Integração de Transferência Eletrônica de Fundos em PDV Web: Uma Abordagem com ASP.NET, Vue.js e WebSocket[1]

Tiago P. Moroni[2], Fabieli De Conti[3]

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Farroupilha 95.174-274 – Farroupilha – RS – Brasil

tiagopmoroni@gmail.com, fabieli.conti@farroupilha.ifrs.edu.br

Abstract. *This study details the integration of a Fund Transfer System (TEF) into a web-hosted point of sale (POS) software used by more than a hundred clients. The motivation arose from the need to comply with state regulations in Rio Grande do Sul regarding the mandatory use of TEF. The structure was created with a Dynamic Link Library (DLL) responsible for communication, through WebSocket, with the integrated TEF House client (Destaxa), enabling card and PIX payments on browser-accessible platforms.*

Resumo. *Este estudo detalha a integração de um sistema de Transferência Eletrônica de Fundos (TEF) em um software de ponto de venda (PDV) hospedado na web, utilizado por mais de 100 clientes. A motivação surgiu da necessidade de cumprir as normas estaduais do Rio Grande do Sul referentes ao uso obrigatório do TEF. A estrutura foi criada com uma Dynamic Link Library (DLL) encarregada da comunicação, através de WebSocket, com o Client da TEF House integrada (Destaxa), viabilizando pagamentos com cartão e PIX em plataformas acessadas via browser.*

Palavras-chave: TEF. PDV Web. Integração de Sistemas. WebSocket. ASP.NET.

1. Introdução

Com o varejo evoluindo e se modernizando, é necessário cada vez mais agilidade e segurança nos seus meios de pagamento integrados. No quesito financeiro e fiscal, os sistemas de PDV cresceram e se tornaram ferramentas indispensáveis para empreendedores e empresas das mais diversas áreas, com diferentes portes. A constante busca de integrar soluções de pagamento eletrônico, como a TEF, vem se tornando uma prioridade, ainda mais em locais onde essa prática se tornou obrigatória por lei.

No estado do Rio Grande do Sul, desde o ano de 2024, a legislação determina o uso do TEF em estabelecimentos que emitem documentos fiscais eletrônicos, o que impulsionou o desenvolvimento de soluções compatíveis com essas exigências (BRASIL 61, 2023).

---

[1] Artigo científico referente ao Trabalho de Conclusão de Curso do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal do Rio Grande do Sul – Campus Farroupilha.

[2] Aluno matriculado no Trabalho de Conclusão de Curso.

[3] Professora orientadora do Trabalho de Conclusão de Curso.

Entretanto, a implementação do TEF em arquiteturas web ainda representa um desafio técnico muitas vezes, visto que a maioria das soluções disponíveis no mercado são voltadas para aplicações desktop, o que entra em conflito diretamente com infraestruturas em nuvem, arquitetura proposta neste projeto. A escassez de alternativas compatíveis com PDVs totalmente baseados em navegador cria uma limitação na usabilidade de alguns sistemas comerciais, especialmente em ambientes onde os terminais operam conectados diretamente à nuvem.

Neste contexto, propõe-se o desenvolvimento de uma solução de integração TEF para um sistema PDV com arquitetura web, atualmente utilizado por mais de 100 clientes. A proposta consiste na criação de uma DLL responsável por estabelecer a comunicação entre o terminal e o *client* da *TEF House* Destaxa. Para a integração, foi utilizado o WebSocket como protocolo de troca de mensagens e a implementação foi realizada com tecnologias modernas, como ASP.NET no *backend* e Vue.js no *frontend*, garantindo compatibilidade com navegadores e permitindo a execução de transações com cartões e PIX diretamente pelo site.

A principal diferença do projeto desenvolvido para outras soluções de pagamentos disponíveis é a integração completa no sistema. A realidade em outras soluções é uma integração que ocorre por fora do sistema com redirecionamento para sites externos, algo que não se encaixa nos requisitos impostos para a demanda. Além de resultar em um fluxo mais fácil e intuitivo para o usuário final, a DLL permite a reutilização de código em futuros projetos com demandas similares.

Este projeto tem como objetivo desenvolver uma solução robusta e que esteja de acordo com as necessidades reais dos clientes que operam na arquitetura em nuvem. O projeto abordará não apenas as questões técnicas da implementação, mas também os obstáculos e problemas enfrentados no desenvolvimento do mesmo. Serão relatados também os critérios adotados nos testes e os resultados obtidos após a adoção dos clientes em situações reais de uso.

2. Referencial Teórico

O referencial teórico deste trabalho aborda seis tópicos principais, que são: evolução dos sistemas de pagamento, TEF, integração de TEF em ambientes web, comunicação em tempo real com WebSocket, estudos e pesquisas anteriores relacionados ao tema, além de DLL, que aborda uma das tecnologias utilizadas no projeto.

2.1. Evolução dos Sistemas de Pagamento

A consolidação do cartão de crédito no Brasil começou na década de 1980, a partir do momento em que as tarjas magnéticas passaram a permitir transações eletrônicas sem a necessidade de grande infraestrutura física, permitindo que mais consumidores utilizassem esse meio de pagamento com segurança. Nos anos 1990, os terminais chamados de PDV, passaram a integrar hardware e software de forma padronizada, permitindo que empresas de menor porte tivessem controle em tempo real das vendas realizadas, sejam elas *online* ou presenciais, resultando na redução de erros manuais,

aumentando assim a confiança no processamento automático de transações (FREITAS, 2017).

Em 2002, o Banco Central do Brasil promoveu uma reformulação profunda no Sistema de Pagamentos Brasileiro (SPB) ao implantar o Sistema de Transferência de Reservas (STR), que passou a liquidar entre diversos bancos simultaneamente as obrigações em tempo real, minimizando riscos de liquidez e crédito e estabelecendo bases sólidas para o crescimento dos pagamentos eletrônicos (BERTOLDI; TRICHES, 2006).

Com o avanço da internet na década de 2010, surgiram as carteiras digitais — *apps* que reúnem múltiplos instrumentos de pagamento em um só lugar. Soluções como Nubank, Mercado Pago e PicPay conquistaram usuários ao oferecer conveniência, programas de vantagens e interface amigável, elevando o patamar de competição no mercado financeiro (IUPANA, 2025; PISMO, 2024).

O lançamento do Pix, em novembro de 2020, marcou o salto para pagamentos instantâneos: disponível 24 horas por dia, sete dias por semana, sem custo para pessoas físicas, o sistema alcançou rapidamente volumes maiores que os cartões de débito e crédito, mudando a rotina de consumidores e varejistas ao permitir transferências em segundos.

2.2. TEF

A TEF é o sistema que automatiza o processamento de pagamentos com cartão de débito e crédito, conectando diretamente o software de gestão do PDV à operadora de cartões. Diferente dos terminais PDV tradicionais, que exigem digitação manual de valores, o TEF envia a requisição de pagamento e recebe o status da transação de forma quase instantânea e integrado ao sistema de escolha, elevando a precisão e a agilidade no atendimento (ENOTAS, 2022).

No Rio Grande do Sul, com a chegada da obrigação de integrar a Nota Fiscal de Consumidor Eletrônica (NFC-e) ao TEF, em vigor desde janeiro de 2024, foi necessário a adoção dessa tecnologia para não só garantir a conformidade fiscal, mas também para evitar possíveis multas. Nesse contexto, qualquer estabelecimento ao emitir NFC-e, precisou migrar para o sistema independentemente do porte. (FAZENDA RS, 2023).

Arquiteturalmente, uma solução TEF em ambiente web costuma envolver uma DLL instalada no servidor ou na máquina local, responsável por traduzir comandos do PDV em mensagens padronizadas para o *client* da *TEF House* (Destaxa). No caso desta pesquisa, a comunicação entre essa DLL e o *client* usa WebSocket, mantendo a conexão aberta para troca de mensagens em tempo real e assegurando baixa latência nas autorizações (DESTAXA, 2024).

Entre os principais benefícios do TEF estão a conciliação bancária automática, que elimina lançamentos manuais e reduz erros de conferência, e a geração de *logs* detalhados de cada transação, facilitando auditorias e a detecção de fraudes. Essas vantagens resultam em maior controle financeiro e em uma experiência de compra mais fluida para o cliente (INFOVAREJO, 2024).

2.3. Integração de TEF em Ambientes Web

Para colocar o TEF a um PDV que roda 100% em navegador, é comum adotar uma solução *API-first*, em que toda a comunicação com a operadora de cartões se faz por chamadas *REST* e WebSocket para um serviço local (WebTEF) ou em nuvem (Mercado Pago, PayGo, etc.). Nesse modelo, o *frontend* consome endpoints que expõem operações de autorização, captura e conciliação, sem depender de plugins proprietários no *browser*.

A infraestrutura dessa integração pode ser uma DLL ou serviço Windows instalado no servidor ou na máquina do PDV, que atua como intermediário entre o navegador e o *client TEF House* (Destaxa). No presente estudo, o componente traduzirá requisições *HTTP* em mensagens padronizadas para o *client* via WebSocket, mantendo a conexão persistente para receber notificações de autorização em tempo real (DESTAXA, 2024).

Entretanto, como o *browser* não consegue se comunicar diretamente com dispositivos de pagamento (PIN Pads), é necessário um canal de comunicação local. Uma abordagem comum é executar um pequeno serviço TCP/WebSocket na mesma rede ou máquina, que a DLL acessa para encaminhar comandos ao hardware e retornar resultados ao *frontend*.

Para garantir uma performance sólida e confiável, recomenda-se implementar *retry logic*, filas locais para transações *offline* e *logs* centralizados. Além disso, arquiteturas multiadquirentes[4] podem ser orquestradas pela DLL, permitindo que o lojista escolha dinamicamente a melhor taxa e mitigando riscos de indisponibilidade de um único provedor financeiro.

2.4. Comunicação em Tempo Real: WebSocket

De forma resumida, o WebSocket é o protocolo que permite manter uma única conexão TCP aberta entre cliente e servidor, suportando troca de mensagens bidirecionais e em tempo real, com muito menos processamento necessário que técnicas baseadas em conexão contínua utilizando protocolo *HTTP*, o chamado *HTTP polling* (TEXSOFT, 2025).

O funcionamento do Websockets, representado na figura 01, começa com um estabelecimento de uma conexão *HTTP*/1.1 conhecida como *handshake*, que transforma a conexão para WebSocket. Após esse passo, cliente e servidor trocam pedaços de texto ou binários sem refazer o *handshake*, mantendo a conexão viva até que qualquer dos lados a encerre (IETF, 2011).

[4] Arquiteturas multiadquirentes consistem na integração com mais de um adquirente de pagamentos, ou seja, diferentes empresas que realizam a captura e liquidação das transações com cartões.

Figura 1. Funcionamento de Websockets

Fonte: GCORE (2023)

Em comparação com *long polling e Server-Sent Events*, o WebSocket consegue reduzir drasticamente a latência de comunicação, pois não há repetidas requisições *HTTP*, e permite comunicação *full-duplex* (bidirecional), essencial quando a ordem e a rapidez das mensagens importam, como em sistemas de pagamento ou dashboards financeiros (ABLY, 2018).

Para operar de modo seguro, é obrigatório usar protocolo WebSocket (*WSS)* com conexão TLS criptografada, aplicar políticas de *Cross-Origin Resource Sharing* (CORS) restritas e validar, sempre que possível, a origem das mensagens (OWASP, 2025).

O WebSocket estabelece uma conexão persistente entre cliente e servidor, diferente do modelo tradicional de requisição-resposta do HTTP. Uma vez estabelecida, essa conexão permite a troca contínua de mensagens sem a necessidade de reabrir conexões ou reenviar cabeçalhos repetidamente, garantindo maior eficiência no tráfego de dados. Além disso, o protocolo suporta mensagens em formato binário e texto, o que facilita a transmissão de diferentes tipos de informação, desde notificações simples até dados mais complexos em tempo real.

Em integrações TEF, o WebSocket garante que o PDV web receba instantaneamente o status da autorização de pagamento pela DLL, sem atrasos ou necessidade de *polling*, proporcionando uma experiência fluida de uso e confiabilidade na comunicação com o *client* da *TEF House* (Destaxa, 2024).

2.5. Estudos e Pesquisas Anteriores

A literatura revela que, embora o TEF seja reconhecido como tecnologia estratégica no varejo, sua integração em ambientes web ainda necessita de aprimoramentos e estudos. Em um estudo que analisou dados de 27 supermercados de diversos portes, constatou-se que projetos de automação, incluindo o TEF, apresentam alto impacto operacional, mas enfrentam barreiras em estabelecimentos menores devido a custos e à complexidade de implantação (PINHEIRO DA SILVEIRA; AGUILAR, 2007).

Em fóruns de desenvolvimento, profissionais confirmam que a única forma viável de um navegador comandar um *PIN Pad* TEF é por meio de um módulo local (DLL) que atua como ponte segura via *socket*, reforçando a abordagem adotada no presente estudo (CORRÊA, 2016).

2.6. DLL

Uma DLL é um arquivo que agrupa rotinas executáveis e recursos compartilháveis, sendo carregada em tempo de execução para fornecer funcionalidades a múltiplas aplicações sem a necessidade de recompilação conjunta. Diferentemente de bibliotecas estáticas, que são incorporadas ao executável no momento da compilação, as DLLs são vinculadas apenas quando suas funções são invocadas, otimizando o uso de memória e permitindo atualizações independentes de componentes. Essa modularidade facilita a manutenção e o versionamento de sistemas complexos, pois basta substituir o arquivo DLL para corrigir ou melhorar funcionalidades específicas, sem impactar o restante da aplicação (Microsoft Learn, 2022).

Desenvolveu-se, em C#, uma DLL responsável por traduzir as solicitações de pagamento enviadas pelo *frontend* em mensagens padronizadas para o *client* da *TEF House* via WebSocket. Ela expõe uma API local consumida pelo *backend*, que também é local para cada terminal. Em cenários de instabilidade de rede, a DLL implementa lógica de *retry*, reduzindo perdas de mensagens e garantindo consistência nas transações financeiras.

3. Procedimentos Metodológicos

Esta seção apresenta os procedimentos metodológicos adotados no desenvolvimento da integração do sistema de TEF. Serão detalhadas as ferramentas e tecnologias utilizadas, o planejamento das etapas de desenvolvimento, o levantamento e análise de requisitos, bem como a modelagem das estruturas de dados e fluxos de comunicação.

3.1. Ferramentas e Tecnologias Utilizadas

A integração foi desenvolvida utilizando tecnologias já utilizadas no projeto, com o objetivo de facilitar tanto o processo quanto futuras manutenções. Além da sua conveniência, as tecnologias também atendem aos requisitos de um projeto que envolve assuntos financeiros e de pagamentos, sendo rápidas, seguras e escaláveis, aptas a atender altos volumes de requisições em curtos intervalos de tempo.

Para a camada de apresentação (*frontend*), foi escolhido o Vue.js, *framework* progressivo para construção de interfaces reativas, que garante alta performance mesmo sob grande número de componentes dinâmicos. Em conjunto, utilizou-se o Vuetify, biblioteca de componentes baseada em Material Design, que acelera o desenvolvimento de layouts consistentes e responsivos (VUEJS, 2024; VUETIFY, 2025).

O ambiente de codificação do *frontend* foi o Visual Studio Code, editor leve com extensões para Vue que fornecem *IntelliSense*, *linting* e *debugging* integrados. As chamadas *HTTP* ao *backend* são realizadas com Axios, biblioteca que simplifica o

tratamento de requisições assíncronas e oferece camadas de lógica intermediárias para gerenciar erros e tokens de segurança (MICROSOFT, 2021).

Já no servidor (*backend*), a aplicação foi implementada em C# sobre ASP .NET Core, adotando arquitetura de microsserviços para isolar domínios de pagamento, autenticação e notificação, o que facilita escalabilidade e manutenção (MICROSOFT, 2023).

O desenvolvimento e depuração ocorreram no Visual Studio, Integrated Development Environment (IDE) robusto com diversas ferramentas para acelerar e quantificar qualidade de desenvolvimento. Para acesso a dados, optou-se pelo Dapper, micro-ORM que combina mapeamento simples com execução de SQL puro, garantindo *overhead* extremamente baixo. (DAPPER, 2025)

Para armazenamento persistente de informações, foi utilizado o banco de dados MySQL, conhecido pela robustez e, por ser um banco SQL, mantém a estrutura lógica das tabelas com o uso de chaves estrangeiras para relacionamentos.

Além dessas, foram utilizadas outras ferramentas para facilitar a depuração e/ou tornar o processo de desenvolvimento mais eficiente. Para testes no *backend*, foi utilizado o Postman, que é um cliente *HTTP* para teste de APIs *REST* e WebSocket, utilizado para validar endpoints no servidor, simular cenários de falha e automatizar coleções de testes durante o ciclo de desenvolvimento (POSTMAN, 2024).

Para testes de consultas, foi utilizado tanto MySQL Workbench, como MySQL Query Browser, para validar a lógica de recuperação de informação e verificar comportamentos inesperados ou tempos de processamento acima do ideal (ORACLE, 2024).

3.2. Delimitação do Escopo de Projeto

Nesta fase, foi identificado as funcionalidades essenciais da integração TEF, os requisitos não funcionais de desempenho, segurança e escalabilidade. Todas as atividades necessárias para a conclusão do projeto foram divididas em pequenas seções, para manter um registro mais claro do progresso. O desenvolvimento iniciou pela camada de *frontend*, desenvolvendo os componentes visuais que permitem ao usuário executar as ações da integração. Em seguida, foi desenvolvida toda a arquitetura do banco de dados para a persistência das informações dos pagamentos. Por último, o desenvolvimento da API que funciona como intermediário entre o website e o *client* de pagamento da Destaxa.

3.3. Levantamento e Análise de Requisitos

Foram coletados, com os clientes e com a Destaxa, os requisitos funcionais e não funcionais do sistema. Também foi observada a documentação da *TEF House*, que especifica como a troca de mensagens WebSocket deve ser modelada para cada ação e define alguns requisitos adicionais para a integração. Segue abaixo, no quadro 01, a lista de requisitos seguida em todo o período do projeto:

Quadro 1. Requisitos Funcionais do Sistema

Requisitos Funcionais	
RF1	Sistema deve possibilitar meio de pagamento PIX
RF2	Sistema deve possibilitar meio de pagamento cartão de crédito/débito
RF3	Todo processo da transação deve acontecer em, no máximo, um minuto
RF4	Sistema deve armazenar e comunicar ao *client* da Destaxa o número sequencial em cada ação com a integração
RF5	Sistema deve possuir uma interface dedicada para visualizar e estornar/confirmar transações pendentes
RF6	Sistema deve mostrar comprovante ao cliente para que, só após, a transação seja confirmada
RF7	Sistema deve possibilitar reimprimir comprovante de pagamento
RF8	Sistema deve possibilitar informar CPF diretamente no *PIN Pad*
RF9	Sistema deve possuir tela de configuração TEF, definindo parâmetros como tempo máximo para pagamento PIX
RF10	Sistema deve permitir que o usuário crie múltiplas parcelas com pagamento TEF em uma venda, cada uma dessas podendo ser com um método de pagamento diferente
RF11	Sistema deve barrar a venda de ser concluída enquanto todas as parcelas não tiverem sido pagas
RF12	Enquanto a venda não tiver sido concluída e o comprovante impresso, as transações em todas as parcelas da venda devem se manter pendentes/sem confirmação
RF13	Sistema deve guiar visualmente o usuário durante o processo, dando *feedback* constante da situação da ação que estiver executando, seja ela um pagamento, um estorno, entre outras

Fonte: Autoria própria

3.4. Modelagem

Com o propósito de documentar o projeto e guiar o desenvolvimento, foi desenvolvido um fluxograma utilizando a ferramenta Miro, plataforma *online* com diversos *templates* de áreas de trabalho para colaboração em equipes. Paralelamente, foi planejado as tabelas do banco, as suas relações utilizando chaves estrangeiras e colunas que seriam criadas para armazenar os dados necessários para que a integração acontecesse de forma fluída e regularizada com as demandas da Destaxa. A modelagem foi definida em seções, para setorizar o planejamento em grupo de informações.

Observa-se que, a partir deste ponto, nomes de tabelas e colunas serão referenciados ao longo do desenvolvimento e nota-se que todos os nomes de tabelas foram padronizados com no máximo três letras, enquanto de colunas vão ter dez letras, isso se dá em razão de o sistema PDV no qual o TEF foi integrado estar amplamente baseado em uma aplicação legada, que ainda é mantida pela empresa. Para que o desenvolvedor possa discernir e se informar para além desses títulos sucintos, é possível consultar diversos dados sobre esses elementos em um dicionário interno na empresa, que armazena título, tamanho máximo, tipo de dado armazenado, entre outras informações de colunas e tabelas. Para documentar as relações e cardinalidades dos dados utilizados durante o projeto, foi criado um diagrama relacional em complemento com um Diagrama Entidade Relacionamento (DER) entre as principais tabelas que envolvem o fluxo de transferência de fundos, podendo ser observado na figura 02.

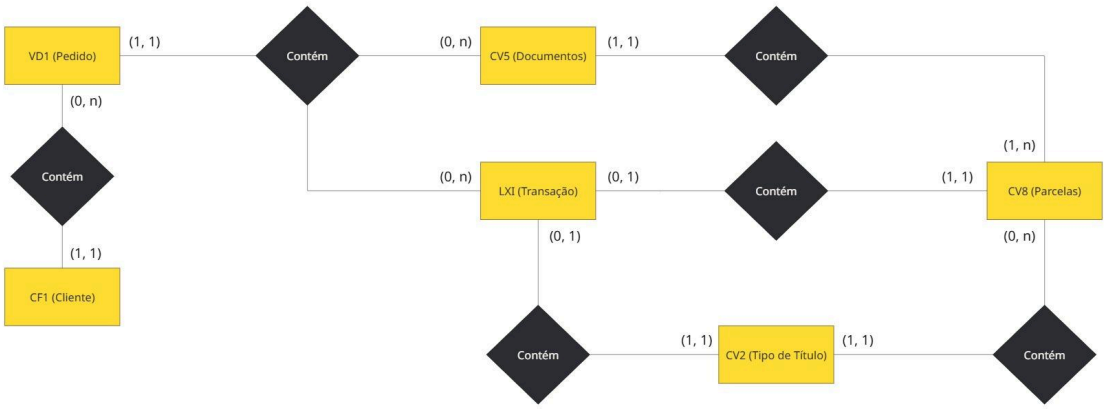

Figura 2. DER de Tabelas do TEF

Fonte: Autoria própria

O dicionário, citado acima, consiste em uma tabela armazenada de forma persistente cujo responsabilidade é armazenar informações descritivas e funcionais de todos os campos do banco de dados. Esse padrão de dados têm dois propósitos, o primeiro é compensar a limitação de dez letras nas nomenclaturas, guardando a informação de nome e descrição detalhada sobre um campo. Já a segunda aplicação para essa tabela é armazenar informações de comportamento do campo, como tamanho, tipo, máscara e obrigatoriedade. Esses parâmetros podem ser diferentes para cada cliente e o aplicativo agirá de acordo com esse dicionário, tornando todo o sistema altamente dinâmico.

A primeira seção criada no fluxograma documentou as tabelas principais que seriam utilizadas para a integração. Uma delas é a tabela LXI, cujo título é “Transação TEF”, ela será responsável por armazenar todos os pagamentos feitos no sistema. Algumas de suas colunas podem ser vistas na figura 03, e as mais relevantes são seu ID (LXITRANSAC), o id do pedido em que ocorreu a transação (LXIPEDIDO), a via do estabelecimento (LXICOMPV1) e a do cliente (LXICOMPV2), o valor da transação (LXIVALOR) e o tipo de operação, indicando se é cartão ou PIX (LXITRAPRO).

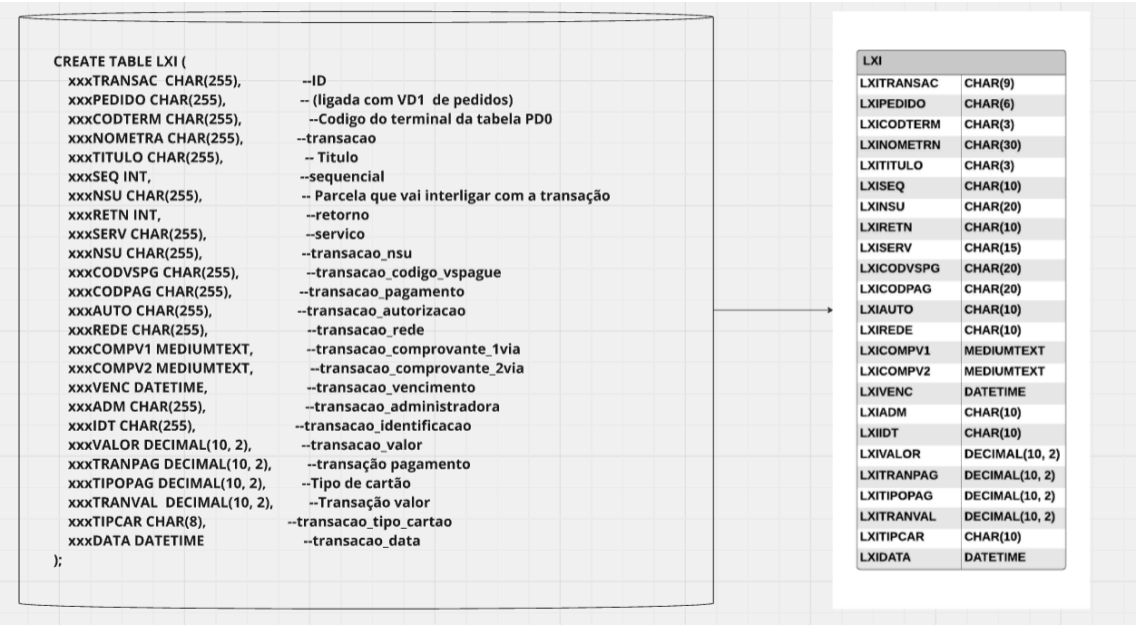

Figura 3. Estrutura da Tabela LXI

Fonte: Autoria própria

Para a segunda tabela de relevância, foi dado a ela o nome de LXH, no dicionário ela pode ser identificada como “Parâmetros de Integração do TEF”, seus principais campos são seu ID (LXHPADRAO), a descrição da integração (LXHNOME), o tempo máximo para pagamentos PIX (LXHTMPTPIX) e alguns campos para que a integração seja identificada pelo *client* Destaxa (LXHUSERINT, LXHSENHINT). Essa tela, como o nome deixa claro, é uma tabela para definir alguns parâmetros adicionais, e só poderá ser manipulado pela equipe de suporte da empresa responsável pelo projeto. O seu modelo de dados é representado na figura 04.

LXH	
LXHPADRAO	CHAR(1)
LXHNOME	CHAR(14)
LXHPCLITRM	DECIMAL(5,0)
LXHNUMCFVD	DECIMAL(1,0)
LXHUSERINT	CHAR(50)
LXHSENHINT	CHAR(50)
LXHTMPTPIX	DECIMAL(10,0)
LXHTMPVERT	DECIMAL(10,0)

Figura 4. Estrutura da Tabela LXH

Fonte: Elaborada pelo autor

A parte seguinte do fluxograma é composta por quatro seções, horizontalmente paralelas, cada uma representando uma das camadas da integração a ser desenvolvida. As camadas são website, responsável pela apresentação visual ao usuário final, API principal, responsável pela lógica principal e armazenamento de informações no banco de dados, DLL, que servirá para enviar as mensagens WebSocket para o provedor TEF e, por último, a camada do *client* Destaxa, que ouvirá as mensagens enviadas pelo PDV.

O primeiro fluxo inicia da camada de *frontend*, quando o operador do terminal informa e clica para pagar uma parcela integrada, nesse momento, o sistema identifica o tipo de transação (PIX/cartão) e envia para a DLL essa informação, junto com valor e número de pedido, como indicado no fluxo da figura 05.

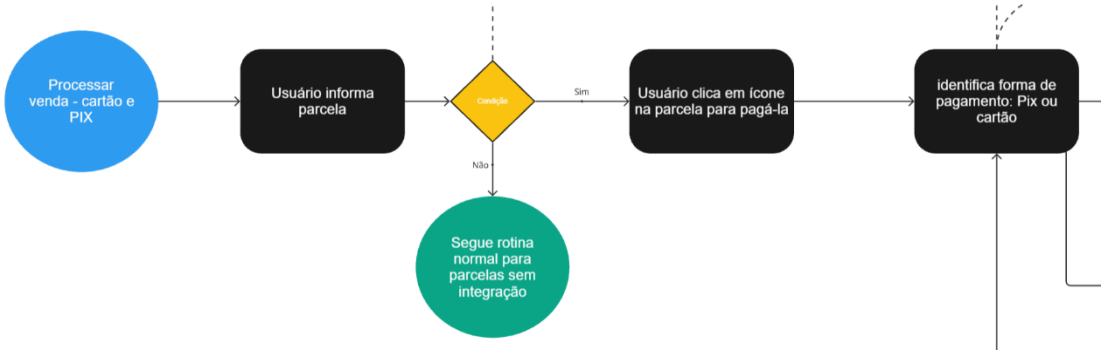

Figura 5. Fluxo Inicial de uma Transação

Fonte: Elaborada pelo autor

No Caso de uma transação PIX, o website irá enviar para se comunicar com a API local do terminal através da rota chamada "*CheckPixTransaction*", que irá verificar se existe alguma transação PIX em andamento, senão, irá gerar um código QR novo

para pagamento, através de informações como sequencial e valor da transação. Após a geração, o *frontend* envia novamente a mesma requisição, que só retorna quando o PIX estiver pago ou o tempo limite da transação expirar, respeitando o fluxo previsto na figura 06.

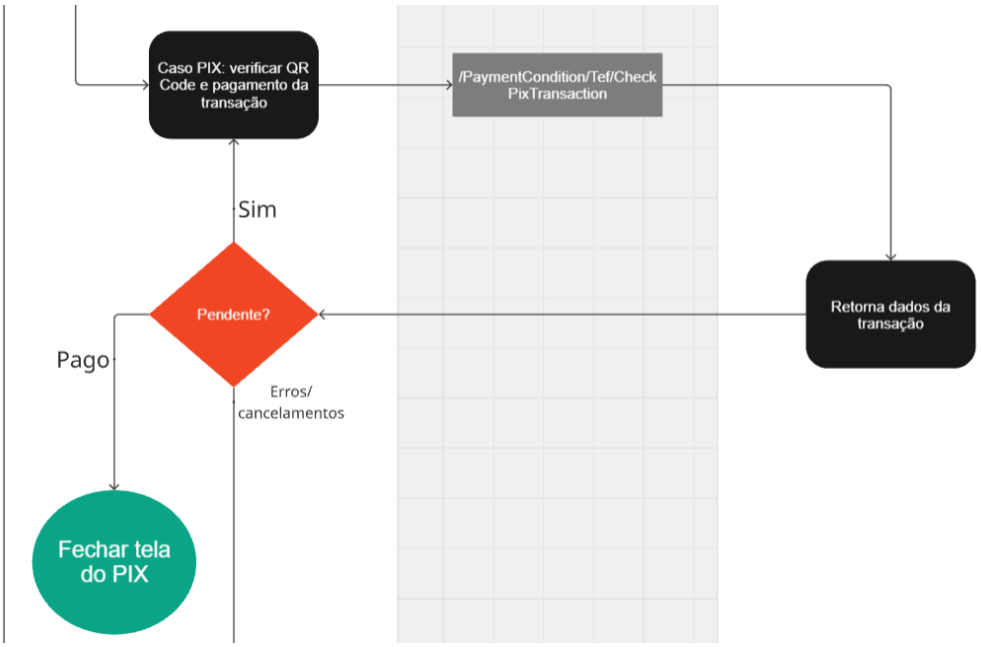

Figura 6. Fluxo de Transação Tipo PIX

Fonte: Elaborada pelo autor

Mais informações sobre processos e fluxos serão abordadas no tópico de desenvolvimento. A criação do protótipo de todos os fluxos e esquemas de dados contribuiu para a conclusão do projeto e para as futuras manutenções, pois servirá como auxílio na identificação e resolução de problemas ou bugs que venham a ocorrer.

4. Desenvolvimento

A presente seção tem como objetivo descrever o processo de desenvolvimento, aprofundando as etapas importantes e destacando o processo de raciocínio utilizado durante a criação de meios de segurança e controle da integração, componentes visuais, API para comunicação e a criação da DLL.

4.1. Feature Flag

Para iniciar qualquer funcionalidade nova que será implementada no cliente que já possui sistema, é amplamente indicado adotar a técnica de Feature Flag. Essa técnica é implementada por meio de condicionais no código que verifica o status da flag para determinar se uma funcionalidade deve ser executada ou não. Essa abordagem oferece

flexibilidade para lançar novas funcionalidades de forma controlada, permitindo testes em ambientes de produção com um grupo isolado de usuários, além de possibilitar a reversão rápida em caso de problemas (FOWLER, 2017).

No caso deste trabalho, foi criada uma Feature Flag indicando se a instância do sistema teria as funcionalidades da nova integração TEF. O campo foi adicionado como caixa de seleção (*checkbox*) na tela de parâmetros gerais do sistema, que já existia e é responsável por manter o estado dessa e de outras *Features Flags*. Caso o usuário opte por ativar a integração, será disponibilizado em seu sistema as ações descritas no trabalho até agora, como pagamento em PIX e cartão, e possibilidade de informar CPF/CNPJ nas vendas.

4.2. Componentes visuais

O próximo passo da integração, ainda no *frontend*, foi a criação dos componentes de processamento de pagamentos, estorno e manutenção de transações pendentes.

4.2.1. Processamento de pagamento

Na tela da venda do sistema, o usuário pode informar parcelas das mais diversas formas de pagamento (dinheiro, crediário, cheque, boleto, entre outros) e, para o projeto, é necessário haver a possibilidade de pagar as parcelas que fossem do tipo de pagamento integrado, PIX ou cartão de débito/crédito. Com esse propósito, foi adicionado em cima das parcelas já existentes, um ícone disponível exclusivamente para esse caso, visível na figura 07. Quando clicado, os dados da venda em andamento são persistidos no banco de dados e é iniciado todo o fluxo de pagamento.

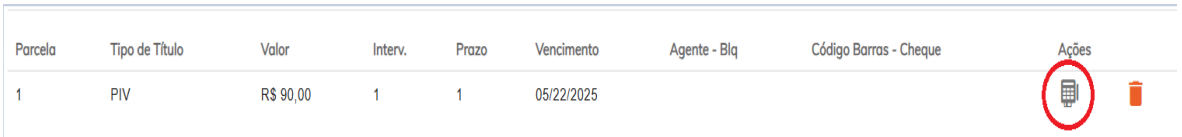

Figura 7. Ícone Para Iniciar Fluxo de Pagamento

Fonte: Elaborada pelo autor

Para que o usuário seja guiado durante o pagamento, foi implementado o componente OryxCartPaymentProcessing, responsável por exibir na tela as principais informações, como valor da parcela, número de parcelas, valor total somado das parcelas, e a instrução do que o usuário deve fazer em seguida. A interface desse componente, apresentada na figura 08, vai ser alterada dinamicamente, no caso de PIX, será apresentado em tela o código QR e o tempo restante para o pagamento.

Figura 8. Pagamento PIX

Fonte: Elaborada pelo autor

Quando expirado, será disponibilizada a opção para solicitar um novo código, como visto na figura 09.

Figura 9. Visualização Ao Expirar PIX

Fonte: Elaborada pelo autor

Pagamentos PIX são solicitados à API local através da função "*GeneratePixQrCode*", que enviará uma nova requisição para o *client* Destaxa do terminal. Esse fluxo retorna para o *frontend* uma imagem codificada em Base64. A codificação Base64 converte dados binários (como a imagem do QR Code) em uma sequência de caracteres de texto *ASCII*, garantindo que possam ser transmitidos de forma segura em formatos de texto (como *JSON* ou *HTML*) sem corromper as informações originais. Essa imagem representa graficamente os dados da transação, permitindo que o cliente final a escaneie com o aplicativo do seu banco para efetuar o pagamento. No *frontend*, essa string é decodificada e renderizada dinamicamente por meio de um componente de imagem do *framework* Vuetify, que interpreta a string e a exibe como uma imagem visível ao usuário. Esse processo elimina a necessidade de geração ou armazenamento de arquivos de imagem no servidor, utilizando o formato Base64 como meio eficiente e seguro de transferência de dados visuais entre *backend* e *frontend*. Na figura 10, é apresentado o código desenvolvido para implementar a lógica do componente.

```
<v-img :src="qrCodeSrc" height="200px" width="200px">
  <div v-if="isPixTimerExpired">
    <div
      @click="generateNewPixQrCode"
      <div>
        <v-icon color="white">mdi-refresh</v-icon>
        <p class="mb-0">Gerar novo código PIX</p>
      </div>
    </div>
  </div>
  <input
    v-on:focus="$event.target.select()"
    ref="qrCodeCopyPaste"
    readonly
    :value="codigoPix"
  />
</v-img>
```

Figura 10. Renderização De Imagem De Código QR

Fonte: Elaborada pelo autor

No caso de pagamento com cartão, a parcela integrada já contém o tipo de pagamento (débito/crédito), quantidade de parcelamento e data de pagamento individual. No momento que o usuário acionar o pagamento, o componente OryxCartPaymentProcessing passa a exibir essas informações, juntamente com um subcomponente, chamado de *infinite loading* no Vuetify, mostrando que o sistema está em estado de processamento, conforme figura 11.

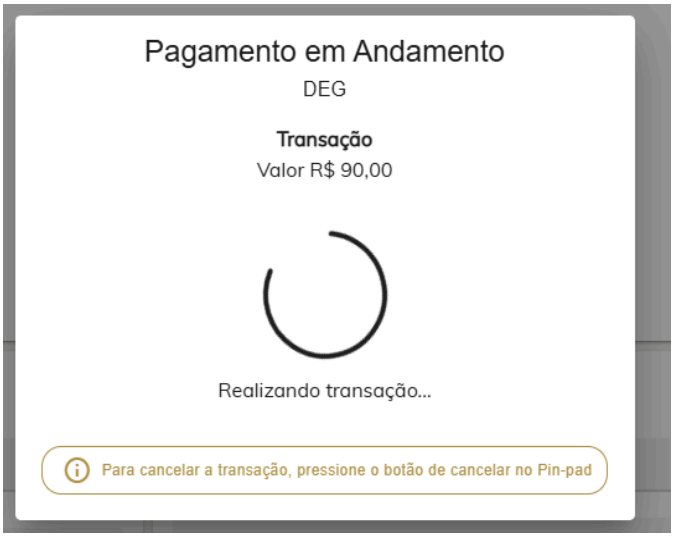

Figura 11. Renderização de Imagem de Código QR

Fonte: Elaborada pelo autor

Após o pagamento de cada parcela, é exibido um retorno do sistema indicando se a operação foi concluída com sucesso ou se houve algum erro. Após o pagamento de todas as parcelas integradas, será permitido ao usuário finalizar a venda no *popup* lateral do sistema, indicado na figura 12, que fica bloqueado até então em uma venda TEF.

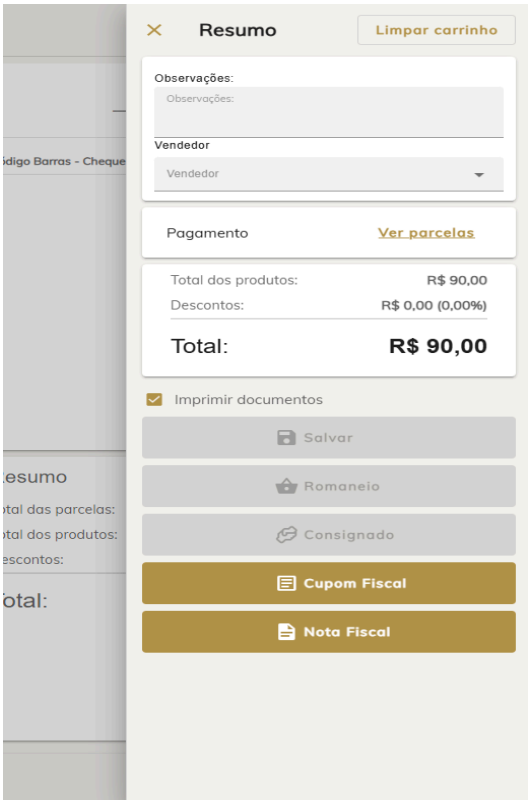

Figura 12. *Popup* Lateral Após Conclusão de Venda

Fonte: Elaborada pelo autor

4.2.2. Informação de CPF/CNPJ

Em vendas de nota fiscal, é importante que a pessoa jurídica possa informar seu CNPJ no documento, para que seja possível comprovar que a aquisição foi feita em nome da empresa, além de cumprir com o SPED fiscal ou escrituração contábil. Já em vendas com cupom fiscal para pessoa física, o mesmo recurso se torna necessário (mas não obrigatório) com o seu CPF, que pode assegurar a devolução ou reembolso, além de certos benefícios que variam de acordo com o estabelecimento, como prêmios de fidelidade ou descontos em compras futuras.

No sistema, essa função está disponível no cabeçalho da tela de venda, como visto na figura 13, sendo uma das primeiras ações que o usuário pode realizar no fluxo.

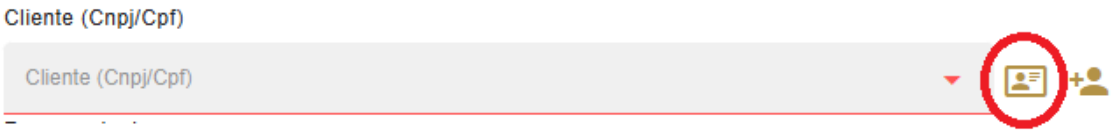

Figura 13. Ícone Para Informar Cpf/Cnpj

Fonte: Elaborada pelo autor

Ao clicar, a tela do *PIN pad* é ativada , possibilitando ao usuário informar o CPF/CNPJ do cliente final, relacionando o pedido e futuros documentos à essa identificação.

4.2.3. Confirmação de transações pendentes

Como descrito nos requisitos funcionais do sistema, ao final da venda, deve ser impresso o documento relacionado (cupom/nota fiscal), caso contrário, o pagamento ficará pendente e o dinheiro não será transferido do cliente final para o estabelecimento vendedor. Para esses casos, é exigido pela Destaxa a criação de um componente visual onde o vendedor possa manualmente associar uma transação pendente a uma parcela de um pedido com documento, permitindo que o pagamento seja confirmado. Tal componente pode ser aberto na tela principal do sistema após uma verificação da existência de pagamentos pendentes, que é apresentado ao usuário na forma de uma mensagem no topo da página, representada na figura 14.

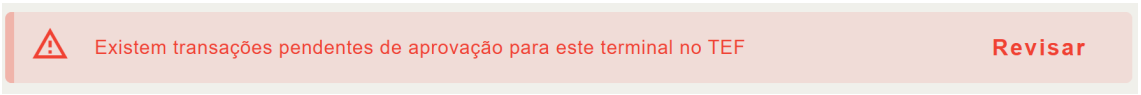

Figura 14. Mensagem de Transação Pendente

Fonte: Elaborada pelo autor

Após aberto, o usuário terá em sua tela as informações da transação pendente mais recente, podendo cancelar o pagamento ou confirmá-la escolhendo o pedido desejado, será mostrado informações como valor, rede de pagamento utilizada, data e código de autorização, informações essas que podem ser vistas na figura 15.

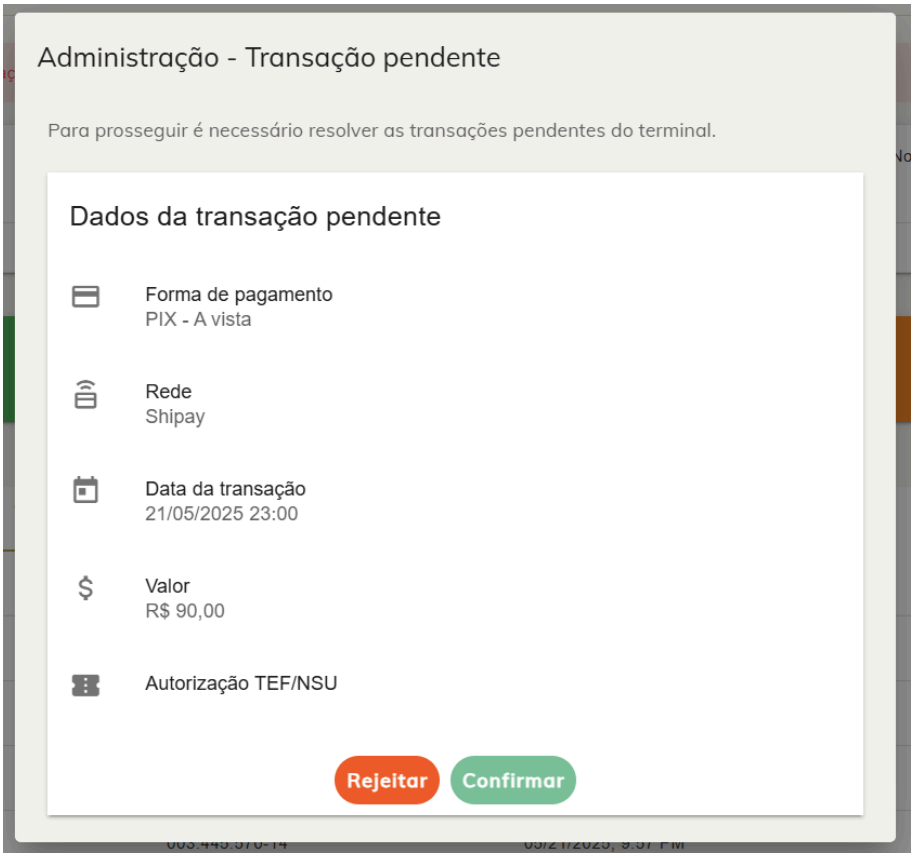

Figura 15. Início Fluxo Administração Pendente

Fonte: Elaborada pelo autor

Escolhendo rejeitar a transação, o sistema cancela o fluxo de envio do pagamento e, caso haja uma próxima transação pendente, irá mostrar ao usuário para que realize novamente a administração, até o momento que todas as transações forem confirmadas ou rejeitadas, nesse caso, o usuário será notificado sobre a conclusão da administração de pendências, sendo apresentado a ele a interface da figura 16.

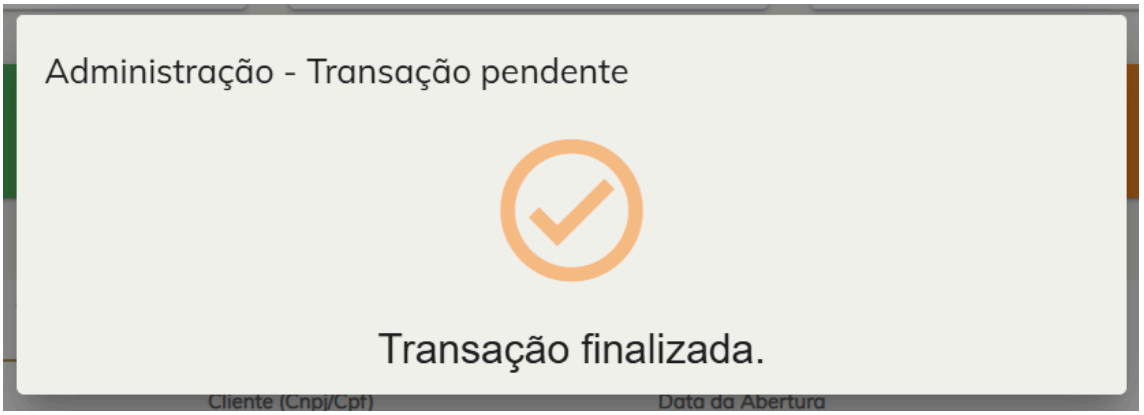

Figura 16. Aviso ao Concluir Administração

Fonte: Elaborada pelo autor

Caso o usuário escolha confirmar, ele será direcionado para um formulário que contém um campo de escolha para o pedido, além de uma lista de parcelas, ambos sendo usados para que o usuário atrele corretamente uma transação aos registros do sistema, permitindo que o pagamento, antes pendente, possa ser confirmado, conforme apresentado na figura 17.

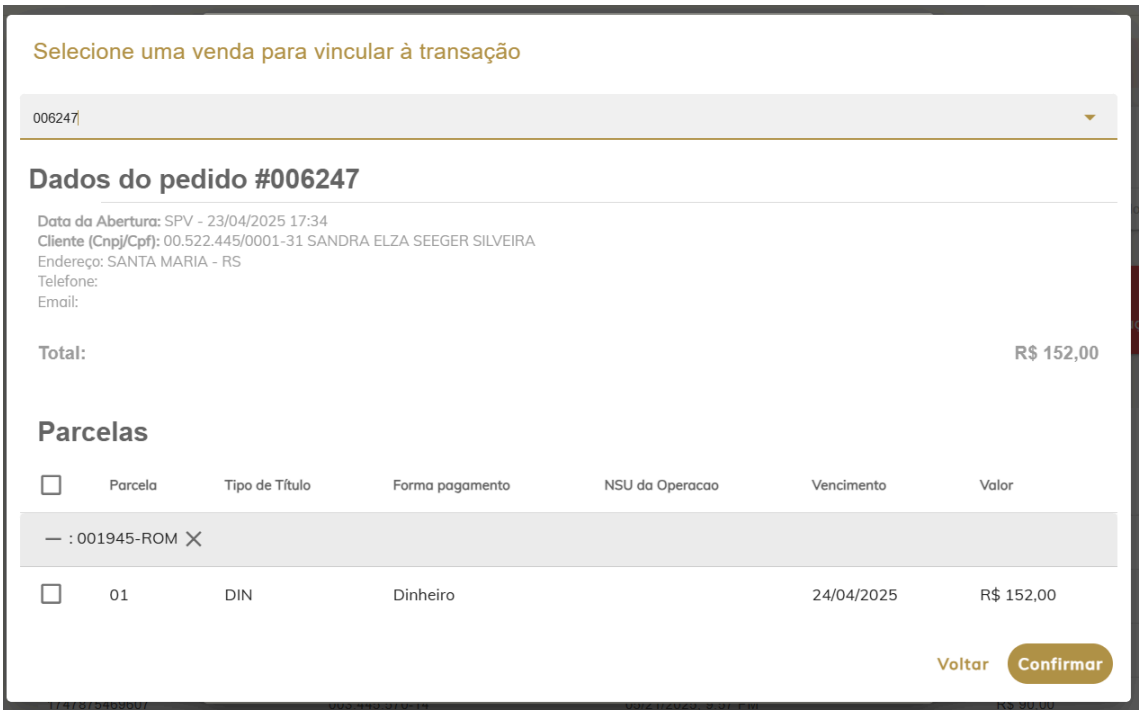

Figura 17. Confirmação de Transação

Fonte: Elaborada pelo autor

4.3. API para comunicação

Após a criação de todos os componentes visuais e a definição dos fluxos de pagamento, foi necessário desenvolver uma API local para gerenciar a comunicação entre o website, a DLL e o *client* da Destaxa. Essa API atua como um intermediário, recebendo as requisições do *frontend*, processando as informações necessárias e encaminhando as instruções para a DLL, que, por sua vez, se comunica com o *client* da Destaxa para realizar as transações de pagamento.

A principal função dessa API é garantir que as informações fluam corretamente entre as diferentes camadas do sistema, mantendo a integridade e a segurança dos dados. Por exemplo, quando uma transação é iniciada no website, a API verifica se há transações pendentes, gera os dados necessários para a nova transação e envia essas informações para a DLL. Além disso, a API também é responsável por receber as respostas das transações, atualizar o status no banco de dados e informar o *frontend* sobre o resultado da operação. Construir a arquitetura e o funcionamento dessa maneira facilita a manutenção e a escalabilidade do sistema, o que permite a atualização ou substituição de cada componente de forma independente, sem impactar as demais partes da aplicação.

Como citado anteriormente, a tecnologia principal utilizada para essa seção foi o ASP.NET, com a linguagem C#. Adotou-se o padrão de design de arquitetura em camadas, onde cada camada tem uma responsabilidade diferente. No caso desse projeto, a camada mais alta é chamada de Controlador (Controller), responsável por receber as requisições da interface visual através de rotas (como "*/TEF/GeneratePixQrCode*"), além de estabelecer regras de acesso para cada uma dessas rotas e delegar a lógica para funções da próxima camada, chamada de Serviços (Services), que recebe os dados do Controlador e os manipula, retornando apenas o modelo de dados que interessa o usuário. Abaixo, na figura 18, pode-se observar o código para uma rota construída no arquivo de Controladores da API.

```
[Authorize]
[HttpPost]
public async Task<ReturnModel<TEFTransactionResponse>> GeneratePixQrCode(TEFPixRequest pixRequest)
{
    var returnModel = new ReturnModel<TEFTransactionResponse>();
    try
    {
        returnModel.ObjectModel = await _TEFService.GeneratePixQrCode(pixRequest);
    }
    catch (Exception ex)
    {
        returnModel.SetCustomError((ex.InnerException != null ? ex.InnerException.Message : ex.Message));
    }
    return returnModel;
}
```

Figura 18. Exemplo de Rota No Fluxo PIX

Fonte: Elaborada pelo autor

Nesse fluxo de manipulação, muitas vezes é necessário a consulta de informações que fogem do escopo dado no início da requisição, fluxo que é delegado para a última camada da API, chamada de Repositório (Repository), responsável apenas pelas consultas no banco de dados. Esse padrão de código pode ser visualizado abaixo, na figura 19.

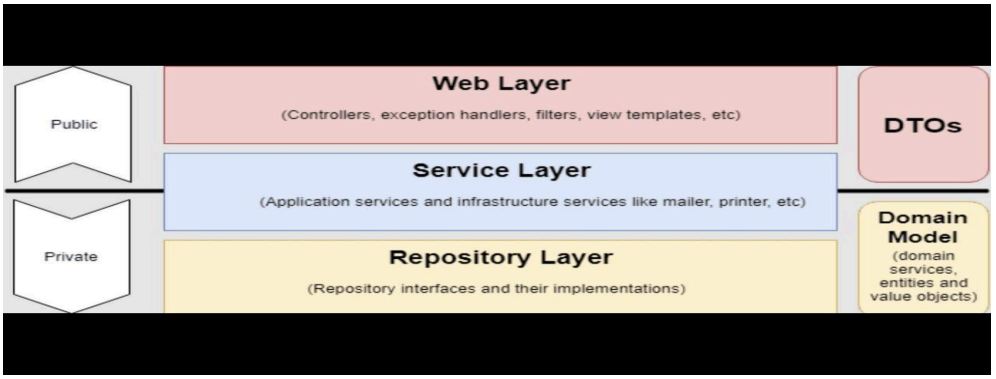

Figura 19. Visualização Arquitetura em Camadas

Fonte: Youtube, 2021

4.4. Desenvolvimento da DLL

Para possibilitar a integração entre o sistema PDV web e o terminal TEF da empresa Destaxa, a DLL é responsável por encapsular toda a lógica de comunicação com o *client* Destaxa, sendo instalado nos terminais dos estabelecimentos comerciais. Essa DLL, como citado anteriormente, atua como uma ponte entre a API web e o *client* local, abstraindo a complexidade da comunicação e padronizando o envio e recebimento de mensagens.

Uma das principais funcionalidades implementadas na DLL foi a conversão de objetos de entrada e saída da API em objetos com estrutura compatível com o protocolo compreendido pelo *client* da Destaxa. Para isso, utilizou-se a biblioteca Mapster, que oferece uma abordagem eficiente para o mapeamento de objetos do tipo DTO (Data Transfer Object). O uso do Mapster garantiu a conversão consistente entre os modelos internos da aplicação e os modelos esperados pelo terminal TEF, eliminando a necessidade de mapeamentos manuais repetitivos e facilitando a manutenção do código. Abaixo, é mostrado um exemplo de conversão entre dados alimentados pelo *frontend* e pela API, informados em inglês, para propriedades que serão lidas pelo *client* Destaxa, informados em português. O código que implementa a lógica citada, é apresentado na figura 20.

```
public record VendaPixCommand(
    string Transacao_valor,
    ServerConfigModel ServerConfig,
    int Sequencial,
    Retorno Retorno = Retorno.ExecuteService,
    Servico Servico = Servico.Execute,
    Transacao Transacao = Transacao.DigitalPay) : IRequest<AutomacaoColetaResponse>;

public static void RegisterVendaPixCommand(TypeAdapterConfig config)
{
    config.NewConfig<(PixRequest request, Socket socket), VendaPixCommand>()
        .ConstructUsing(
            src => new VendaPixCommand(
                src.request.TransactionValue.ToString("0.00", System.Globalization.CultureInfo.InvariantCulture),
                new ServerConfigModel()
                {
                    IPAddress = src.request.IPAddress,
                    Port = src.request.Port,
                    Socket = src.socket,
                },
                src.request.Sequential,
                Retorno.ExecuteService,
                Servico.Execute,
                Transacao.DigitalPay))
        ;

}
```

Figura 20. Conversão de Dados com Mapster

Fonte: Elaborada pelo autor

Além disso, a DLL utiliza a tecnologia WebSocket para realizar a comunicação com o *client* Destaxa. Diferente do modelo tradicional baseado em requisições *HTTP* síncronas, o WebSocket permite a comunicação bidirecional e em tempo real, fundamental para o correto funcionamento de transações TEF, que envolvem múltiplas etapas e exigem troca contínua de mensagens entre servidor e *client*.

Ao receber uma solicitação da API (por exemplo, uma transação de cartão ou a geração de um QR Code PIX), a DLL converte os dados usando o Mapster, estabelece a conexão WebSocket com o *client* local e envia a mensagem formatada conforme o protocolo da Destaxa. Em seguida, ela aguarda a resposta do *client*, interpreta o retorno e envia a resposta apropriada de volta à API, que por sua vez a repassa ao *frontend*. Na figura 21, é mostrado o código da lógica para envio e recebimento de mensagens com WebSocket .

```
public async Task SendMessage(Socket socket, string message)
{
    if(socket == null)
        throw new ArgumentNullException(nameof(socket));

    byte[] messageBytes = Encoding.UTF8.GetBytes(message);

    if(Parameters.TefLog)
        await RegisterMessageSent(message);

    socket.Send(messageBytes, SocketFlags.None);
}

public async Task<string> ReceiveMessage(Socket socket, int timeout =
0)
    byte[] buffer = new byte[10000000];
    socket.ReceiveTimeout = timeout == 0 ? 180000 : timeout;
    int bytesRead = socket.Receive(buffer);

    string content = Encoding.UTF8.GetString(buffer, 0, bytesRead);

    if (Parameters.TefLog)
        await RegisterMessageReceived(content);

    return content;
}
```

Figura 21. Envio e Recebimento de Mensagens WebSocket

Fonte: Elaborada pelo autor

A abordagem com uso de WebSocket permitiu o isolamento da lógica de integração TEF, facilitando testes, reaproveitamento e eventual substituição do *client* TEF sem afetar diretamente a API principal ou o *frontend* da aplicação.

5. Resultados

Este capítulo apresenta os resultados obtidos. O objetivo é demonstrar como as funcionalidades propostas foram testadas, avaliadas e quais resultados foram observados em termos de desempenho e confiabilidade.

A apresentação desses resultados visa fornecer uma visão clara e objetiva sobre a eficácia da integração TEF desenvolvida, bem como identificar pontos de aprimoramento que podem contribuir para o contínuo desenvolvimento e aperfeiçoamento do sistema.

5.1. Testes realizados

A cronologia da realização dos testes pode ser dividida em dois momentos, que são: durante a homologação do sistema com a Destaxa; e em produção, diretamente em clientes com o sistema já implantado.

No primeiro momento, a *TEF House* exige a execução de testes com a versão final da integração, como pré-requisito para a liberação da chave de ativação necessária à sua implantação nos clientes. Essa etapa é chamada de homologação e é constituída de diversos testes com as implementações obrigatórias, descritas nos requisitos do sistema, além de adições opcionais, como a possibilidade para o usuário final informar seu CPF/CNPJ em compras. Alguns dos testes feitos foram compras em sequência com diferentes métodos de pagamento, reimpressão de comprovante e confirmação de transações pendentes. Na primeira rodada dos testes de homologação, foram feitas observações por parte da Destaxa no fluxo de pagamento PIX, que foram corrigidas para a segunda rodada, onde o ambiente de homologação foi aprovado.

O segundo momento ocorreu após a homologação ser concluída, nos primeiros clientes onde as mudanças para pagamento integrado foram implementadas. Seguiu-se uma lista rígida de testes, com o intuito de assegurar o correto funcionamento das funcionalidades e mitigar possíveis imprevistos.

5.2. Desempenho do sistema

Para garantir que o fluxo de pagamentos, confirmações e impressões estivessem dentro do tempo esperado para um sistema profissional e prático, realizou-se uma nova série de testes em ambiente de homologação, considerando três cenários distintos: servidor e banco de dados rodando localmente, servidor rodando na nuvem com banco de dados local e, por último, servidor interno com banco de dados na nuvem.

Para cada cenário, foram feitas dez transações com diferentes métodos de pagamento e calculado a média de tempo. Dessas transações, quatro eram com PIX e, das outras seis, metade era com cartão de crédito e a outra metade com cartão de débito. No primeiro cenário, o desempenho foi o mais rápido de todos, devido a rapidez em que as informações são obtidas na rede, com média de 15,7 segundos, valor estimado como representativo para a maioria dos clientes, já que a sua infraestrutura segue esse formato.

5.3. Limitações do sistema

Com a construção e testes dos sistemas, foram notadas algumas limitações que impactam diretamente na estabilidade da integração. Uma das principais questões está relacionada à infraestrutura de rede utilizada e a tecnologia dos *PIN pad*, que apresenta instabilidade de conexão, tornando o sistema difícil de monitorar em certos pontos e imprevisível em seu comportamento. Essa instabilidade pode resultar em falhas de comunicação entre as camadas do sistema, afetando a experiência do usuário e a eficiência das transações.

Além disso, por falta de tempo e inexperiência com desenvolvimento de pagamento, o desenvolvimento do sistema não incorporou o uso de transações (*transactions*) no banco de dados. A não implementação dessa função pode comprometer a solidez das operações, não garantindo que todas as etapas de um processo sejam concluídas com sucesso ou revertidas em caso de falha. Isso pode resultar em inconsistências nos dados guardados e apresentados ao usuário, especialmente em situações de interrupção inesperada ou erro durante o processamento de uma transação.

6. Considerações Finais

O objetivo principal do projeto foi a integração de um sistema TEF ao PDV já existente e com uma base de clientes madura, buscando exigências fiscais e um sistema prático para realizar transações. Para atingir esse resultado, foi planejado e criado uma arquitetura flexível, que permitiu a implementação nos mais variados tipos de clientes.

No desenvolvimento, foram criados componentes que facilitam a interação do usuário com os principais fluxos integrados e que foram testados em diferentes cenários de arquiteturas de servidores, apresentando resultados satisfatórios para uma aplicação usada comercialmente, que exige agilidade. Tanto o *framework* Vue quanto a biblioteca de componentes Vuetify se provam sólidas para aplicações que lidem com movimentações financeiras.

Além da parte técnica, foram criados guias e vídeos intuitivos, cujo objetivo é ajudar o usuário final a ter uma introdução com uma curva de aprendizado mais suave, diminuindo o número de chamados de suporte.

Todos os requisitos funcionais foram atingidos, porém, foram identificadas certas limitações, derivada da mesma arquitetura flexível comentada anteriormente. Para trabalhos futuros que também envolvam transações financeiras, se revela necessário a adição de ferramentas e mecanismos que aumentem a estabilidade e confiabilidade dos fluxos, garantindo a integridade das operações e uma experiência agradável para o usuário.

Pensando em melhorias futuras no projeto, pode ser adicionado *transactions* para garantir a consistência e integridade das informações, além da adição de suporte para outras TEF Houses. Para futuras implementações, a DLL pode ser integrada com futuros projetos que necessitem solução de pagamentos.

7. Referências

ABLY. **Long Polling vs WebSockets – which to use in 2024**. Disponível em: https://ably.com/blog/websockets-vs-long-polling. Acesso em: 23 abr. 2025.

AL TEXSOFT. **What is WebSocket? The WebSocket API and protocol explained**. Disponível em: https://www.altexsoft.com/blog/websockets/. Acesso em: 21 abr. 2025.

BRASIL 61. **RS: A partir de janeiro, será obrigatória a integração da NFC-e com a TEF**. 10 nov. de 2023. Disponível em: https://brasil61.com/n/rs-a-partir-de-janeiro-sera-obrigatoria-a-integracao-da-nfc-e-com-a-tef-bras2310082. Acesso em: 11 abr. 2025.

CORRÊA, C. **Tefs para sistemas web**. Fórum Projeto ACBr, 2016. Disponível em: https://www.projetoacbr.com.br/forum/topic/70550-tef-rodar-em-web/. Acesso em: 23 abr. 2025.

DAPPER. **Dapper – a simple object mapper for .NET**. Disponível em: https://github.com/DapperLib/Dapper. Acesso em: 14 set. 2025.

DESTAXA. **Integração Destaxa**. Disponível em: https://destaxa.com.br/integracao-destaxa/. Acesso em: 25 abr. 2025.

ENOTAS. **O que é e como funciona a TEF (Transferência Eletrônica de Fundos)?** Disponível em: https://enotas.com.br/blog/o-que-e-tef/. Acesso em: 21 abr. 2025.

FAZENDA RS. **Varejistas devem estar atentos à obrigatoriedade de integração entre NFC-e e meios de pagamento eletrônico**. Disponível em: https://fazenda.rs.gov.br/conteudo/19389/varejistas-devem-estar-atentos-a-obrigatoriedade-de-integracao-entre-nota-fiscal-e-meios-de-pagamento-eletronico. Acesso em: 25 abr. 2025.

FOWLER, Martin. **Feature Toggles**. Disponível em: https://martinfowler.com/articles/feature-toggles.html. Acesso em: 12 maio 2025.

FREITAS, Paulo Springer de. **Mercado de cartões de crédito no Brasil: problemas de regulação e oportunidades de aperfeiçoamento da legislação**. Brasília: Senado Federal, Consultoria Legislativa do Senado Federal, 2007. Disponível em: https://www12.senado.leg.br/publicacoes/estudos-legislativos/tipos-de-estudos/textos-para-discussao/td-37-mercado-de-cartoes-de-credito-no-brasil-problemas-de-regulacao-e-oportunidades-de-aperfeicoamento-da-legislacao/@@download/file/TD37-PauloSpringer.pdf. Acesso em: 23 abr. 2025.

GCORE. **What Is WebSocket? | How Does It Work?** 25 out. 2023. Disponível em: https://gcore.com/learning/what-is-websocket. Acesso em: 23 abr. 2025.

IETF. **RFC 6455 – The WebSocket Protocol**. Disponível em: https://datatracker.ietf.org/doc/html/rfc6455. Acesso em: 23 abr. 2025.

INFOVAREJO. **Quais as vantagens e desvantagens do TEF?** Disponível em: https://infovarejo.com.br/vantagens-desvantagens-tef/. Acesso em: 21 abr. 2025.

INFOVAREJO. **TEF: estruturas e serviços necessários para utilização**. Disponível em: https://infovarejo.com.br/estruturas-e-servicos-para-tef/. Acesso em: 21 abr. 2025.

IUPANA. **As carteiras digitais mais utilizadas na América Latina**. Disponível em: https://iupana.com/2025/03/10/as-carteiras-digitais-mais-utilizadas-na-america-latina/?lang=pt-br. Acesso em: 20 abr. 2025.

MICROSOFT. **Using Vue in Visual Studio Code**. Disponível em: https://code.visualstudio.com/docs/nodejs/vuejs-tutorial. Acesso em: 27 abr. 2025.

MICROSOFT. **Visual Studio Code for Vue.js**. Disponível em: https://www.vuemastery.com/blog/vs-code-for-vuejs-developers/. Acesso em: 27 abr. 2025.

MICROSOFT. **WebSockets support in ASP.NET Core**. Disponível em: https://learn.microsoft.com/aspnet/core/fundamentals/websockets?view=aspnetcore-9.0. Acesso em: 27 abr. 2025.

MICROSOFT LEARN. **Dynamic link library (DLL)**. Disponível em: https://learn.microsoft.com/en-us/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library. Acesso em: 08 mai. 2025.

OWASP. **Testing WebSockets – WSTG**. Disponível em: https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/11-Client_Side_Testing/10-Testing_WebSockets. Acesso em: 23 abr. 2025.

PINHEIRO DA SILVEIRA, M. A.; AGUILAR, N. G. S. **Impactos da informatização na gestão de supermercados**. Revista de Administração Mackenzie, v. 8, n. 1, p. 108–132, jan. 2007. Disponível em: https://www.scielo.br/j/ram/a/hmFjGDs3f4GrCFWLMq3yNvP/. Acesso em: 23 abr. 2025.

PISMO. **Carteiras digitais em 2024: navegando pelas novas tendências para tecnologias emergentes**. Disponível em: https://www.pismo.io/pt/blog/digital-wallets-in-2024-navigating-new-trends-for-emerging-technology/. Acesso em: 20 abr. 2025.

REUTERS. **Brazil's Pix payments are killing cash. Are credit cards next?** 2 abr. 2024. Disponível em: https://www.reuters.com/business/finance/brazils-pix-payments-are-killing-cash-are-credit-cards-next-2024-04-02/. Acesso em: 20 abr. 2025.

REUTERS. **New feature of Brazil's Pix system to draw $30 billion in e-commerce payments**. 28 jan. 2025. Disponível em: https://www.reuters.com/business/finance/new-feature-brazils-pix-system-draw-30-billion-e-commerce-payments-2025-01-28/. Acesso em: 21 abr. 2025.

TRICHES, D.; BERTOLDI, A. **A evolução do sistema de pagamentos brasileiro: uma abordagem comparada com os países selecionados no período 1995-2003**. Revista de Economia Contemporânea, v. 10, n. 2, ago. 2006. Disponível em: https://www.scielo.br/j/rec/a/pYp7JjgywWJZF6q8hrqVxNM/#. Acesso em: 11 abr. 2025.

VUEJS. **Performance | Vue.js**. Disponível em: https://vuejs.org/guide/best-practices/performance. Acesso em: 27 abr. 2025.

VUETIFY. **Get started with Vuetify 3**. Disponível em: https://vuetifyjs.com/en/getting-started/installation/. Acesso em: 27 abr. 2025.

YOUTUBE. Calling Service Layer On Controller In Asp.Net Core |Controller service repository pattern | Part-10. Disponível em: https://www.youtube.com/watch?v=ZFIXvzllDrg. Acesso em: 21 maio 2025.