindo-se entre requisitos funcionais e não funcionais. Ainda, outra etapa importante foi a definição do quadro Kanban, para a organização das etapas do desenvolvimento.

Em relação ao banco de dados foi utilizado o banco de dados PostgreSQL de forma *serverless*, por meio da ferramenta neon. Já o *backend* a *Application Programming Interface* (API) foi desenvolvida utilizando a linguagem C#, juntamente com o *Entity Framework*, sendo responsável pelo processamento das requisições e regras de negócio do sistema. A hospedagem dessa API foi realizada na plataforma Railway, com intermédio do Docker. Referente ao *frontend*, foi criado com o *framework* Next.js, visando desempenho e uma maior robustez para auxiliar no desenvolvimento de uma aplicação *mobile-first*. Esta seção descreverá as etapas chave do desenvolvimento da aplicação e as metodologias utilizadas.

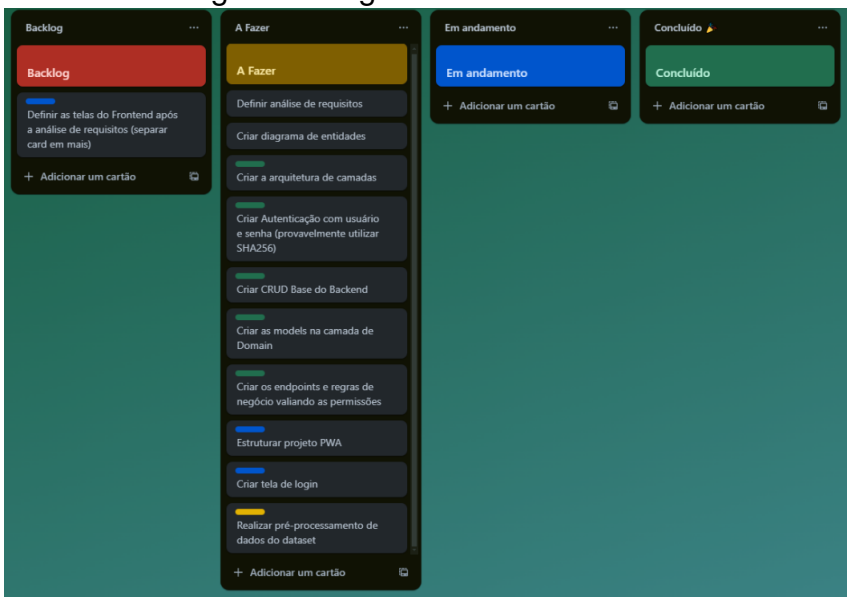
2.1 MODELAGEM E PLANEJAMENTO

Para a construção de uma aplicação ser realizada com excelência, a análise de requisitos é de extrema importância. Isso se comprova, pois para Lopes, Majdenbaum e Audy (2003), mesmo que um programa seja codificado e bem projetado, se não for corretamente analisado, ele poderá não corresponder à expectativa final e pode trazer problemas ao desenvolvedor.

Sendo assim, foi realizada uma análise de requisitos, sendo dividida entre requisitos funcionais e não funcionais. Os requisitos funcionais descrevem as funcionalidades que o sistema deve oferecer, ou seja, as ações e comportamentos esperados. Já os requisitos não funcionais

referem-se às características de qualidade do sistema, como desempenho, segurança, usabilidade e manutenibilidade. Após a definição dos requisitos funcionais, foi criado um quadro Kanban, por meio da ferramenta *online* Trello. O quadro foi dividido nas colunas, que representam cada etapa do desenvolvimento: Backlog, A Fazer, Em Andamento e Concluído. Após a definição do quadro, todas as tarefas foram criadas, baseadas na análise de requisitos, conforme mostra a Figura 1

Figura 1 - Figura inical do Kanban.

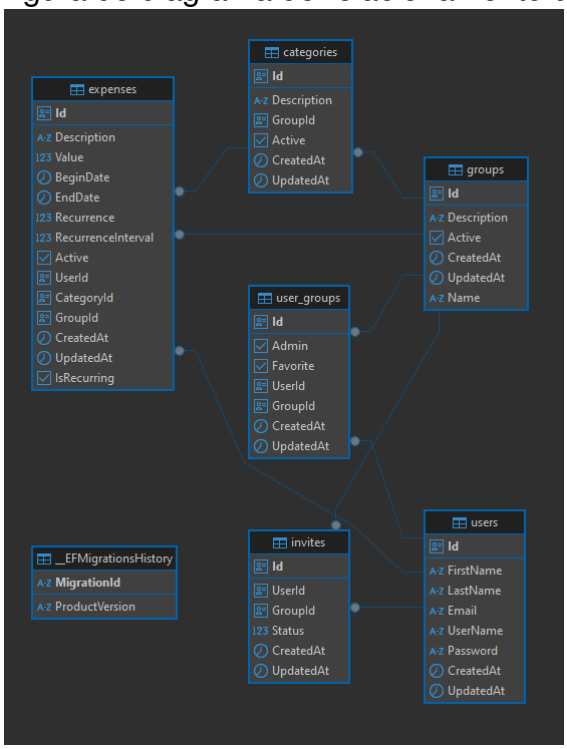


Fonte: Elaborado pelo autor.

2.2 BANCO DE DADOS

Para o gerenciamento dos dados armazenados, foi utilizado o PostgreSQL, como banco de dados relacional, implantado por meio da plataforma Neon - um serviço *serverless* que oferece escalabilidade automática e alta disponibilidade, facilitando a manutenção e o desempenho da aplicação. A escolha pelo PostgreSQL se deve à sua robustez, suporte a transações ACID e compatibilidade integral com o *Entity Framework Core*, atendendo plenamente aos requisitos de desempenho e consistência exigidos pelo sistema de divisão de despesas em grupo. A estrutura do banco de dados foi projetada seguindo os princípios de normalização, com ênfase na integridade referencial e na minimização de redundâncias, contendo entidades principais como usuários, grupos, categorias e despesas, conforme mostra a Figura 2.

Figura 2 - Figura do diagrama de relacionamento de entidade.



Fonte: Elaborado pelo autor.

A modelagem da tabela de despesas apresenta uma abordagem que preza pela performance para o tratamento de recorrências. Em vez de armazenar cada ocorrência como um registro separado, o que levaria a uma gama desnecessária de dados, o sistema mantém um único registro com metadados que definem o padrão de recorrência, como mostra a tabela *expenses* ainda na Figura 2. Essa estratégia permite calcular dinamicamente as ocorrências durante as consultas, visando trazer vantagens significativas como otimização de armazenamento, facilidade de manutenção e garantia de consistência. Todos os valores de data são armazenados em UTC, assegurando precisão temporal independente do fuso horário do usuário.

A plataforma Neon, com sua arquitetura *serverless*, complementa essa otimização oferecendo escalabilidade automática para lidar com variações na carga de trabalho. Essa combinação entre modelagem e infraestrutura adequada busca resultar em um sistema capaz de lidar eficientemente com o cálculo de despesas recorrentes, mesmo em cenários com grande volume de dados.

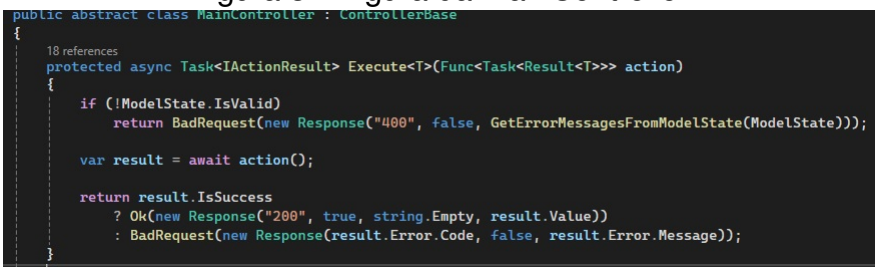
2.3 APLICAÇÃO BACKEND

Para o desenvolvimento do *backend*, optou-se pela utilização do C# com .NET 8.0, por ser uma plataforma robusta e amplamente utilizada em aplicações empresariais. O .NET oferece um ecossistema maduro com suporte nativo a injeção de dependência, validação de dados e construção de APIs RESTful, os quais são aspectos fundamentais em sistemas que exigem precisão e escalabilidade.

2.3.1 Camadas da aplicação

A arquitetura do sistema foi organizada em camadas bem definidas, que visam a separação de responsabilidades e escalabilidade do código. A camada de API é responsável por receber as requisições HTTP, utilizando de *controllers*, que herdam de uma classe *MainController* que centraliza validações comuns e retorna sempre uma resposta padrão da API, demonstrada pela Figura 3.

Figura 3 - Figura da MainController.



```
public abstract class MainController : ControllerBase
{
    18 references
    protected async Task<IActionResult> Execute<T>(Func<Task<Result<T>>> action)
    {
        if (!ModelState.IsValid)
            return BadRequest(new Response("400", false, GetErrorMessagesFromModelState(ModelState)));

        var result = await action();

        return result.IsSuccess
            ? Ok(new Response("200", true, string.Empty, result.Value))
            : BadRequest(new Response(result.Error.Code, false, result.Error.Message));
    }
```

Fonte: Elaborado pelo autor.

A camada *Application* contém os casos de uso do sistema, orquestrando as operações necessárias e coordenando a comunicação entre as demais camadas. A camada *Business* concentra as entidades de domínio e as principais validações de entidade e regras de negócio. Para a realização dessas, foi utilizado o pacote *FluentValidation* (versão 11.11.0), uma biblioteca que permite definir regras de forma declarativa e reutilizável, garantindo que os dados estejam consistentes antes de serem processados, tendo sua utilização demonstrada conforme a Figura 4. Além dessas, uma camada Infra foi criada para servir de funções que qualquer outra camada pudesse utilizar.

Figura 4 - Figura da utilização do *FluentValidation* para validar o Usuario.

```
3 references
public class UserValidation : AbstractValidator<User>
{
    2 references
    public UserValidation()
    {
        RuleFor(u => u.FirstName)
            .NotEmpty().WithMessage("O campo {PropertyName} deve ser preenchdo")
            .Length(2, 100).WithMessage("O campo {PropertyName} deve ter entre {MinLength} e {MaxLength} caracteres")
            .WithName("Primeiro nome");

        RuleFor(u => u.LastName)
            .NotEmpty().WithMessage("O campo {PropertyName} deve ser preenchdo")
            .Length(2, 100).WithMessage("O campo {PropertyName} deve ter entre {MinLength} e {MaxLength} caracteres")
            .WithName("Último nome");

        RuleFor(u => u.Email)
            .NotEmpty().WithMessage("O campo {PropertyName} deve ser preenchdo")
            .Length(2, 100).WithMessage("O campo {PropertyName} deve ter entre {MinLength} e {MaxLength} caracteres")
            .EmailAddress().WithMessage("O campo {PropertyName} está em um formato inválido");

        RuleFor(u => u.Password)
            .NotEmpty().WithMessage("O campo {PropertyName} deve ser preenchdo")
            .Length(6, 255).WithMessage("O campo {PropertyName} deve ter entre {MinLength} e {MaxLength} caracteres")
            .WithName("Senha");
    }
}
```

Fonte: Elaborado pelo autor.

O acesso ao banco de dados é gerenciado pela camada *Data*, que utiliza o *Entity Framework Core* vcolocar (EF) como *Object-Relational Mapping* (ORM). O EF foi escolhido por sua integração nativa com o .NET, sua capacidade de mapeamento objeto-relacional fluente e seu suporte a migrações, que facilitam a evolução do esquema do banco de dados ao longo do tempo. Além disso, o EF Core oferece suporte a consultas LINQ, permitindo escrevê-las de forma expressiva e com verificação em tempo de compilação, reduzindo erros comuns em SQL puro.

2.3.2 Padrões de design seguidos

Com o objetivo de aumentar a escalabilidade e legibilidade, foram seguidos alguns *Design Patterns* na construção da aplicação, sendo os principais: o *Repository Pattern*, *Result Pattern* e o *Unit Of Work*.

Para otimizar o gerenciamento das transações e diminuir o acoplamento entre a lógica de negócio e a persistência de dados, foi implementado o *Repository Pattern* em conjunto com o *Unit Of Work* (UOW). O Repository Pattern abstrai as operações de banco de dados, restringindo cada repositório a um *DbSet* específico e evitando consultas indiscriminadas, sendo necessária a criação de um repositório genérico que possa ser utilizado pelos repositórios especializados da entidade. Já o UOW garante que operações múltiplas sejam executadas atomicamente, mantendo a consistência dos dados e que, em caso de falha, desfaça qualquer operação executada.

Com relação ao tratamento de erros, evitou-se o uso excessivo

de exceções, substituindo-o pelo *Result Pattern*, um padrão funcional que encapsula o resultado de uma operação em um objeto explícito, indicando sucesso (*Success*) ou falha (*Error*). Essa abordagem traz maior clareza ao fluxo do código, elimina o custo excessivo do servidor associado ao tratamento de exceções e obriga o desenvolvedor a lidar explicitamente com possíveis falhas. Na Figura 5, é possível ver o *Result Pattern* e o UOW sendo utilizados em conjunto no mesmo método. Em casos de erros não tratados, um *middleware* global de exceções captura a falha e retorna a mesma resposta padronizada que a camada de API retorna quando ocorre alguma falha, conforme mostra a Figura 6

Figura 5 - Método de inserção de um Grupo na camada *Application*.

```
public async Task<Result<IdView>> Add(GroupDto dto)
{
    await unityOfWork.BeginTransactionAsync();

    var group = new Group(dto.Name, dto.Description);

    await groupService.Add(group);

    if (notifier.HasNotification())
    {
        await unityOfWork.RollbackTransactionAsync();
        return Result.Failure<IdView>(new Error("400", notifier.GetNotificationMessage()));
    }

    var userGroup = new UserGroup(tokenHelper.GetUserIdFromClaim(), group.Id, true, false);

    await userGroupService.Add(userGroup);

    if (notifier.HasNotification())
    {
        await unityOfWork.RollbackTransactionAsync();
        return Result.Failure<IdView>(new Error("400", notifier.GetNotificationMessage()));
    }

    await unityOfWork.CommitTransactionAsync();

    return Result.Success(mapper.Map<IdView>(group));
}
```

Fonte: Elaborado pelo autor.

Figura 6 - *Middleware* para retorno de exceções não tratadas.

Fonte: Elaborado pelo autor.

2.3.3 Segurança da aplicação

A segurança da API foi implementada utilizando *JSON Web Tokens* (JWT), um padrão amplamente adotado para autenticação *stateless*. O processo inicia-se com o login, onde o usuário recebe um token assinado com uma chave secreta após a validação das credenciais. Todas as requisições subsequentes devem incluir esse token no cabeçalho *Authorization*, o qual é validado por um *middleware* que verifica sua autenticidade e extrai as informações do usuário.

Além disso, em grande parte das operações, o sistema recupera do token JWT o identificador do usuário para validar se a ação solicitada está dentro de suas permissões. Essa verificação é fundamental para garantir a segurança e integridade dos dados, impedindo que usuários acessem ou modifiquem informações que não lhes pertencem. Um exemplo claro dessa validação ocorre quando um usuário tenta consultar as categorias de um grupo específico: antes de retornar os dados, o sistema verifica se o usuário autenticado realmente faz parte desse grupo, rejeitando a requisição caso não tenha a permissão necessária.

2.3.4 Cálculo de despesas

Prezando pela performance, foi criado um serviço responsável pelo cálculo de despesas recorrentes, o qual implementa a lógica para determinar quantas vezes uma despesa se repete dentro de um determinado período. Essa abordagem busca trazer benefícios como redução no volume de dados armazenados e flexibilidade para ajustes nas regras de recorrên-

cia. O cálculo considera diferentes padrões como diário, semanal, mensal e personalizado, sempre trabalhando com datas em UTC para garantir consistência em operações que envolvem usuários em diferentes fusos horários.

A integração entre a camada de persistência e o serviço de cálculo é feita de forma eficiente, com o EF mapeando corretamente a estrutura do banco de dados para os objetos de domínio. Quando uma consulta por período é realizada, o sistema recupera as despesas relevantes e aplica o cálculo de recorrência apenas nos registros necessários, podendo ser demonstrado pelas Figuras 7 e 8.

Figura 7 - Serviço para calcular ocorrências da despesa.

```
2 references
public int CalculateOcurrencesByDateRange(Expense expense, DateTime start, DateTime end)
{
    if (!expense.IsRecurring) return (expense.BeginDate >= start && expense.BeginDate <= end) ? 1 : 0;


    var effectiveStart = expense.BeginDate > start ? expense.BeginDate : start;
    var effectiveEnd = expense.EndDate.HasValue && expense.EndDate.Value < end ? expense.EndDate.Value : end;

    int occurrences = 0;
    switch (expense.Recurrence)
    {
        case RecurrenceType.Daily:
            occurrences = (int)(effectiveEnd - effectiveStart).TotalDays + 1;
            break;

        case RecurrenceType.Weekly:
            occurrences = ((int)(effectiveEnd - effectiveStart).TotalDays / 7) + 1;
            break;

        case RecurrenceType.Monthly:
            occurrences = ((effectiveEnd.Year - effectiveStart.Year) * 12 + effectiveEnd.Month - effectiveStart.Month) + 1;
            break;

        case RecurrenceType.Custom:
            if (expense.RecurrenceInterval > 0)
            {
                occurrences = ((int)(effectiveEnd - effectiveStart).TotalDays / expense.RecurrenceInterval) + 1;
            }
            break;
    }

    return occurrences;
}
```

Fonte: Elaborado pelo autor.

Figura 8 - Serviço para retornar o valor calculado pela ocorrência

```
public async Task<Result<List<ExpenseSummaryView>>> GetExpenseSummaryByGroup(GetExpenseSummaryByGroupDto dto)
{
    var group = await userGroupRepository.GetByUserAndGroup(tokenHelper.GetUserIdFromClaim(), dto.GroupId);
    if (group is null)
        return Result.Failure<List<ExpenseSummaryView>>
            (
            new Error
                (
                    "404", $"Grupo de código {dto.GroupId} não encontrado, ou usuário logado não pertence ao grupo"
                )
            );

    var expenses = await expenseRepository.GetByGroupAndDateRange(dto.GroupId, dto.StartDate, dto.EndDate);

    var summaryList = new List<ExpenseSummaryView>();

    foreach (var expense in expenses)
    {
        var occurrences = expenseService.CalculateOcurrencesByDateRange(expense, dto.StartDate, dto.EndDate);

        var totalValue = occurrences * expense.Value;

        summaryList.Add(new ExpenseSummaryView
        {
            Id = expense.Id,
            Description = expense.Description,
            TotalValue = totalValue,
            RecurrenceType = expense.Recurrence,
            IsRecurring = expense.IsRecurring,
            RecurrencyInterval = expense.RecurrenceInterval,
            UserName = expense.User.UserName,
            CategoryName = expense.Category.Description,
        });
    }

    return Result.Success(summaryList);
}
```

Fonte: Elaborado pelo autor.

2.4 APLICAÇÃO FRONTEND

Para o desenvolvimento da interface do sistema, foi escolhido o Next.js (versão 15.2.4) como *framework* principal, devido à sua capacidade de oferecer renderização híbrida (SSR, SSG e CSR), otimização de performance nativa e excelente suporte para PWAs. O Next.js também traz benefícios como roteamento inteligente, pré-renderização de páginas e cache eficiente, reduzindo significativamente o tempo de carregamento e melhorando a experiência do usuário.

A comunicação com o backend foi implementada utilizando Axios (versão 1.9.0) em conjunto com TanStack Query (versão 5.75.0), uma biblioteca muito utilizada para o gerenciamento de estado assíncrono. Todas as requisições foram centralizadas em uma pasta *lib*, organizada por entidades, seguindo um padrão de API *Client* que facilita a manutenção e evolução do código. O TanStack Query oferece cache automático, atualização em *background* e *stale-while-revalidate*, eliminando a necessidade de estados globais complexos para gerenciar dados da API, facilitando muito o desenvolvimento.

Além disso, o Axios foi configurado com um interceptador de requisição que insere automaticamente o token JWT armazenado em cookies no *header Authorization*, garantindo que todas as chamadas após o login

sejam autenticadas. Ainda no quesito autorização, para o controle de rotas públicas e privadas, foi utilizado o *middleware* nativo do Next.js, que verifica a presença de um cookie contendo o token de autenticação conforme a Figura 9. Caso o cookie não exista, o usuário é redirecionado para a página de login, caso contrário, tem acesso às rotas privadas.

Figura 9 - *Middleware* para verificação das rotas

```
export async function middleware(request: NextRequest) {
  const path = request.nextUrl.pathname;
  const publicRoute = publicRoutes.find((route) => route.path === path);
  const authToken = request.cookies.get("token_share_the_bill")?.value;
  // Se a rota é pública e não há token, permita o acesso
  if (!authToken && publicRoute) {
    return NextResponse.next();
  }

  // Se não há token e a rota não é pública, redirecione para o login
  if (!authToken && !publicRoute) {
    const redirectUrl = request.nextUrl.clone();
    redirectUrl.pathname = REDIRECT_WHEN_NOT_AUTHENTICATED_ROUTE;
    return NextResponse.redirect(redirectUrl);
  }

  // Se há token e a rota é pública com redirecionamento, redirecione para a página inicial
  if (
    authToken &&
    publicRoute &&
    publicRoute.whenAuthenticated === "redirect"
  ) {
    const redirectUrl = request.nextUrl.clone();
    redirectUrl.pathname = REDIRECT_WHEN_AUTHENTICATED_ROUTE;
    return NextResponse.redirect(redirectUrl);
  }

  // Se há token e a rota não é pública, verifique se o token é válido
  if (authToken && !publicRoute) {
    const isValid = isTokenValid(authToken);

    if (!isValid) {
      const redirectUrl = request.nextUrl.clone();
      redirectUrl.pathname = REDIRECT_WHEN_NOT_AUTHENTICATED_ROUTE;

      const response = NextResponse.redirect(redirectUrl);
      response.cookies.delete("token_share_the_bill"); // Remove o token inválido
      return response;
    }

    return NextResponse.next(); // Token válido, permita o acesso
  }

  return NextResponse.next();
}
```

Fonte: Elaborado pelo autor.

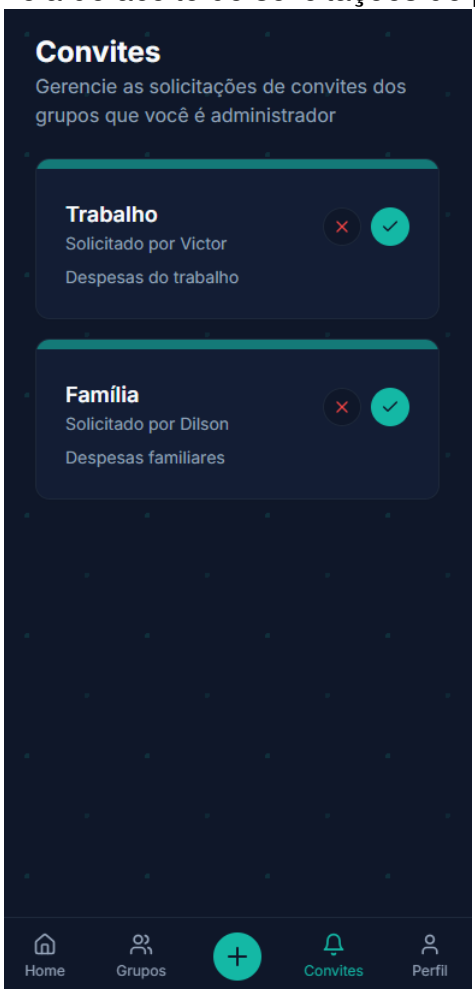
A estilização da aplicação foi feita com shadcn/ui, uma biblioteca de componentes reutilizáveis construída sobre o Tailwind CSS, que oferece alta customização e *design system* consistente. A interface foi desenvolvida com foco em *mobile-first*, simulando a experiência de um aplicativo nativo, inclusive com uma navbar inferior, a qual é padrão em apps mobile e navegação fluida utilizando o pacote de navegação nativo do Next.js, o next/navigation. A responsividade foi garantida em todos os componentes, assegurando uma boa experiência em qualquer dispositivo, preparando-a

para ser transformada em uma PWA com recursos como cache de *assets* e carregamento instantâneo. Algumas telas da aplicação que demonstram essas características estão exemplificadas nas Figuras 10 e 11

Figura 10 - Tela do gerenciamento do grupo

Fonte: Elaborado pelo autor.

Figura 11 - Tela do aceite de solicitações de participação



Fonte: Elaborado pelo autor.

Para a validação de formulários, foi adotado o Zod (versão do zod), uma biblioteca de validação *schema-first* que oferece tipagem segura e integração perfeita com TypeScript. O Zod foi utilizado tanto para validar dados de entrada quanto para garantir a conformidade dos objetos antes do envio para a API, na tentativa de uma redução de erros.

2.5 DEPLOY DA APLICAÇÃO

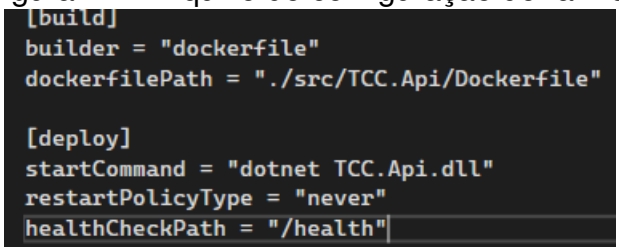
A implantação da aplicação foi realizada utilizando plataformas modernas de hospedagem que oferecem integração contínua com o repositório do GitHub, garantindo atualizações automáticas sempre que novas alterações são enviadas para a *branch* principal.

Para o frontend, desenvolvido em Next.js, optou-se pela Vercel, plataforma que oferece suporte nativo ao framework, proporcionando recursos como pré-renderização automática, cache inteligente e distribuição

global de conteúdo através de sua CDN. A integração com o GitHub foi configurada para acionar um novo deploy sempre que alterações são enviadas para a *branch master*, assegurando que a versão em produção esteja sempre atualizada com as últimas modificações.

O backend, construído em .NET 8, foi implantado no Railway, utilizando seu plano básico pago que oferece o necessário para a aplicação em seu estágio atual. Para isso, foi criado um Dockerfile, contendo todas as configurações necessárias para *v* e execução da aplicação. Complementando esta configuração, um arquivo railway.toml foi adicionado ao projeto com as diretrizes específicas para o *deploy*, incluindo o caminho do Dockerfile, o comando de inicialização e o caminho para o health check da API conforme mostra a Figura 12. A integração contínua com o GitHub também foi configurada no Railway, garantindo que cada atualização na branch principal resulte em um novo *deploy* automático. Ambas as plataformas oferecem monitoramento básico e logs em tempo real, permitindo acompanhar o comportamento da aplicação em produção, permitindo fácil escalabilidade para implementações futuras.

Figura 12 - Arquivo de configuração do railway



Fonte: Elaborado pelo autor.

Com os materiais definidos e os métodos bem estabelecidos, o desenvolvimento da aplicação pôde seguir um caminho claro e consistente. Cada etapa, desde a organização das tarefas até a definição da arquitetura e das tecnologias utilizadas, foi essencial para garantir a fluidez do processo. A estrutura construída ao longo dessa fase serviu como base para a implementação das funcionalidades propostas, permitindo que as próximas etapas do projeto fossem conduzidas com maior segurança e previsibilidade.

3 DISCUSSÃO E RESULTADOS

Durante o desenvolvimento da aplicação, ficou evidente a importância da etapa da análise. Antes mesmo de iniciar o desenvolvimento

propriamente dito foi fundamental compreender o problema a ser resolvido, identificar as necessidades e traduzir em funcionalidades claras e factíveis. Esse processo inicial ajudou a guiar as decisões técnicas ao longo do projeto, evitar retrabalhos, uma vez que os objetivos estavam bem definidos desde o início.

Paralelamente, o uso do método Kanban para organizar as atividades teve papel essencial na condução do desenvolvimento. A visualização das tarefas em um fluxo contínuo permitiu uma melhor gestão do tempo e das prioridades. Essa abordagem contribuiu diretamente para manter o foco nas entregas mais relevantes, garantindo que cada etapa do desenvolvimento fosse concluída com qualidade e dentro dos prazos estabelecidos.

Com a base do projeto bem fundamentada, a adoção de boas práticas de engenharia de software influenciou diretamente na qualidade do sistema e na capacidade de evoluí-lo. Desde o começo, houve a preocupação de organizar a estrutura do código com base em princípios como separação de responsabilidades, modularidade e reutilização, o que se mostrou essencial para manter a clareza do sistema à medida que novas funcionalidades eram implementadas.

A arquitetura do *backend* foi pensada para promover independência entre as camadas, com a lógica de negócios bem isolada das regras de apresentação e persistência, facilitando a manutenção do código. Com a definição de interfaces e contratos entre os módulos, tornou-se mais simples implementar mudanças sem comprometer outras partes do sistema, o que é um fator determinante para a escalabilidade da solução.

Além disso, a combinação de C# com EF proporcionou um ambiente produtivo para a implementação das regras de negócio, enquanto mantinha o controle sobre as operações de persistência. A organização do código seguindo padrões como *Repository* e *Unit of Work* facilitou a manutenção e a testabilidade do sistema. Junto desses padrões, padrão *Result*, utilizado extensivamente na camada de aplicação, permitiu um fluxo explícito de tratamento de falhas que melhora a robustez do sistema.

O foco na usabilidade e na construção de uma interface responsiva desde o início do projeto e a utilização de tecnologias como o Tailwind CSS, contribuíram para que a aplicação se mantivesse acessível em diferentes dispositivos, sem necessidade de retrabalho posterior. Essa abordagem *mobile-first* preza por garantir que a experiência do usuário seja consistente, especialmente considerando que boa parte do uso previsto da aplicação se daria em *smartphones*.

Outro ponto que se destacou foi a forma como a aplicação lidou com as despesas recorrentes e o compartilhamento de dados entre membros de um grupo. A lógica por trás dessas funcionalidades foi construída de maneira flexível e extensível, o que abre caminho para futuras melhorias, como integrações externas ou automações mais complexas. Com o sistema bem estruturado, a implementação dessas novas capacidades tende a ser mais simples e menos arriscada.

Em termos de resultados, ainda que os testes tenham sido conduzidos em um ambiente controlado e com um número limitado de usuários, ficou evidente que a organização do projeto e o cuidado com a estruturação do código contribuíram diretamente para a estabilidade e fluidez do sistema. Isso reforça a importância de alinhar o desenvolvimento não apenas às necessidades imediatas do projeto, mas também às boas práticas que garantem sua evolução ao longo do tempo.

Por fim, a aplicação foi implantada em ambiente de produção, utilizando um serviço de hospedagem compatível com os requisitos da *stack* adotada. Esse processo de *deploy* foi fundamental não apenas como uma exigência do projeto, mas como uma etapa que confirmou a maturidade da solução desenvolvida. O fato de a aplicação estar acessível publicamente valida tanto a sua viabilidade técnica quanto a sua utilidade prática, encerrando o ciclo de desenvolvimento com um produto funcional, testado e pronto para uso real.

Apesar dos resultados positivos alcançados com o desenvolvimento da aplicação, é importante reconhecer algumas limitações que acompanharam o estudo. Uma das principais restrições está relacionada à dependência de conectividade, devido ao fato de ser uma aplicação web, seu funcionamento está diretamente atrelado ao acesso à internet. Isso limita seu uso em cenários *offline*, o que pode ser um fator relevante para determinados grupos de usuários, especialmente em regiões com acesso instável à rede.

No que diz respeito aos trabalhos relacionados, embora existam diversas soluções voltadas à organização financeira pessoal e ao controle de despesas coletivas, muitas delas carecem de recursos específicos para a gestão colaborativa em tempo real, especialmente com foco em recorrência de despesas e divisão clara entre os membros. A proposta desenvolvida neste estudo buscou justamente preencher essa lacuna, com uma abordagem que une praticidade, usabilidade e foco em colaboração, ainda que com escopo mais restrito em comparação às soluções consolidadas do

mercado. Mesmo assim, esse diferencial foi suficiente para demonstrar o potencial de inovação da aplicação dentro do seu recorte proposto.

4 CONCLUSÃO

Este trabalho teve como objetivo principal desenvolver uma aplicação para controle de despesas em grupo, permitindo o acompanhamento e compartilhamento de gastos de forma colaborativa, com funcionalidades específicas para despesas recorrentes e visualização integrada do orçamento. Os objetivos propostos foram alcançados por meio da aplicação de metodologias ágeis, modelagem de banco de dados eficiente e uma arquitetura de software bem definida, consolidando conhecimentos técnicos adquiridos ao longo do curso de Ciências da Computação.

Os resultados demonstraram que a aplicação desenvolvida atende às necessidades identificadas na justificativa do projeto, oferecendo uma solução diferenciada em relação a ferramentas existentes, que frequentemente negligenciam a gestão financeira coletiva. A implementação de cálculos dinâmicos para despesas recorrentes, a segurança baseada em JWT e a interface *mobile-first* destacam-se como contribuições relevantes, alinhadas com os requisitos de usabilidade e escalabilidade inicialmente propostos.

Além disso, o projeto reforçou a importância de boas práticas de engenharia de software, como a separação de camadas, padrões de design e tratamento de erros robusto, que foram essenciais para a manutenibilidade e evolução do sistema. E a implantação em produção, validou a viabilidade técnica da solução.

Como trabalhos futuros, sugere-se a implementação de análises preditivas com *machine learning* para projeção de gastos, integração com serviços financeiros externos (como bancos e carteiras digitais) e a adição de funcionalidades *offline* para ampliar a acessibilidade.

Por fim, este estudo não apenas cumpriu seu propósito acadêmico, integrando conhecimentos técnicos e metodológicos do curso, mas também ofereceu uma solução prática para um problema relevante na sociedade: a falta de ferramentas eficientes para educação financeira coletiva. A aplicação desenvolvida representa um passo inicial para futuras otimizações, reforçando o potencial da computação como aliada na organização financeira pessoal e familiar.

REFERÊNCIAS

ANDRADE, I. K. M. et al. **Aplicativo para controle financeiro de estudantes universitários**. 2024. e3641–e3641 p.

COELLA, M. T. et al. **Planejamento Financeiro Familiar: A importância da organização e controle no orçamento familiar**. 2014.

CORDEIRO, N. J. N.; COSTA, M. G. V.; SILVA, M. N. da. **Educação Financeira no Brasil: uma perspectiva panorâmica**. 2018. 69–84 p.

JOHRI, E. et al. Expense management system. In: IEEE. **2023 4th IEEE Global Conference for Advancement in Technology (GCAT)**. [S.l.], 2023. p. 1–6.

LOPES, L. T.; MAJDENBAUM, A.; AUDY, J. L. N. Uma proposta para processo de requisitos em ambientes de desenvolvimento distribuído de software. In: **WER**. [S.l.: s.n.], 2003. p. 329–342.

MANEKAR, A. et al. **smart expense tracker application using naive bayes**.

MELO, L. A. de; PEDRO, J. G. **a escassex de educação financeira no sistema educacional brasileiro**. 2023. e234845–e234845 p.

SANTOS, J. A. d.; CRUZ, R. S. R. d. et al. **Gestão Financeira Familiar: aplicativo compartilhado para planejamento, monitoramento e controle do orçamento**. [S.l.]: 121, 2023.

TAMIZHSELVI, A.; ANBU, M.; RADHAKRISHNAN, K. Financial and individual future expense prediction based on frequent patterns using micro services. In: IEEE. **2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT)**. [S.l.], 2022. p. 532–536.