Sistema de Gerenciamento para Empresas de Logística

Leonardo Peripolli Pereira, Fabrício Tonetto Londero
Curso de Sistemas de Informação
UFN - Universidade Franciscana
Santa Maria - RS
leonardo.ppereira@ufn.edu.br, fabriciotonettolondero@gmail.com

***Resumo*—Este artigo apresenta o projeto e desenvolvimento de um sistema web para realizar o gerenciamento de pagamentos de entregadores de uma empresa de logística *last mile*, com base na quantidade de mercadorias entregues em determinado período. A demanda surgiu da necessidade de facilitar a comunicação interna entre funcionários administrativos, bem como para melhorar a manipulação dos dados. Foram empregadas a metodologia FDD juntamente com a técnica Kanban para gestão de atividades. O trabalho foi elaborado visando o desenvolvimento com ASP .Net Core por meio da estrutura Blazor, integrado com o SGBD Microsoft SQL Server. O trabalho, ao cumprir os requisitos propostos, atingiu os objetivos de melhorar o fluxo das informações e estruturar os processos da empresa.**

***Palavras-chave*: Desenvolvimento web; gestão de pagamentos; logística *last mile*;**

I. Introdução

O mercado de logística *last mile* de e-commerce tem experimentado um crescimento exponencial nos últimos anos, impulsionado pela crescente demanda dos consumidores por entregas rápidas e eficientes. Em paralelo, a gestão das empresas que operam nesse setor tornou-se cada vez mais complexa, exigindo soluções inovadoras para garantir a eficiência operacional e a satisfação do cliente. O desenvolvimento de um sistema de pagamentos eficiente é crucial para a sustentabilidade dessas empresas, permitindo uma melhor gestão de fluxos de caixa e garantindo a segurança dos dados financeiros.

O contínuo avanço das tecnologias tem sido um aliado fundamental na gestão das empresas de logística, especialmente no que tange ao controle de processos e à otimização de operações. A necessidade de sistemas que permitam um controle rigoroso e automatizado dos processos é cada vez mais evidente, dada a complexidade e a escala das operações logísticas no contexto do e-commerce. Esses sistemas não apenas facilitam a administração financeira, mas também aumentam a confiança dos *stakeholders* através de registros precisos e acessíveis em tempo real.

Nesse contexto, o presente trabalho tem como objetivo projetar um sistema de pagamentos de prestação de serviços dos entregadores para uma empresa de logística, focado em melhorar a eficiência e a precisão das operações financeiras. A metodologia adotada inclui a análise das necessidades específicas da empresa, o levantamento de requisitos e o desenvolvimento de diagramas. Espera-se que o sistema proposto contribua significativamente para a melhoria da gestão financeira da empresa, proporcionando um modelo que possa ser replicado em outras organizações do setor.

A. Justificativa

Atualmente uma empresa gaúcha de logística depende de diversas planilhas de quantidades de volumes entregues e outra planilha de integração dos dados para realização do pagamento referente aos volumes entregues pelos motoristas, além do envio via e-mail para comunicação interna. Ações como gerenciar os pagamentos [1] de cada entregador, bem como o controle da quantidade de mercadoria entregue e seus respectivos valores a receber, de acordo com o período de fechamento realizado tornam o processo burocrático e com questões de integridade fragilizadas. Com base nisso, pretende-se desenvolver um sistema [2] que facilite a comunicação entre os funcionários, além de melhorar a manipulação dos dados.

B. Objetivos

Esse trabalho tem como objetivo desenvolver um sistema web intuitivo que permita um bom fluxo de comunicação entre os funcionários da empresa durante os períodos de fechamento de serviços, além de facilitar a inserção e a manipulação dos dados de entrega e de pagamento dos entregadores. Para que os objetivos gerais sejam alcançados, identificaram-se outros objetivos específicos:

- Pesquisar, aplicar e escrever sobre banco de dados, desenvolvimento web e ferramentas;
- Pesquisar, aplicar e escrever sobre metodologias ágeis;
- Pesquisar, aplicar e escrever sobre diagramas UML;
- Estudar e aplicar boas práticas referentes a qualidade de software;
- Desenvolver a estrutura e o código-fonte do sistema.

II. Referencial teórico

Esta seção apresenta os fundamentos das temáticas abordas durante a pesquisa e expõe conceitos sobre as tecnologias a serem utilizadas durante a fase de desenvolvimento do sistema.

A. *Cadeia logística de e-commerce*

E-commerce é a definição de "comércio eletrônico", ou seja, diz respeito a um fornecedor que negocia bens e serviços por meio do uso da internet e de mídias eletrônicas. A venda ocorre por meio de um carrinho de compras virtual e um gateway de pagamento, que geralmente processa cartões de crédito, débito, boleto [3] e pix.

Com o crescente avanço da tecnologia da informação, torna-se imprescindível evidenciar que o e-commerce é uma via beneficiária de mão dupla, isto é, vantajoso tanto para os fornecedores, quanto para os clientes. Para o primeiro, pois não necessitam de loja física e podem vender para todo o país de maneira descomplicada. Para o último, pois podem analisar o custo-benefício de diversos concorrentes sem a necessidade de percorrer distâncias fisicamente [3].

O atual esquema logístico do e-commerce varejista de pequenos produtos compreende o envio da loja para uma transportadora privada, o armazenamento, a expedição e o transporte até uma empresa que atue especificamente no ramo de *last mile* na região de endereço do destinatário. O segmento da logística denominado *last mile* (última milha) se refere ao trecho final de uma cadeia de suprimentos, portanto, uma empresa que presta serviços desse gênero é responsável por realizar entregas ao destinatário final [4].

Ao analisar a Figura 1, é possível observar as características de cada etapa da cadeia logística, bem como diferenciar o *last mile* do *first mile* e do *middle mile*. A primeira milha se caracteriza pela introdução do produto na rede logística, ou seja, pelo envio da fábrica ou comércio para o operador logístico. Por sua vez, o operador é responsável pela etapa seguinte, da milha intermediária, que consiste na distribuição dos produtos em armazéns intermediário ou diretamente para os agentes de *last mile*.

Figura 1. Etapas da cadeia logística [5].

Uma companhia desse setor deve atender uma região preestabelecida no seu estado, devendo realizar entregas em todas as cidades e localidades dessa área. Para realizar esse feito, faz-se necessário contratar entregadores em todos os municípios, e, a depender da quantidade de habitantes, pode ser fundamental o acordo com diversos motoristas na mesma região.

Além disso, faz-se necessário destacar que o trecho *last mile* é considerado uma etapa crítica da rede logística, dado que trata diretamente com a experiência do cliente, e pode influenciar tanto de maneira positiva quanto de maneira negativa, sendo um fator fundamental para que o cliente volte a fazer negócios com o comércio eletrônico em questão.

B. *Sistemas de Informação*

Segundo Stair [6], "sistema de informação é um conjunto de elementos ou componentes inter-relacionados que coleta, manipula, armazena e dissemina dados e informações, e fornece reação corretiva para alcançar um objetivo", ou seja, o usuário fornece as informações de entrada, o sistema manipula essas informações por meio de processamento, a fim de as transformar em dados úteis e conceder a saída dos dados, geralmente em forma de documentos ou relatórios. Por último, ocorrem os *feedbacks*, que são informações oriundas do software, utilizadas para realização de mudanças nos processos do sistema.

Apesar das individualidades, benefícios e malefícios de cada sistema, é por meio do uso de sistemas de informação que as organizações podem obter vantagem competitiva em comparação com empresas concorrentes. Portanto, a utilização correta da tecnologia pode ser um fator determinante na geração de valor e na percepção de competência por clientes e fornecedores [2].

Conforme a Figura 2, de Laudon, as empresas podem ser divididas em quatro níveis (operacional, conhecimento, gerencial e estratégico) e cinco áreas funcionais (contabilidade, finanças, vendas, fabricação e recursos humanos) [7].

Figura 2. Tipos de Sistemas de Informação [7].

Podem ser citados 4 principais tipos de sistemas de informação de acordo com a Figura 2, um para cada nível. Os Sistemas de Processamento de Transações atuam no nível operacional e são responsáveis por registrar as transações rotineiras essenciais de um negócio, como um registro de pedidos de venda; os Sistemas de Trabalhadores do Conhecimento atendem ao nível de conhecimento das empresas e agem na criação de informações úteis para um trabalho eficiente; os Sistemas de Informação Gerenciais

operam no nível gerencial e apoiam principalmente nas questões de planejamento, controle e decisão, geralmente ao gerar relatórios; por último, os Sistemas de Apoio Executivo atuam no nível estratégico e são empregados na abordagem de decisões não rotineiras, que exigem análise, percepção e avaliação, como uma previsão anual de orçamentos.

C. Tecnologias

1) GitHub: GitHub é uma plataforma gratuita baseada em nuvem que possibilita trabalhar, armazenar e compartilhar códigos-fonte. Por se tratar de uma ferramenta fundamentada em Git, ela permite que um grupo de pessoas trabalhe no mesmo código ao mesmo tempo de maneira eficiente, pois realiza o versionamento de código. O sistema de controle de versões evita que erros simples aconteçam, como o uso de diferentes versões do código-fonte e a dessincronização de trabalhos paralelos. Além disso, ela permite o acesso e a recuperação de versões anteriores em caso de erro ou falha [8].

2) C#, .NET e ASP.NET Core: C# é uma linguagem de programação orientada a objetos e de tipagem forte desenvolvida pela Microsoft para executar programas de maneira segura e robusta por meio da plataforma .NET [9].

Por sua vez, .NET é uma aplicação gratuita para desenvolvimento de sistemas. Pode interpretar diversas linguagens de programação, porém a mais utilizada é o C#. Seus principais fundamentos são confiabilidade, segurança, desempenho e produtividade [9].

O ASP.NET Core é uma estrutura de código aberto que permite o desenvolvimento de aplicativos web na plataforma .NET. Os projetos elaborados por meio dessa estrutura seguem o padrão MVC (*Model-View-Controller*), que facilita a separação de responsabilidades. Nesse modelo, as requisições dos usuários são direcionadas a um Controlador, responsável por interagir com o Modelo para executar as ações solicitadas. O Controlador, então, seleciona a Visualização apropriada para apresentar ao usuário, fornecendo os dados necessários obtidos do Modelo [10]. Complementando o ASP.NET Core, o Blazor é uma estrutura moderna de front-end baseada em HTML, CSS e C# que possibilita o desenvolvimento ágil de aplicativos Web por meio da plataforma .NET [11].

3) Entity Framework: Entity Framework atua como um ORM (*Object-Relational Mapping*) e estabelece uma estrutura de comunicação segura de alto nível entre o .NET e o banco de dados, proporcionando suporte a consultas, atualizações e controle de alterações [12]. Um ORM nada mais é que uma maneira automatizada de salvar os objetos de uma aplicação em uma tabela de um banco de dados relacional [13].

O Entity Framework oferece três abordagens de trabalho distintas: *Code First*, em que o modelo do banco de dados é definido com base nas classes C#; *Database First*, em que ocorre o inverso, pois as classes são geradas com base no modelo do banco de dados; e *Model First*, onde é gerado o esquema do banco de dados a partir da elaboração de um modelo no EF Designer (como um diagrama) [12].

4) SQL Server: Pertencente a Microsoft, é um Sistema de Gerenciamento de Banco de Dados do tipo relacional, ou seja, os dados são modelados com base em tabelas e relações. Suas principais características são a segurança, a facilidade na manipulação dos dados e a integração simples com outras ferramentas Microsoft. Quanto a segurança, seus pontos fortes são a autenticação integrada do Windows e a possibilidade de criptografia dos dados [14].

No caso do ASP.NET Core, a sincronização dos dados é feita de maneira descomplicada por meio do Entity Framework.

5) API RESTful: REST é um tipo de arquitetura de software que define alguns princípios para que o funcionamento de API[1] seja confiável, escalável e de alta performance. Portanto, uma API RESTful é uma interface de comunicação entre dois sistemas que se baseia nos fundamentos da arquitetura REST [16].

É importante destacar que as API realizam requisições utilizando métodos HTTP por meio de um link URL. Os quatro métodos HTTP mais comuns são GET, POST, PUT e DELETE. O GET é utilizado para ler um recurso na URL e retornar informações; POST é usado para criar novos recurso por meio de dados enviados ao servidor; PUT tem o objetivo de alterar um recurso já existente; e DELETE busca eliminar o recurso identificado na URL [16].

Além disso, prezando pela segurança digital, o JWT (JSON Web Token) é amplamente utilizado em APIs RESTful para autenticação e autorização de usuários, oferecendo uma forma compacta e segura de transmitir informações. Com sua assinatura digital, o JWT garante a integridade dos dados, dispensando a necessidade de armazenar o estado de autenticação no servidor. Se trata de um token compacto que contém três partes — cabeçalho, payload e assinatura — codificadas em Base64, permitindo autenticação com criptografia e transitando de forma segura no cabeçalho HTTP Authorization como um Bearer Token [17].

6) FDD: FDD, ou Feature Driven Development, é uma metodologia ágil de desenvolvimento de software que visa priorizar a entregas de funcionalidades com base em ordens de prioridade estabelecidas pelo cliente. A lista de funcionalidades, além de servir como guia para planejamento e desenvolvimento, pode ser utilizada para verificar o progresso do projeto [18].

Para isso, o método é composto por 5 processos que detalham o planejamento e desenvolvimento:

[1]*Application Programming Interface*, ou simplesmente API, é uma interface padronizada que possibilita a comunicação e a integração entre dois sistemas diferentes. Ademais, a API normalmente é responsável por estabelecer as regras e permissões de comunicação entre os dois sistemas, fazendo-se necessário que o desenvolvedor ajuste o seu sistema para o pleno funcionamento [15].

1 - Desenvolver um Modelo Abrangente, que consiste em realizar um estudo detalhado do domínio do negócio e definir o escopo do projeto;

2 - Construir uma Lista de Funcionalidades é a etapa em que é feito o levantamento hierárquico de requisitos em conformidade com as necessidades do cliente;

3 - Planejar Através de Funcionalidades, as funcionalidades identificadas são organizadas em iterações e detalhadas, estabelecendo um plano de projeto que guiará o desenvolvimento;

4 - Projetar Através de Funcionalidades, cada funcionalidade é detalhadamente projetada, incluindo a criação de modelos de interface do usuário e diagramas de sequência, garantindo uma base sólida para a implementação;

5 - Construir Através de Funcionalidades é o último processo, que consiste no desenvolvimento do código de acordo com plano de projeto, de maneira iterativa e incremental [18].

7) Kanban: O Kanban é uma ferramenta que atua em conjunto com a metodologia ágil, com o objetivo de auxiliar na plena execução do método utilizado. É uma técnica de gestão de mudanças que possibilita a visualização e o gerenciamento de todo o fluxo de trabalho. Todas as etapas de projeto e desenvolvimento são dispostas em um quadro, que separa o que já foi entregue, o que está em progresso e o que deverá ser feito [19].

D. Trabalhos correlatos

1) A Importância de Sistemas de Informação para a Competitividade Logística: O artigo de Nazário [2] destaca o avanço da Tecnologia da Informação e seu papel nas operações empresariais, permitindo reduções de custo e vantagem competitiva. Em particular, enfoca o impacto dos sistemas de informação no desenvolvimento da logística. Destaca-se a crescente implementação de sistemas de gestão empresarial de todos os níveis da pirâmide, e que as aplicações não se restringem apenas às grandes empresas. Esses softwares visam integrar a gestão empresarial, proporcionando uma comunicação unificada e eliminando a disparidade de informações nos relatórios gerenciais. O artigo ainda levanta questões sobre como a logística está sendo abordada nesse contexto de avanço tecnológico e implementação de sistemas de gestão. Concluiu-se que a adoção desses sistemas promete melhorar as operações logísticas e reduzir custos, mas as organizações devem avaliar cuidadosamente o valor real dessas soluções para suas necessidades específicas.

2) AGROPAY: inovação na gestão de pagamentos de funcionários rurais: O trabalho de Zimmermann [1] realiza a proposta do desenvolvimento de um sistema que facilite a gestão de pagamentos de trabalhadores rurais, em que é enfrentado o desafio da complexidade na catalogação e cálculo de salários pelos proprietários rurais. A plataforma oferece uma solução prática e automatizada, permitindo o cadastro dos trabalhadores, inserção de informações sobre os dias trabalhados e valores a serem pagos, além de uma interface intuitiva com relatórios em tempo real para controle dos gastos. Espera-se que o projeto tenha um impacto significativo no gerenciamento da remuneração dos empregados, reduzindo custos e evitando erros. Com uma organização clara e intuitiva, o site facilita o gerenciamento dos pagamentos e dos trabalhadores cadastrados. Foram utilizadas metodologias e ferramentas como UML para o desenvolvimento, visando uma arquitetura clara e dinâmica. A conclusão foi o projeto de um software com potencial de facilitar a gestão eficiente dos pagamentos e dos trabalhadores cadastrados, priorizando a privacidade dos dados.

3) Desenvolvimento de sistema de ouvidoria: No Trabalho de Conclusão de Curso de Drescher [20] é proposto o desenvolvimento de um sistema web de Ouvidoria para uma faculdade, utilizando a linguagen de programação C#, o framework ASP.NET Core, o padrão de desenvolvimento REST e o banco de dados SQL Server. Foram elaborados diagramas UML, modelo entidade-relacionamento e dicionário de dados, além do desenvolvimento de um software seguindo o paradigma DDD e arquitetado em camadas para dividir responsabilidades. O resultado foi um sistema web apresentando telas e funcionalidades principais, que visa proporcionar um canal rápido, prático e seguro de comunicação entre os alunos e a instituição. Por fim, foi constatado que a organização necessitava de um sistema como esse, pois os serviços melhoraram e a ouvidoria passou a receber mais mensagens, além de agregar valor à instituição.

4) Inferência dos Trabalhos Correlatos: Quanto ao primeiro trabalho, observa-se a importância dos sistemas de informação no ramo logístico e a relevância do presente projeto para aprimorar processos e reduzir custos. Portanto, ao desenvolver um sistema web intuitivo, buscamos melhorar a eficiência operacional conforme foi apresentado no artigo.

Igualmente à proposta do Agropay, foi reconhecida a necessidade de simplificar a gestão de pagamentos para garantir eficiência e precisão. Assim como o sistema proposto, almeja-se facilitar a administração de pagamentos e promover um impacto significativo no gerenciamento dos recursos financeiros da empresa.

Assim como na aplicação de Drescher, o intuito é projetar e implementar um sistema web intuitivo, seguro, eficiente e de alto desempenho. Portanto, analisando os resultados do sistema de ouvidoria, destaca-se que a utilização das mesmas tecnologias é fundamental para o cumprimento destes objetivos.

Apesar do contexto de ofício distinto e do uso de métodos e tecnologias diferentes das apresentadas no presente trabalho, todos os trabalhos foram de importância para prover o embasamento teórico necessário e nortear o presente projeto de sistema, além de ressaltar a necessidade de soluções práticas e automatizadas para melhorar processos dentro das empresas.

III. METODOLOGIA

O trabalho utilizou as boas práticas da metodologia ágil de desenvolvimento FDD. Esta seção descreve como o método foi aplicado e discrimina as diferentes fases do processo.

A. Desenvolver um modelo abrangente

É a fase inicial da metodologia. É realizada para entender o domínio de negócio e definir o escopo do projeto. Na Figura 3, é possível visualizar um diagrama de domínio do sistema, a fim de possibilitar uma visão geral do que será desenvolvido. Esse diagrama compreende a parte do mundo real que é relevante para o desenvolvimento do sistema.

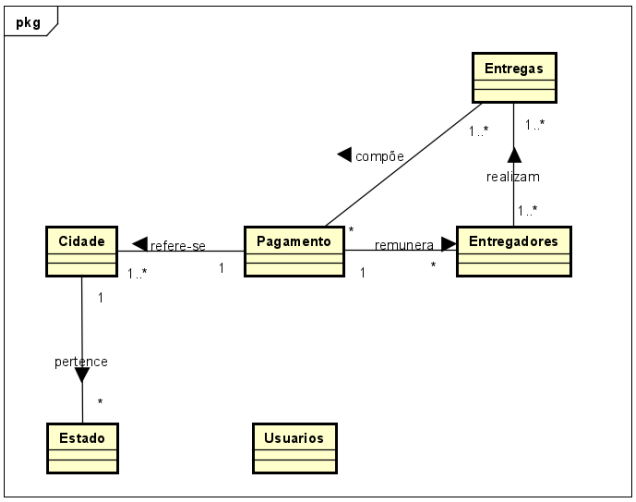

Figura 3. Diagrama de domínio.

B. Construir uma Lista de Funcionalidades

Na segunda etapa do FDD, é necessário desenvolver uma lista de funcionalidades, descrevendo os requisitos funcionais e não funcionais do sistema. Os requisitos funcionais são os serviços que o sistema deve fornecer e funcionalidades específicas que ele deve executar. Por outro lado, os requisitos não funcionais são os que não possuem relação direta com os serviços e normalmente especificam as características do sistema [21].

	REQUISITOS FUNCIONAIS
RF01	Permitir o login de usuários
RF02	Permitir o CRUD de motoristas apenas para administradores
RF03	Permitir o CRUD de cidades apenas para administradores
RF04	Permitir o CRUD de tipos de entregas apenas para administradores
RF05	Permitir a consulta de dados e inserção de registros com base no período selecionado
RF06	Permitir que os usuários informem a quantidade de entregas de cada motorista
RF07	Permitir que apenas o administrador altere o valor por entrega de um entregador
RF08	Permitir que apenas o administrador visualize os pagamentos de cada entregador
RF09	Permitir que apenas o administrador envie mensagem ao entregador com o valor a receber
RF10	Permitir que apenas o administrador copie um texto padrão para colar no sistema bancário

Tabela I
TABELA DE REQUISITOS FUNCIONAIS. ELABORADA PELO AUTOR.

	REQUISITOS NÃO FUNCIONAIS
RNF01	O sistema será implementado para uso na web
RNF02	O sistema deve ser intuitivo
RNF03	O sistema deve permitir a inserção de dados de maneira segura
RNF04	O sistema deve fornecer um feedback em até 3 segundos

Tabela II
TABELA DE REQUISITOS NÃO FUNCIONAIS. ELABORADA PELO AUTOR.

C. Planejar através de funcionalidades

A terceira etapa da metodologia consiste em planejar o desenvolvimento. Baseado na lista de funcionalidades construída, é descrita uma ordem de implementação de funcionalidades.

Na Figura 4, pode-se analisar o diagrama de atividades do projeto, utilizado para demonstrar o processo de negócio em que o software é empregado, por meio da discriminação das atividades envolvidas no fluxo [21]. Nele, é possível visualizar o principal fluxo de atividades do sistema, que consiste em facilitar e automatizar o trabalho do setor

financeiro da empresa. Com o controle dessas atividades por meio do sistema, os pagamentos tendem a ser menos suscetíveis ao erro e devem demandar menos tempo para serem realizados.

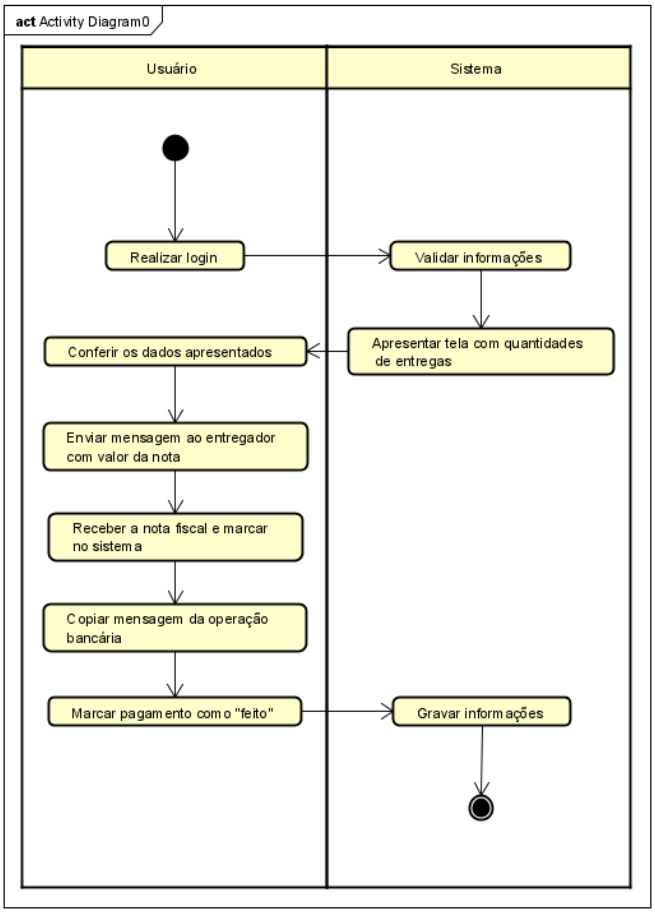

Figura 4. Diagrama de atividades.

D. Projetar por funcionalidades

A Figura 5 apresenta o Diagrama Entidade Relacionamento do software, contendo a estrutura de tabelas do banco de dados.

A entidade associativa de Pagamento_Entregas existe para vincular um tipo de entrega com o pagamento em grandes quantidades, evitando que fosse realizada por meio de incrementação. Por outro lado, a entidade associativa Entregador_Entregas tem o objetivo de estabelecer uma ligação entre um tipo de entrega e um entregador, permitindo que no *front end* a quantidade inserida já seja automaticamente vinculada com o tipo de entrega em questão. Além disso, cabe destacar que a manutenção dessa tabela diz respeito apenas ao gerente financeiro, o que visa impedir que o setor administrativo altere os valores de um entregador. Os campos de "situação", presentes em diversas tabelas, tem a funcionalidade de tratar um registro como ativo e inativo, a fim de não permitir a deleção de dados para manter o histórico do sistema inalterado. Por último, faz-se necessário observar que a entidade de Usuários não possui relação com nenhuma outra entidade no diagrama, pois eles irão gerenciar todas as tabelas.

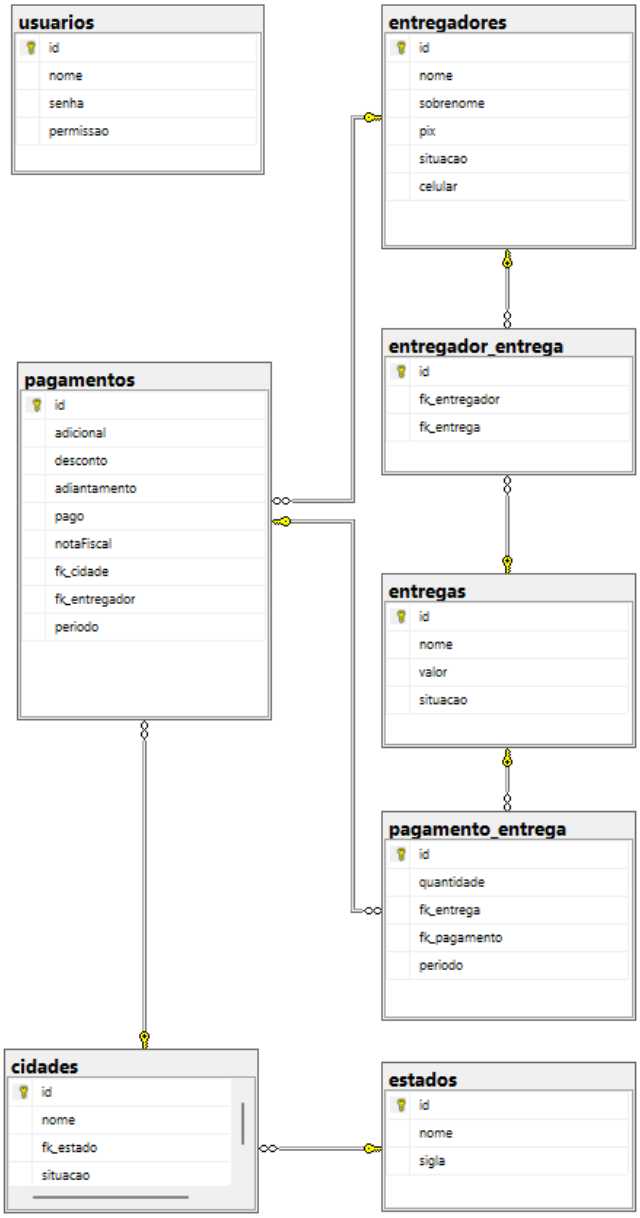

Figura 5. Diagrama Entidade Relacionamento.

Já na Figura 6, pode-se observar o Diagrama de Caso de Uso do sistema, que apresenta as funcionalidades que os usuários podem realizar. Nesse caso, a maioria das operações ficam restritas ao ator Gerente Financeiro, o que garante maior segurança dos dados monetários e ainda possibilita um sistema mais intuitivo ao setor administrativo, visto que não irão visualizar funções desnecessárias ao seu papel.

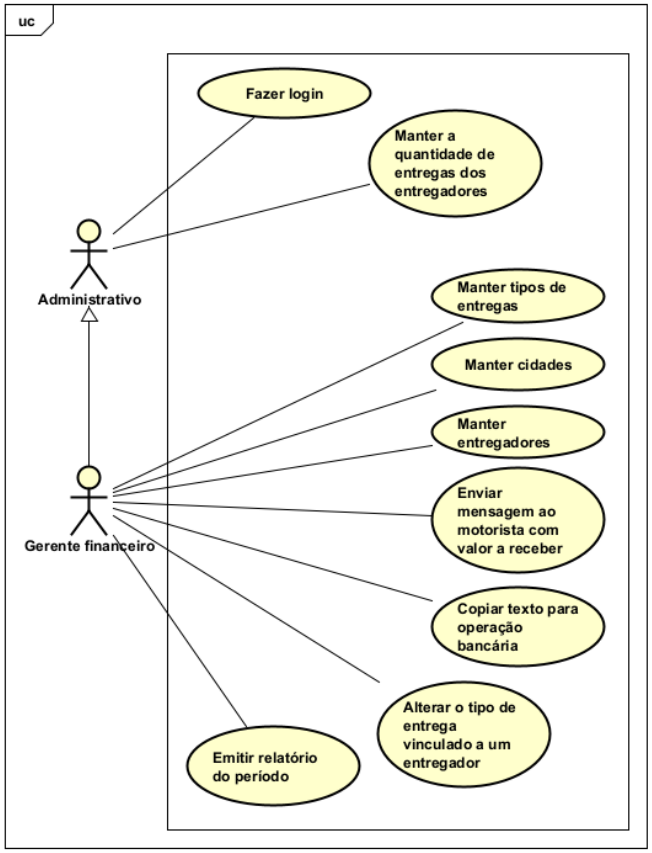

Figura 6. Diagrama de Caso de Uso.

E. Construir por funcionalidades

Na última seção do FDD, ocorre a implementação do software, e consiste no desenvolvimento de cada funcionalidade identificada nas etapas iniciais. O sistema elaborado é composto por uma API e uma interface web, ambos integrados para garantir uma experiência fluida e eficiente ao usuário final.

1) Estrutura da API: A API foi desenvolvida em ASP.NET Core, organizando o projeto em camadas de Models e Controllers, responsáveis pela estrutura dos dados e pelos pontos de entrada das requisições, respectivamente. As classes contidas da pasta Models representam as entidades do banco de dados, sendo definidas com propriedades que refletem as colunas e os relacionamentos das tabelas. Por sua vez, os Controllers atuam como intermediários, recebendo as requisições HTTP e mapeando cada funcionalidade necessária para realizar operações de criação, leitura, atualização e exclusão nas entidades, além de exigir a autenticação do usuário por meio do token JWT.

Para conectar a API ao banco de dados, foi criada uma classe de contexto do Entity Framework, que serve como ponto de acesso principal para as operações com o banco. Essa classe gerencia a comunicação com o banco e permite configurar regras de mapeamento, como chaves primárias e relacionamentos entre tabelas, promovendo uma camada de abstração entre o código e o banco de dados.

Na Figura 7, é possível visualizar uma das *endpoints* presentes no PagamentoController, em que é feita um requisição GET com base no período consultado. Além disso, na Figura 8 é exibida a organização de classes e pastas do projeto do *backend*.

```
[HttpGet]
[Route("pagamentos/{periodo}")]
0 referências
public async Task<IActionResult> getByPeriodoAsync(//consulta por periodo
    [FromServices] Contexto contexto,
    [FromRoute] DateOnly periodo)
{
    var pagamento = await contexto
        .Pagamentos
        .AsNoTracking()
        .Where(e => e.Periodo == periodo)
        .ToListAsync();

    return pagamento == null ? NotFound() : Ok(pagamento);
}
```

Figura 7. Exemplo de endpoint no projeto da API.

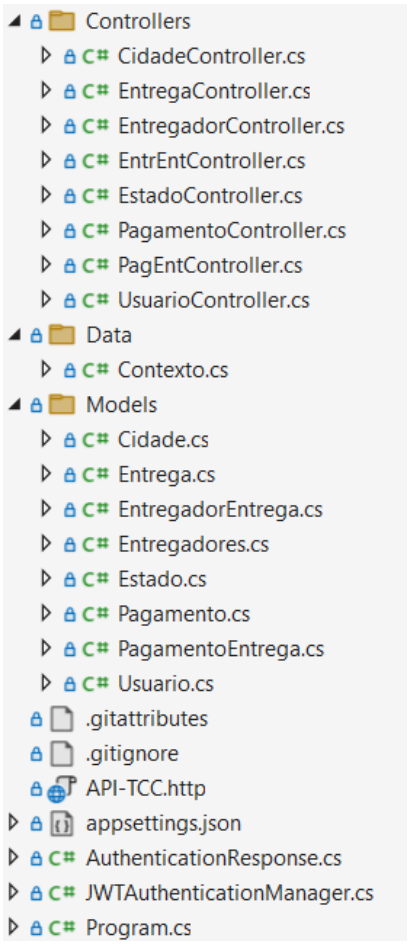

Figura 8. Estrutura do projeto da API.

2) Interface Web: A interface web foi construída em um projeto separado usando o Blazor, que consome a API desenvolvida, e torna possível a separação entre *back end* e *front end*. A estrutura do projeto Blazor é organizada em pastas que contêm componentes e serviços. As páginas, localizadas na pasta Pages, representam as diferentes telas do aplicativo, onde cada uma é responsável por uma funcionalidade específica e interação com o usuário.

Além disso, o projeto inclui uma pasta chamada Services, com três classes de gerenciamento de serviços. Uma classe gerencia a comunicação com a API, enquanto outra contém métodos dedicados à autorização e autenticação de usuários, como login, logout e geração do token. Também está presente um serviço específico para o WhatsApp, que facilita o envio de mensagens, permitindo a automação da comunicação com os usuários. Essa organização em camadas melhora a manutenibilidade do código e também proporciona uma arquitetura mais limpa e modular, essencial para o desenvolvimento de aplicações escaláveis.

A Figura 9 mostra a construção da função LoadPagamentos, que carrega os pagamentos utilizando a data selecionada como filtro. A partir dos dados obtidos, é feita uma formatação para apresentar os números de maneira mais intuitiva e calculado os valores bruto e líquido. Adicionalmente, a Figura 10 apresenta a maneira que foi estruturada o *front end*, com as páginas e os serviços.

```
private async Task LoadPagamentos()
{
    try
    {
        DateTime data;
        if (filtroPagamento.Quinzena == 1)
        {
            data = new DateTime(filtroPagamento.Ano, filtroPagamento.Mes, 1);
        }
        else
        {
            data = new DateTime(filtroPagamento.Ano, filtroPagamento.Mes, 16);
        }

        pagamentos = await ApiService.GetDataAsync<List<Pagamento>>($"api/Pagamento/pagamentos/{data:yyyy-MM-dd}");

        foreach (var pagamento in pagamentos)
        {
            if (pagamento.Adiantamento.HasValue)
            {
                pagamento.Adiantamento = decimal.Round(pagamento.Adiantamento.Value, 2);
            }
            if (pagamento.Adicional.HasValue)
            {
                pagamento.Adicional = decimal.Round(pagamento.Adicional.Value, 2);
            }
            if (pagamento.Desconto.HasValue)
            {
                pagamento.Desconto = decimal.Round(pagamento.Desconto.Value, 2);
            }
            var entregador = entregadores.FirstOrDefault(e => e.Id == pagamento.FkEntregador);
            if (entregador != null)
            {
                pagamento.ValorBruto = await CalcularValorBrutoAsync(entregador.Id, pagamento.Periodo);
                UpdateValorLiquido(pagamento, entregador);
            }
        }
    }
    catch (Exception)
    {
        ShowAlert("Erro ao carregar pagamentos", Color.Danger);
    }
}
```

Figura 9. Função para carregar os pagamentos.

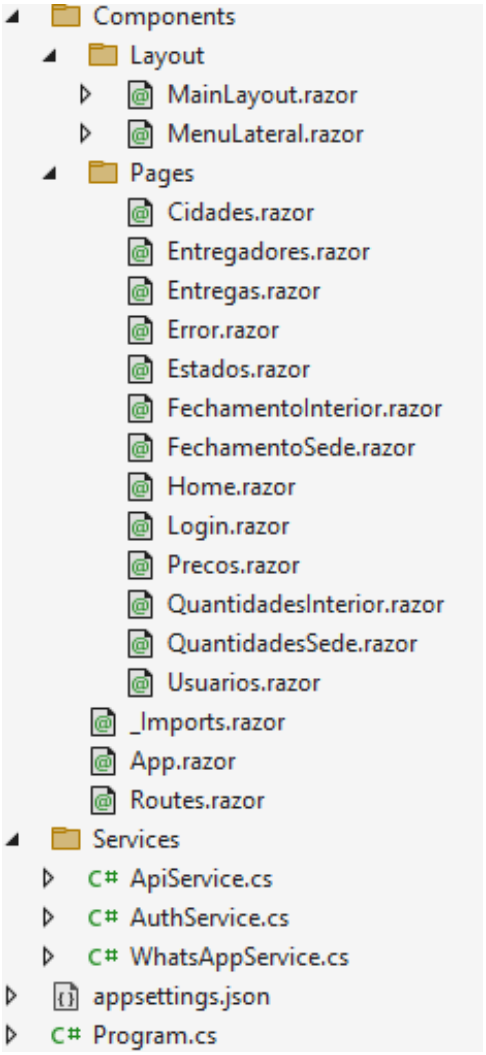

Figura 10. Estrutura do projeto Blazor.

IV. Resultados Obtidos

Como resultado do presente trabalho, foi desenvolvido um sistema web que permite a gestão de pagamentos dos entregadores de uma empresa de logística. Além disso, o software possibilita a integração de todos os dados pertinentes ao gerenciamento de motoristas e da abrangência atendida pela organização.

A Figura 11 exibe todos os entregadores cadastrados, sejam ativos ou inativos. Também permite as ações de cadastrar um novo entregador, alterar os dados de entregador já existente e inativar um registro. Já a Figura 12, expõe a organização da tela de gestão de pagamentos, que proporciona o lançamento de descontos e adicional, além das ações de envio do valor para o entregador, marcar como "recebida"a nota fiscal, copiar a chave pix e marcar como "pago"o registro, que estiliza a linha com a cor amarela. Os dados são apresentados com base no período consultado.

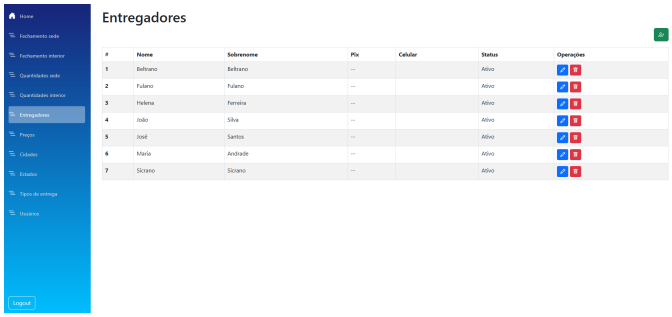

Figura 11. Tela de gestão dos entregadores.

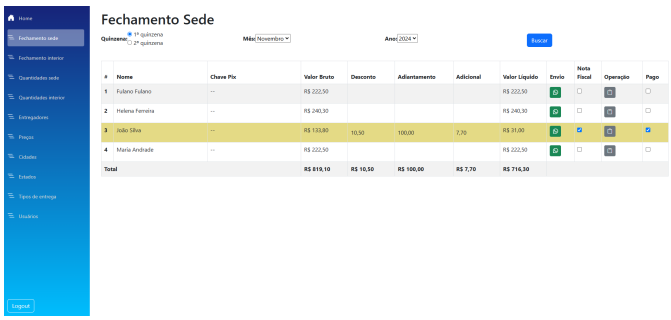

Figura 12. Tela de gestão dos pagamentos.

Ademais, o software lida com duas permissões de usuário. A permissão "1"é atribuída aos gestores, oferecendo total controle sobre o sistema, enquanto a permissão "2"é para usuários administrativos, com acesso restrito a funcionalidades de cadastro e alteração de dados, conforme pode ser verificado no Diagrama de Caso de Uso da Figura 6. A Figura 13 demonstra a tela inicial de um usuário com a permissão inferior, apresentando apenas três opções no menu lateral.

Figura 13. Tela inicial de usuário com permissão inferior.

V. Considerações finais

O desenvolvimento do sistema web para o gerenciamento de pagamentos a entregadores de uma empresa de logística de e-commerce alcançou os objetivos propostos, promovendo um controle mais eficiente dos dados e facilitando o processo de trabalho. Após sua implantação na empresa, o sistema apresentou resultados positivos, como a facilitação no controle de pagamentos, a agilização de tarefas operacionais e a melhoria da transparência dos valores aos entregadores. A implementação da solução mostrou-se eficaz para otimizar os processos e aumentar a eficiência operacional, comprovando a relevância das tecnologias no enfrentamento dos desafios do setor.

A metodologia aplicada permitiu não apenas uma análise detalhada dos requisitos, mas também o desenvolvimento de diagramas e protótipos que orientaram o desenvolvimento do software tanto no *back end* quanto no *front end*. O sistema foi construído com uma interface intuitiva e funcionalidades robustas, garantindo que os requisitos levantados fossem integralmente atendidos.

As contribuições deste sistema vão além do contexto específico para o qual foi desenvolvido, podendo servir como modelo para outras organizações que realizam pagamentos periódicos com base na produtividade. O presente trabalho reforça a importância de soluções integradas no contexto da logística moderna, promovendo uma maior eficiência e agilidade nas operações de pagamento e em todos os aspectos pertinentes.

Referências

[1] Gustavo de Sousa Zimmermann e Paulo César dos Santos. “AGROPAY: inovação na gestão de pagamentos de funcionários rurais”. Em: 15.3 (2023).

[2] Paulo Nazário. “A importância de sistemas de informação para a competitividade logística”. Em: *Revista Tecnologística, São Paulo, ano* 5 (1999), p. 31.

[3] Vipin Jain, Bindoo Malviya e Satyendra Arya. “An overview of electronic commerce (e-Commerce)”. Em: *Journal of Contemporary Issues in Business and Government* 27.3 (2021), pp. 665–670.

[4] Stanley Frederick WT Lim, Xin Jin e Jagjit Singh Srai. “Consumer-driven e-commerce: A literature review, design framework, and research agenda on last-mile logistics models”. Em: *International Journal of Physical Distribution & Logistics Management* 48.3 (2018), pp. 308–332.

[5] OptimoRoute. “A Guide to Middle Mile Logistics”. Em: (2022). Disponível em <https://optimoroute.com/middle-mile-logistics/>.

[6] Ralph M Stair et al. *Princípios de sistemas de informação*. Cengage Learning, 2023.

[7] Kenneth Craig Laudon e Jane Price Laudon. *Sistemas de informação gerenciais: administrando a empresa digital*. Pearson, 2004.

[8] GitHub Inc. “Sobre o GitHub e o Git”. Em: (2024). Disponível em <https://docs.github.com/pt/get-started/start-your-journey/about-github-and-git>.

[9] Microsoft. “Documentação do .NET”. Em: (2024). Disponível em <https://learn.microsoft.com/pt-br/dotnet/>.

[10] Microsoft. “Documentação do ASP.NET”. Em: (2024). Disponível em <https://learn.microsoft.com/pt-br/aspnet/core/?view=aspnetcore-8.0>.

[11] Microsoft. “Documentação do Blazor”. Em: (2024). Disponível em <https://dotnet.microsoft.com/pt-br/apps/aspnet/web-apps/blazor>.
[12] Microsoft. “Entity Framework”. Em: (2024). Disponível em <https://learn.microsoft.com/pt-br/ef/ef6/>.
[13] Christian Bauer. *Hibernate in action*. 2005.
[14] Microsoft. “Documentação do Microsoft SQL”. Em: (2024). Disponível em <https://learn.microsoft.com/pt-br/sql/?view=sql-server-ver16>.
[15] Amazon. “O que é uma API?” Em: (2023). Disponível em <https://aws.amazon.com/pt/what-is/api/>.
[16] Ana Isabel Alves Marques. “Desenvolvimento de API para aplicação cloud”. Tese de dout. 2018.
[17] JWT. “Documentação do JWT”. Em: (2024). Disponível em <https://jwt.io/introduction>.
[18] FG Silva, Sandra CP Hoentsch e Leila Silva. “Uma análise das Metodologias Ágeis FDD e Scrum sob a Perspectiva do Modelo de Qualidade MPS. BR”. Em: *Scientia Plena* 6.3 (2010).
[19] Jesper Boeg. “Kanban em 10 passos”. Em: *C4Media* (2010), p. 27.
[20] Rhuan Matheus Drescher. “Desenvolvimento de sistema de ouvidoria”. Em: (2019).
[21] Ian Sommerville. *Engenharia de software*. Pearson, 2018.