

SOCIEDADE EDUCACIONAL PINHALZINHO
HORUS FACULDADES

Cristyan Schabarum

**DESENVOLVIMENTO DE PROTÓTIPO DE SOFTWARE PARA GESTÃO DE
ENTREGAS FISCAIS E CONTÁBEIS (eGerir)**

Pinhalzinho/SC

2023

CRISTYAN SCHABARUM

**DESENVOLVIMENTO DE PROTÓTIPO DE SOFTWARE PARA GESTÃO DE
ENTREGAS FISCAIS E CONTÁBEIS (eGerir)**

Trabalho de conclusão de curso para o curso de Sistemas de Informação
apresentada à Horus Faculdades como parte dos requisitos para obtenção
do grau de Bacharel em Sistemas de Informação

Orientador: Prof. Renan José Borgheti

Pinhalzinho/SC

2023

AGRADECIMENTO

Agradeço à minha família, amigos e orientador por seu apoio inestimável ao longo da minha jornada acadêmica. Sua presença, incentivo e orientação foram essenciais para a conclusão desta monografia. Também expresso gratidão a todos que contribuíram de alguma forma para este projeto. Esta monografia é dedicada a todos vocês, e espero que meus esforços tenham valido a pena.

Cristyan Schabarum.

“O espaço entre a teoria e a prática não é tão grande como é, a teoria na prática.”

(Lucas Bispo de Oliveira)

RESUMO

O contexto tributário para empresas no Brasil é caracterizado por sua complexidade, desafios e a necessidade premente de uma gestão eficaz. Conforme destacado por Favarin e Oliveira (2021), o sistema tributário nacional é marcado por uma variedade de impostos, altas taxas e uma legislação complexa. Diante desses desafios, Sant'Anna e Pereira (2021) ressaltam a importância da conformidade tributária, enfatizando as penalidades significativas para o não cumprimento das obrigações fiscais. A carga tributária, conforme evidenciado por Ramos et al. (2021), atinge até 63,4% da receita líquida, tornando-se uma das mais elevadas globalmente, com uma distribuição desigual. Nesse contexto desafiador, o presente trabalho propõe-se a desenvolver um software de gestão contábil, visando simplificar e tornar transparente o cumprimento das obrigações fiscais e contábeis. Essa iniciativa busca não apenas facilitar o processo, mas também permitir que as empresas foquem na geração de receita, implementando um planejamento tributário eficaz.

Palavras-chaves: Complexidade tributária; Carga tributária; Software de gestão contábil; Planejamento tributário eficaz.

ABSTRACT

The tax context for businesses in Brazil is characterized by its complexity, challenges, and the urgent need for effective management. As highlighted by Favarin and Oliveira (2021), the national tax system is marked by a variety of taxes, high rates, and complex legislation. Faced with these challenges, Sant'Anna and Pereira (2021) emphasize the importance of tax compliance, highlighting significant penalties for non-compliance with tax obligations. The tax burden, as evidenced by Ramos et al. (2021), reaches up to 63.4% of net income, making it one of the highest globally, with uneven distribution. In this challenging context, this work aims to develop accounting management software, aiming to simplify and make transparent the compliance with tax and accounting obligations. This initiative seeks not only to streamline the process but also to enable companies to focus on revenue generation by implementing effective tax planning.

Keywords: Tax complexity; Tax burden; Accounting management software; Effective tax planning.

LISTA DE FIGURAS

Figura 01: Exemplificação da arquitetura adotada no projeto.....	25
Figura 02: Ilustração de um SGBD.....	28
Figura 03: Exemplificação do diagrama ER do projeto.....	36
Figura 04: Exemplificação de diagrama de classe produzido para o projeto.....	37
Figura 05: Exemplificação de diagrama de caso de uso.....	39
Figura 06: Exemplificação de diagrama de sequência.....	40
Figura 07: Exemplificação de diagrama de sequência.....	41
Figura 08: Exemplificação de criação da imagem do banco de dados postgre.....	42
Figura 09: Exemplificação de criação das tabelas.....	43
Figura 10: Exemplificação de criação das chaves primárias e estrangeiras.....	43
Figura 11: Exemplificação da criação da estrutura de diretórios do backend.....	44
Figura 12: Exemplificação de criação de arquivo service.....	46
Figura 13: Exemplificação de um arquivo controller.....	47
Figura 14: Exemplificação de arquivo router.....	48
Figura 15: Exemplificação da documentação das rotas feitas no swagger.....	49
Figura 16: Exemplificação de criação de um projeto VCL.....	50
Figura 17: Exemplificação da estrutura dos diretórios no frontend.....	51
Figura 18: Estrutura da classe TFrwMenuForms.....	52
Figura 19: Estrutura da classe TFrwAPIControl.....	53
Figura 20: Ilustra o formulário padrão de mensagens do sistema.....	54
Figura 21: Ilustração da herança da tela de cadastro.....	55
Figura 22: Implementação da tela de cadastro de empresa.....	56
Figura 23: Tela de login do sistema.....	57
Figura 24: Menu principal do sistema.....	57
Figura 25: Tela de cadastro.....	58
Figura 26: Projeto VCL e FMX adicionado ao grupo.....	59
Figura 27: Imagem da tela de Kanban do projeto.....	60

LISTA DE ABREVIACÕES

II	Imposto sobre Importação
IR	Imposto de Renda
IRPF	Imposto de Renda sobre Pessoa Física
IRPJ	Imposto de Renda sobre Pessoa Jurídica
FPE	Fundo de Participação dos Estados
FPM	Fundo de Participação dos Municípios
FNO	Fundos Constitucionais do Norte
FNE	Fundos Constitucionais do Nordeste
FCO	Fundos Constitucionais do Centro-Oeste
IOF	Imposto sobre Operações Financeiras
ICMS	Imposto sobre Circulação de Mercadorias e Serviços
ITCMD	Imposto sobre Transmissão Causa Mortis e Doação
IPVA	Imposto sobre a propriedade de veículos automotores
FUNDEB	Fundo de Manutenção e Desenvolvimento da Educação Básica e de Valorização dos Profissionais da Educação
IPTU	Imposto sobre a Propriedade Predial e Territorial Urbana
ITBI	Imposto sobre a Transmissão Inter-Vivos de Bens Imóveis e de Direitos a Eles Relativos
ISS/ISSQN	Imposto sobre Serviços de Qualquer Natureza
MEI	Microempreendedor individual
DAS	Documento de Arrecadação do Simples Nacional
IA	Inteligência Artificial
IOT	Internet das coisas
MVC	Model-View-Controller
URL	Uniform Resource Locator
UML	Unified Modeling Language
SGBD	Sistema de gerenciamento de banco de dados
PIB	Produto interno bruto

SUMÁRIO

1 INTRODUÇÃO.....	11
2 OBJETIVOS DA PESQUISA.....	12
2.1 OBJETIVO GERAL.....	12
2.2 OBJETIVOS ESPECÍFICOS.....	12
3 CARGA TRIBUTÁRIA.....	13
3.1 BREVE HISTÓRICO DO SISTEMA TRIBUTÁRIO BRASILEIRO.....	13
4 SISTEMA TRIBUTÁRIO BRASILEIRO.....	15
4.1 ESTRUTURA TRIBUTÁRIA BRASILEIRA.....	15
4.2 PRINCIPAIS TRIBUTOS FEDERAIS, ESTADUAIS E MUNICIPAIS.....	16
4.2.1 Impostos Federais.....	16
4.2.2 Impostos Estaduais.....	17
4.2.3 Impostos municipais.....	18
4.3 IMPOSTOS DIRETOS E INDIRETOS: DEFINIÇÃO E DIFERENÇAS.....	19
4.4 PRINCIPAIS DESAFIOS DO SISTEMA TRIBUTÁRIO BRASILEIRO.....	20
4.5 REFORMA TRIBUTÁRIA (PEC 45/2019 EM TRAMITAÇÃO).....	22
4.6 DESAFIOS ENFRENTADOS NO CUMPRIMENTO DOS PRAZOS NO PAGAMENTO DOS IMPOSTOS E OBRIGAÇÕES ACESSÓRIAS.....	23
5 PROCEDIMENTOS METODOLÓGICOS.....	24
6 TECNOLOGIAS ENVOLVIDAS.....	25
6.1 KANBAN.....	25
6.2 ARQUITETURA.....	26
6.3 TECNOLOGIAS USADAS.....	27
6.3.1 Docker.....	28
6.3.2 Banco de Dados.....	29
6.3.2.1 PostgreSQL.....	31
6.3.3 Linguagem de programação.....	32
6.3.3.1 Delphi.....	33
6.3.3.2 JavaScript.....	33
6.3.4 UML.....	34
6.3.5 Ferramentas de desenvolvimento.....	34
6.3.5.1 RAD Studio.....	35
6.3.5.2 Visual Studio Code.....	36
7 ANÁLISE E IMPLEMENTAÇÃO DO PROTÓTIPO.....	37
7.1 DESCRIÇÃO DO PROTÓTIPO.....	37
7.2 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS.....	37
7.2.1 Requisitos funcionais.....	38
7.2.2 Requisitos não funcionais.....	39
7.3 DIAGRAMA DE ENTIDADE E RELACIONAMENTO.....	40

7.4 DIAGRAMA DE CLASSE.....	41
7.5 DIAGRAMA DE CASO DE USO.....	42
7.6 DIAGRAMA DE SEQUÊNCIA.....	44
7.7 BACKEND.....	46
7.8 FRONTEND.....	53
7.8.1 VCL.....	53
7.8.2 FMX.....	63
8 CRONOGRAMA.....	66
9 CONSIDERAÇÕES FINAIS.....	67
REFERÊNCIAS BIBLIOGRÁFICAS.....	69

1 INTRODUÇÃO

O cenário tributário atual para as empresas brasileiras pode ser descrito como complexo e desafiador. De acordo com o artigo de Armando Favarin e Edilene Santana de Oliveira (2021), intitulado "O cenário tributário atual no Brasil e os desafios para as empresas", o sistema tributário brasileiro é caracterizado por uma grande variedade de impostos e contribuições, altas taxas e uma legislação tributária extremamente complexa e frequentemente sujeita a mudanças.

Além disso, segundo o estudo de Paula Sant'Anna e Luis Felipe Vital Nunes Pereira (2021), publicado na Revista de Administração Mackenzie, as empresas brasileiras enfrentam desafios adicionais em relação à compliance tributária, uma vez que o não cumprimento das obrigações fiscais pode resultar em multas pesadas e sanções legais.

De acordo com o estudo de Aléssio Tony Ramos et al. (2021), publicado na Revista Brasileira de Finanças, as empresas no Brasil enfrentam uma carga tributária total que pode chegar a 63,4% da receita líquida, sendo uma das mais elevadas do mundo. Além disso, a distribuição dos impostos no Brasil é desigual, com uma alta concentração de tributos sobre o consumo e uma baixa tributação sobre a renda e a propriedade.

Outro artigo importante é o de Thiago Neves et al. (2021), publicado na Revista de Contabilidade e Organizações, que destaca a complexidade e a falta de transparência do sistema tributário brasileiro como fatores que contribuem para a insegurança jurídica e para o ambiente de negócios desfavorável no país.

Em resumo, o cenário tributário atual para as empresas brasileiras é caracterizado por uma grande complexidade e desafios relacionados à conformidade tributária e à carga tributária em si. É importante que as empresas estejam atentas às mudanças na legislação tributária e adotem práticas eficazes de gestão tributária para minimizar os impactos financeiros e legais de possíveis não conformidades.

O foco desta pesquisa é auxiliar na gestão e na distribuição de atividades a membros das equipes contábeis, relacionadas às entregas das obrigações fiscais e contábeis (Impostos e obrigações acessórias), simplificando e deixando esse processo mais transparente de modo que permita às empresas a focar na geração de receita, e realizem um planejamento tributário eficaz.

2 OBJETIVOS DA PESQUISA

2.1 OBJETIVO GERAL

Desenvolver um protótipo de software que atenda às necessidades de escritórios contábeis na gestão e na distribuição de suas atividades de maneira eficiente e que auxilie no controle das entregas tributárias e obrigações acessórias (Difal, Dirf, Sped, Reinf, etc...) de modo que possibilite uma organização e visão de progresso adequada utilizando a metodologia de Kanban.

2.2 OBJETIVOS ESPECÍFICOS

- Estudar o cenário tributário atual brasileiro a fim de compreender suas características e complexidade.
- Propor um protótipo que auxilie a organização e delegação de atividades nas equipes contábeis.
- Identificar e propor uma sistemática eficiente de maneira que favoreça os profissionais que forem utilizar dessa ferramenta.
- Desenvolver um Kanban capaz de atender e auxiliar na gestão e distribuição das entregas a serem realizadas pelas equipes contábeis.

3 CARGA TRIBUTÁRIA

Esta seção aborda um breve histórico do sistema tributário brasileiro, desde a sua concepção.

3.1 BREVE HISTÓRICO DO SISTEMA TRIBUTÁRIO BRASILEIRO

A origem da tributação sobre a produção brasileira se inicia desde a sua colonização pelos portugueses, Lunelli afirma que durante o século 18, o Brasil Colônia era tributado em 20% do ouro extraído, conhecido como "O Quinto", imposto cobrado por Portugal. Esse tributo, que era odiado pelos brasileiros, recaía principalmente sobre a produção de ouro. A Coroa Portuguesa tentou cobrar os "quintos atrasados" de uma só vez, no evento conhecido como "A Derrama". A revolta da população resultou na Inconfidência Mineira em 1789, liderada por Tiradentes, que buscava a libertação do Brasil de Portugal. Atualmente, a carga tributária brasileira está em torno de 33,71% do PIB (Ministério da Fazenda, 2023), 13,71% a mais do que era exigido por Portugal na época da Inconfidência Mineira.

A estrutura tributária que vigorou no Império brasileiro foi amplamente herdada pela República brasileira até a década de 1930. Naquele período, a economia do país era principalmente agrícola e altamente aberta ao comércio exterior, e a principal fonte de receitas públicas durante o Império provinha dos impostos de importação, que representaram cerca de dois terços da receita pública em alguns anos (VARSAÑO, 1996).

Na época da proclamação da República, o imposto de importação já respondia por metade da receita total do governo. A Constituição de 1891 adotou o sistema tributário existente no final do Império, com algumas mudanças para acomodar o regime federativo e dar aos estados e municípios autonomia financeira.

Foi criado um regime de separação de fontes tributárias, com impostos de competência exclusiva da União e dos estados. A União ficou com o imposto de importação, taxas de selo, taxas de correios e telégrafos federais, enquanto aos estados foi concedida a competência exclusiva para decretar impostos sobre a exportação, sobre imóveis rurais e

urbanos, sobre a transmissão de propriedades e sobre indústrias e profissões, além de taxas de selo e contribuições para seus correios e telégrafos.

Segundo Varsano (1996) os municípios foram autorizados pelos estados a fixar impostos municipais para garantir sua autonomia, e tanto a União quanto os estados tiveram poder para criar outras receitas tributárias. Embora existissem impostos sobre vencimentos pagos pelo governo e benefícios distribuídos por sociedades anônimas, os impostos discriminados na Constituição eram principalmente tributos sobre o comércio exterior ou impostos tradicionais sobre a propriedade ou sobre a produção e as transações internas. A tributação de fluxos internos de produtos começou com a cobrança de um imposto sobre o fumo em 1892 e, posteriormente, foi estendida a outros produtos, culminando com a criação do imposto de consumo. Em 1922, foi criado o imposto sobre vendas mercantis, mais tarde conhecido como imposto de vendas e consignações e transferido para a esfera estadual. Somente a partir de 1924, o governo instituiu um imposto de renda geral.

4 SISTEMA TRIBUTÁRIO BRASILEIRO

Neste tópico, será realizada uma análise abrangente da estrutura tributária brasileira, destacando os principais impostos em âmbito nacional, estadual e municipal, além de descrever as diferenças de impostos diretos e indiretos. Além disso, serão elencados os principais desafios enfrentados pelos contribuintes no Brasil ao atender a todas as exigências fiscais e tributárias. Neste tópico também será abordado a PEC 45/2019 que trata da reforma tributária em andamento.

4.1 ESTRUTURA TRIBUTÁRIA BRASILEIRA

Segundo Campanelle (2012) a estrutura tributária brasileira difere daquela dos países desenvolvidos, os quais possuem sistemas tributários mais avançados e modernos que incidem mais sobre a renda e propriedade do que sobre o consumo. Esses sistemas são capazes de fornecer bens e serviços públicos de alta qualidade proporcionalmente à tributação. No entanto, no Brasil, os tributos são mais direcionados ao consumo, e a carga tributária é maior quanto pior é a distribuição de renda. Uma análise da carga tributária revela que mais de 47% da arrecadação provém do consumo, menos de 20% da renda e menos de 5% das transações financeiras e propriedade. Embora os princípios da capacidade contributiva e progressividade estejam presentes na Constituição brasileira, eles são amplamente desrespeitados na prática. Além disso, embora a contribuição para a Previdência Social seja meritória, ela também é repassada aos preços dos bens e serviços, afetando tanto empregadores como trabalhadores.

A Constituição Federal de 1988 relacionada à tributação, tinha como objetivo simplificar o sistema tributário e organizar suas bases. Nesse processo, os impostos especiais (ISTR e ISC) foram fundidos com os impostos únicos (IUCL e IUEE) para criar o ICMS. No entanto, a proposta de criar um imposto sobre o valor agregado (IVA), com base mais ampla e princípio de destino, não foi implementada. Isso resultou na coexistência de três impostos sobre valor agregado (IPI, ICMS e ISS) compartilhados entre a União, Estados e Municípios. Essa situação levanta discussões frequentes no Congresso Nacional sobre a unificação desses

tributos em um IVA, como parte de uma reforma tributária. (HENRIQUE DE CAMPOS JÚNIOR, 2020). Esse assunto será abordado também no item 4.5 da presente monografia, que trata sobre a PEC 45/2019, atualmente em tramitação.

4.2 PRINCIPAIS TRIBUTOS FEDERAIS, ESTADUAIS E MUNICIPAIS

A seguir, apresentamos uma distribuição dos principais tributos nos âmbitos federal, estadual e municipal. Essa divisão reflete a complexa estrutura tributária brasileira, composta por diferentes impostos, taxas e contribuições, cada um com suas respectivas competências e finalidades. Essa diversidade de tributos visa atender às necessidades financeiras das diferentes esferas de governo, possibilitando a execução de políticas públicas e o financiamento de serviços essenciais à população. Dessa forma, a compreensão desses tributos é fundamental para uma análise abrangente do sistema tributário do país.

4.2.1 Impostos Federais

Segundo (Maciel & Carvalho, 2008) a competência para a cobrança do Imposto sobre Importação (II) é federal e sua incidência ocorre quando há entrada de mercadoria no território nacional com o objetivo de ser incorporada à economia interna, não se limitando a um ingresso físico temporário.

Já o Imposto de Renda (IR), que incide sobre a renda, é de competência federal para sua apuração. O Imposto de Renda é um tributo federal aplicado anualmente sobre a renda dos brasileiros, acompanhando a evolução patrimonial ao longo dos anos. Desde 1922, o governo requer que trabalhadores e empresas informem à Receita Federal seus ganhos anuais, para posteriormente avaliar se o valor cobrado corresponde aos rendimentos. O propósito do Imposto de Renda é de natureza social, buscando que aqueles com maiores rendimentos contribuam mais para financiar melhorias na qualidade de vida de toda a população. Desde 1979, o Imposto de Renda se tornou uma das principais fontes de receita do Governo Federal,

com arrecadação de R\$ 1,878 trilhão em 2021, representando um aumento de 17,36% em relação ao ano anterior (Receita Federal, 2023).

O Imposto de Renda sobre Pessoa Física (IRPF) incide sobre os rendimentos da pessoa física e é gerado e retido na fonte da renda.

Já o Imposto de Renda sobre Pessoa Jurídica (IRPJ) tem como fato gerador os rendimentos sobre a pessoa jurídica. Dos recursos arrecadados, 21,5% são destinados para o Fundo de Participação dos Estados (FPE), 23,5% para o Fundo de Participação dos Municípios (FPM) e 3% para os Fundos Constitucionais do Norte (FNO), do Nordeste (FNE) e do Centro-Oeste (FCO).

A sigla "IOF" significa Imposto sobre Operações Financeiras, e o nome completo do tributo é "Imposto sobre Operações de Crédito, Câmbio e Seguro, ou relativas a Títulos ou Valores Mobiliários". Ele é um tributo federal que se aplica a pessoas físicas e empresas e incide sobre operações relacionadas a crédito, câmbio, seguros e títulos mobiliários. Além de ser uma fonte de receita para o governo, o IOF também serve como indicador econômico, pois o aumento na arrecadação desse imposto reflete um maior volume de operações financeiras no período medido, o que pode indicar uma demanda crescente por crédito no Brasil.

4.2.2 Impostos Estaduais

Segundo (Maciel & Carvalho, 2008) o ICMS é um imposto que incide sobre as operações de circulação de mercadorias consideradas negócio jurídico mercantil. O Imposto ICMS é um tributo estadual que incide sobre uma variedade de produtos, incluindo bens importados, e é cobrado de forma indireta, ou seja, seu valor é adicionado ao preço do produto ou serviço. O ICMS é acionado quando a titularidade do bem ou serviço é transferida para o comprador, ou seja, quando a mercadoria é vendida ou o serviço é prestado ao consumidor. A regulamentação e a porcentagem do ICMS são determinadas por cada Estado e pelo Distrito Federal, resultando em diferentes taxas para diferentes localidades.

O ITCMD é o Imposto sobre Transmissão Causa Mortis e Doação, um tributo estadual que incide sobre a transferência de bens por herança ou doação. Para ser cobrado, a transferência deve ser não onerosa, ou seja, não resultar de uma venda. Sua base legal está na

Constituição Federal (artigo 155) e no Código Tributário Nacional (artigos 33 a 45), sendo regulamentado pelos estados, que determinam as alíquotas aplicáveis.

O IPVA é um imposto anual destinado aos proprietários de veículos automotores nos estados brasileiros. Sua cobrança visa arrecadar recursos para o governo local, que distribui parte desses fundos tanto para o estado quanto para os municípios. Em São Paulo, por exemplo, 20% da arrecadação é destinada ao FUNDEB, que apoia a educação básica e valoriza os profissionais da educação.

4.2.3 Impostos municipais

Segundo (Maciel & Carvalho, 2008) o Imposto sobre a Propriedade Territorial Urbana (IPTU) é gerado pela propriedade, domínio útil ou posse de um imóvel urbano. O IPTU é um imposto municipal cobrado de proprietários de imóveis urbanos, como casas, apartamentos ou salas comerciais. A quantia a ser paga depende da avaliação do imóvel, e as prefeituras têm autonomia para definir seus critérios de cobrança. A arrecadação é destinada exclusivamente ao município, podendo ser utilizada para financiar obras na cidade.

Já o Imposto sobre a Transmissão Inter-Vivos de Bens Imóveis e de Direitos a Eles Relativos (ITBI) é gerado pela transmissão de bens imóveis e direitos de aquisição. O ITBI é um imposto municipal cobrado na compra e venda de imóveis, também conhecido como Imposto sobre a Transmissão de Bens Imóveis ou Imposto Sobre Transmissão de Imóveis em algumas cidades. Ele está previsto na Constituição Federal, no Art. 156, inciso II, e refere-se à transferência da propriedade do vendedor para o comprador, aplicando-se a casas, apartamentos e imóveis na planta. O pagamento do ITBI é necessário para que a documentação de transmissão da propriedade seja liberada, tornando o comprador oficialmente proprietário do imóvel.

O Imposto sobre Serviços de Qualquer Natureza (ISS) incide sobre prestações de serviços, que envolvem uma obrigação de fazer algo a terceiros. Todos os recursos arrecadados são receita do município. O ISS, também conhecido como ISSQN, é um tributo relacionado à prestação de serviços em todo o Brasil. É regulado pelas Leis Complementares 116/2003 e 175/2020, sendo de responsabilidade dos municípios e do Distrito Federal sua estruturação e arrecadação. A alíquota varia de 2% a 5%, dependendo da localização e do tipo de serviço prestado. Empresas no Lucro Real ou Presumido pagam mensalmente,

enquanto profissionais autônomos pagam após a emissão da nota. No caso de MEIs e organizações no Simples Nacional, a coleta é realizada através do DAS (Documento de Arrecadação do Simples Nacional).

4.3 IMPOSTOS DIRETOS E INDIRETOS: DEFINIÇÃO E DIFERENÇAS

A classificação dos impostos pode ser dividida em dois grandes grupos: impostos diretos, que estão diretamente ligados à capacidade de pagamento individual, e impostos indiretos, que incidem sobre bens e serviços independentemente da capacidade de pagamento do indivíduo. Essa classificação é relevante para compreender os impactos dos impostos no poder de compra dos indivíduos e na estrutura tributária. Os impostos indiretos possuem maior peso orçamentário e incidem sobre toda a população, sendo que as classes sociais mais desfavorecidas arcam com o maior ônus tributário. Já os impostos diretos possuem menor grau de arrecadação e são originados das classes sociais mais altas, as quais possuem maior capacidade de pagamento. (SANTOS, 2003).

Segundo o Sitecontabil (2021) os impostos diretos, como o imposto de renda, IPVA e IPTU, têm uma relação direta com a renda do contribuinte, onde quanto maior a renda, maior o tributo. Por outro lado, impostos indiretos, como ICMS, ISS e IPI, são cobrados sobre produtos e serviços, sendo repassados aos consumidores por meio de aumentos nos preços.

Esses impostos afetam as empresas de diversas maneiras, incluindo impactos financeiros e desafios de gestão. Os impostos indiretos são particularmente complexos, variando por município e criando dificuldades na organização dos pagamentos e no planejamento financeiro. Aumentos de impostos podem levar a reclamações dos clientes que não compreendem os aumentos de preços, prejudicando o relacionamento com eles.

Além disso, as empresas enfrentam desafios internos de conformidade devido à complexidade dos impostos indiretos. Erros no pagamento podem resultar em juros e multas. Por outro lado, algumas empresas pagam mais impostos do que o devido por falta de organização e conhecimento de suas obrigações.

Portanto, a gestão eficaz dos impostos é fundamental para as finanças das empresas, garantindo que não haja gastos desnecessários e evitando multas e juros. Isso destaca a

importância do controle e planejamento adequado dos impostos para o sucesso financeiro das empresas.

4.4 PRINCIPAIS DESAFIOS DO SISTEMA TRIBUTÁRIO BRASILEIRO

Uma revolução está em curso, transformando a indústria, economia e sociedade em um ritmo acelerado. A tecnologia está mudando profundamente a forma como vivemos, trabalhamos, geramos riqueza, consumimos e nos relacionamos (SCHWAB, 2016).

A digitalização alterou a maneira de fazer negócios e gerenciar riquezas, criando modelos empresariais novos e tornando obsoletos modelos tradicionais. Além disso, está mudando a natureza do trabalho e a forma como o Governo e a sociedade se comunicam. Portanto, é necessário que o Governo e as instituições jurídicas adaptem-se aos tempos atuais para lidar com necessidades e demandas emergentes, e administrar canais e instrumentos sem precedentes, como o acesso à internet.

O mundo pode estar à beira da quarta revolução industrial, com impactos econômicos e sociais evidentes e consequências imprevisíveis no longo prazo (SCHWAB, 2016, p. 11), incluindo o funcionamento do Estado. A inteligência artificial (IA), big data, criptomoedas, robótica, impressoras 3D, internet das coisas (IoT) e nanotecnologia têm o potencial de impactar radicalmente a economia e a sociedade, tornando obsoleta grande parte das regras e instituições jurídicas hoje vigentes.

Segundo De Moraes (2023) a revolução digital tem impactado não somente a economia, mas também a tributação. As mudanças nas relações sociais e econômicas têm exigido a criação de novos impostos e formas de cobrança, tanto a nível nacional como internacional. Os sistemas tributários atuais não parecem ser totalmente eficazes para lidar com os desafios trazidos pela nova economia digital. As bases estabelecidas no século XX estão se tornando rapidamente obsoletas, incapazes de lidar com as novas práticas comerciais e modelos de negócios.

No contexto atual, embora a base tributária de renda ainda não tenha se tornado obsoleta, é evidente a tendência de redução do espaço nacional para ampliação da tributação do lucro corporativo. Esse fenômeno é causado por vários fatores interconectados, como a intensificação da globalização, a facilidade de fluxo de capitais e a alta mobilidade das

empresas multinacionais, especialmente as de tecnologia. A computação em nuvem, por exemplo, retira o caráter local dos negócios estabelecidos, tornando-os disponíveis para usuários em qualquer lugar do mundo, independentemente de um espaço nacional específico.

No atual cenário, a tributação sobre o emprego e a remuneração do trabalho, tanto individual quanto da folha de pagamento, está perdendo relevância devido à substituição crescente da mão de obra por máquinas e à flexibilização das formas de trabalho. As relações trabalhistas tradicionais estão sendo desconstruídas, com a diminuição do trabalho formal com carteira assinada, em que os impostos eram descontados na fonte e as contribuições eram feitas para a previdência social sobre os salários (De Moraes, 2023).

O aumento do desemprego estrutural, da informalidade, da transformação de empregados em empresas e do trabalho independente, impulsionado pela economia colaborativa e pelos novos modelos de negócios, está corroendo essa base tributária, afetando diretamente o custeio e a estrutura da previdência social (De Moraes, 2023).

Segundo De Moraes (2023) a economia digital traz desafios ao sistema tributário brasileiro, que parece ter esgotado seu potencial de expansão e está enfrentando mudanças estruturais. A eficiência arrecadatória, que antes era uma das principais qualidades do sistema tributário, agora está em dúvida. A carga tributária brasileira já foi próxima de 35% do PIB entre 2005 e 2008. Em 2017 estava abaixo de 33%. Já em 2022 a carga tributária bruta brasileira foi de 33,71% do PIB. Embora a recessão tenha um papel nesse declínio, há sinais claros de quebra estrutural no sistema tributário, com a retração de bases essenciais para a carga tributária atual, como petróleo, automóveis, comunicações, indústria de transformação e mesmo o emprego formal.

No Brasil, a modernização dos tributos enfrenta obstáculos específicos. Além das dificuldades enfrentadas em outros países, adaptar a tributação à economia digital no contexto brasileiro requer a superação de obstáculos relacionados ao desenho constitucional do sistema tributário e ao formalismo presente em nossa cultura jurídica. Para adequar a legislação tributária à nova economia digital, são necessárias mudanças legislativas importantes, o que implica em lidar com pelo menos três obstáculos: a rigidez constitucional, os conflitos federativos e a autonomia financeira dos entes subnacionais.

O primeiro obstáculo para a modernização dos tributos no Brasil está na falta de flexibilidade do sistema tributário, o qual tem bases rígidas, estabelecidas no próprio texto da Constituição Federal em vigor. Segundo FUCK (2017, p. 73-5) e TORRES (2010, p. 28), modernizá-lo, adaptando-o às inovações e urgências da economia digital, implica, em muitos aspectos, mudar a própria Constituição. Essa é uma peculiaridade nacional, uma vez que

nenhum outro país tem constituição tão detalhada e minuciosa em matéria tributária quanto a brasileira. A Constituição de 1988 define limites ao poder de tributar e direitos fundamentais dos contribuintes (art. 150), além de dispor exaustivamente sobre as competências tributárias (e.g. arts. 149, 153, 154, 155 e 156) e sobre partilha do produto da arrecadação de impostos (e.g. arts. 157, 158 e 159), alcançando tratamento especialmente minucioso no caso do ICMS (§ 2º do art. 155).

De acordo com De Moraes (2023) a adaptação do sistema tributário brasileiro à economia digital enfrenta o segundo obstáculo, relacionado à definição de competências, que está firmada no texto da Constituição de 1988. A tributação na nova economia exige escolher quem tributa o quê, o que requer uma revisão das premissas estabelecidas na década de 1960 e em 1988, que partiam de uma economia de bens tangíveis e repartiam competências entre Estados e Municípios, dividindo as operações entre circulação de mercadorias e prestação de serviços. Essas premissas são inadequadas para a nova economia digital.

O terceiro desafio está intimamente ligado aos dois primeiros e pode ser o mais difícil de ser superado. Se for necessário reformar a Constituição para adequar os tributos aos bens digitais e eliminar a antiga distinção entre mercadorias e serviços, como garantir a autonomia financeira dos estados e municípios? Em outras palavras, como garantir que esses entes tenham os recursos necessários para desempenhar suas funções institucionais? (De Moraes, 2023).

Qualquer reforma constitucional significativa que retire ou unifique as competências tributárias dos entes subnacionais precisará abordar essa questão (De Moraes, 2023).

4.5 REFORMA TRIBUTÁRIA (PEC 45/2019 EM TRAMITAÇÃO)

A Proposta de Emenda à Constituição (PEC) nº 45 de 2019 (ROSSI, 2019), conhecida como a "Reforma Tributária", foi apresentada no Congresso Nacional do Brasil com o objetivo de reformular o sistema tributário do país. A proposta visa simplificar e modernizar o sistema de impostos, unificando vários tributos em um único Imposto sobre Valor Agregado (IVA). Isso envolve a extinção de impostos como o ICMS, ISS, PIS e COFINS, substituindo-os pelo IVA.

Além disso, a PEC 45/2019 propõe mudanças na distribuição de recursos entre estados e municípios e estabelece um mecanismo de transição para minimizar os impactos da reforma. Essa iniciativa visa aprimorar a eficiência do sistema tributário brasileiro, tornando-o mais transparente e menos oneroso para as empresas (ROSSI, 2019).

4.6 DESAFIOS ENFRENTADOS NO CUMPRIMENTO DOS PRAZOS NO PAGAMENTO DOS IMPOSTOS E OBRIGAÇÕES ACESSÓRIAS

Bezerra, Oliveira e Araújo (2021) ressaltam a relevância do compliance tributário na gestão das obrigações acessórias, enfatizando que o desafio principal enfrentado pelas empresas reside na complexidade do sistema tributário brasileiro e na falta de transparência e clareza na interpretação da legislação tributária. Segundo os autores, a gestão das obrigações acessórias está intimamente ligada ao cumprimento das obrigações fiscais, sendo fundamental a implementação de medidas de compliance tributário para assegurar a conformidade fiscal das empresas.

No estudo de Lemos, Brito e Santos (2020), é destacada a elevada complexidade do sistema tributário brasileiro, juntamente com a falta de transparência por parte do Fisco e a carência de capacitação dos profissionais responsáveis pelo cumprimento dessas obrigações. Os autores também propõem estratégias para aprimorar o cumprimento dos deveres instrumentais, como a adoção de tecnologias e o aprimoramento da capacitação dos profissionais encarregados.

O cumprimento dos prazos estabelecidos para a entrega das obrigações fiscais e contábeis no Brasil é uma tarefa complexa devido a vários fatores.

Primeiramente, o Brasil possui um dos sistemas tributários mais complexos e burocráticos do mundo, com mais de 377 mil normas tributárias editadas após a promulgação da Constituição Federal, em 1988³. Isso torna a gestão das obrigações fiscais e contábeis um desafio, pois os contribuintes precisam estar constantemente atualizados sobre as mudanças na legislação tributária e os prazos de vencimento (TCU, 2015).

Outro desafio é a falta de organização e controle sobre as obrigações fiscais e contábeis. Muitas empresas não possuem um sistema eficaz para rastrear e gerenciar suas obrigações fiscais e contábeis, o que pode levar ao atraso na entrega das informações (GÜTSCHOW, 2023).

5 PROCEDIMENTOS METODOLÓGICOS

No que diz respeito à metodologia utilizada para essa monografia, foi realizada uma extensa pesquisa em documentos, artigos e portais de informações relacionados a tributação, para entender melhor como funciona o cenário tributário brasileiro, e como é a nossa carga tributária, visando descrever a alta complexibilidade e difícil gestão das entregas tributárias a serem feitos pelas empresas.

Por conta dessa alta complexidade e difícil gestão, surge uma grande oportunidade de melhoria e implementação de ferramentas a fim de facilitar a vida das pessoas envolvidas nesse processo, e com isso surge a proposta de um software para gestão dessas entregas.

A premissa do software a ser desenvolvido engloba a previsibilidade e a organização eficiente dos tributos a serem entregues através de métodos de organização ágeis. Para isso será desenvolvido um Kanban para organização das entregas e a criação de alguns relatórios para proporcionar a previsibilidade necessária nas entregas. Além de permitir um controle por Empresa, Filial ou Grupo e uma abertura por contador, proporcionando um experiência mais “maleável” e que atenda as necessidades específicas de cada usuário.

6 TECNOLOGIAS ENVOLVIDAS

Este capítulo se dedica à exploração e explicação das tecnologias e padrões adotados no desenvolvimento deste protótipo. Ao longo das próximas seções, serão apresentadas as escolhas fundamentais que orientaram a arquitetura e o design do sistema. Este exame detalhado proporcionará uma compreensão abrangente do contexto tecnológico subjacente, fornecendo uma base sólida para a análise crítica do desenvolvimento e para futuras implementações e melhorias.

6.1 KANBAN

A gestão visual constitui o elemento central do sistema Kanban, cuja concepção visa, primordialmente, viabilizar a produção "Just in Time" mediante o controle visual dos processos produtivos e dos estoques. A metodologia emprega cartões visuais para sinalizar a necessidade de reposição de materiais e produtos, sendo que o termo "kanban" em japonês traduz-se como "cartão visual" (OHNO, 1997). Consoante Rocha e Sousa (2021), o Sistema Kanban representou uma significativa evolução no controle de produção, permitindo a visualização das informações referentes à movimentação e geração de produtos ao final do processo. Tal abordagem culmina na produção Just in Time, isto é, na entrega pontual e na quantidade necessária, resultando na minimização de estoques, custos e desperdícios.

No contexto do Kanban, cartões são empregados para autorizar tanto a produção quanto a movimentação dos itens ao longo do processo produtivo. Os princípios orientadores do Kanban abrangem a visualização do trabalho em progresso, a identificação contínua de melhorias, fomentando uma cultura kaizen, onde a responsabilidade por aprimoramentos é compartilhada por todos, e a explicitação das políticas adotadas, com a visualização completa dos passos, desde o conceito geral até o software ou produto passível de lançamento. Ademais, destaca-se a importância de mensurar e gerenciar o fluxo para embasar decisões sólidas e antecipar as futuras consequências de cada escolha (BOEG, 2011; MACHADO JUNIOR et al., 2019).

6.2 ARQUITETURA

Estrutura MVC (Model-View-Controller) é um padrão de arquitetura de software amplamente utilizado para desenvolver aplicativos da web. Consiste em quatro camadas distintas, cada uma com sua função específica:

View: A camada de visualização é a parte do aplicativo que lida com a interface do usuário. Ela contém as telas e formulários que serão renderizados e exibidos para os usuários. A view é responsável por apresentar os dados de forma compreensível e interativa.

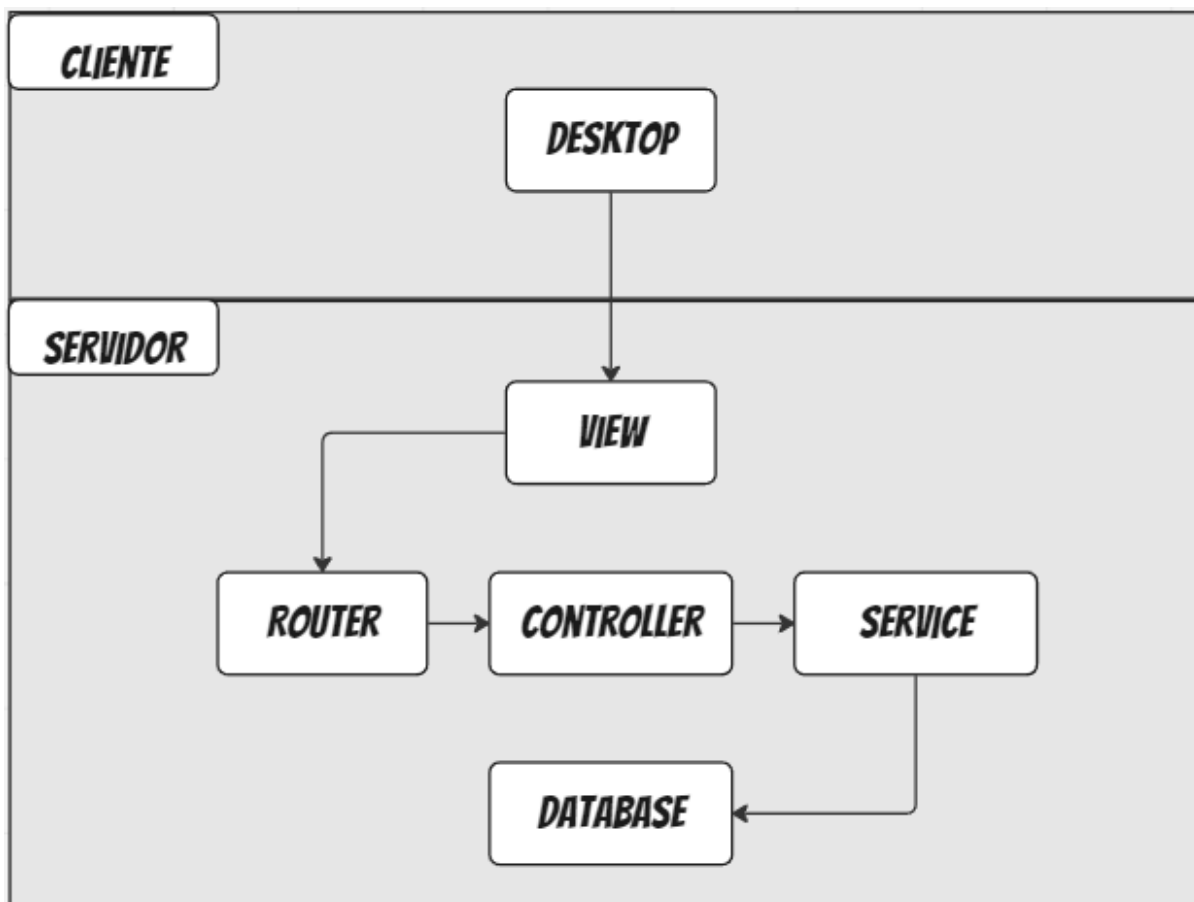
Router: A camada do roteador desempenha um papel vital na gestão das solicitações dos usuários. Ela mapeia URLs (Uniform Resource Locators) para ações específicas, ou seja, para funções das controllers. Por exemplo, quando um usuário acessa a URL "meu-dominio/contador", o roteador associa essa URL a uma função específica no controller responsável por controlar e persistir as alterações nos contadores.

Controller: A camada controller é responsável por processar as solicitações dos usuários e coordenar a lógica de negócios do aplicativo. Ela gerencia os elementos da camada service e passa as informações solicitadas pelos clientes para a camada de visualização. O controller atua como um intermediário que conecta a interface do usuário à lógica subjacente do aplicativo.

Service: A camada de serviço é onde ocorre o armazenamento e recuperação de informações. Ela se comunica com o banco de dados e é responsável por realizar operações de persistência, como salvar, recuperar ou modificar dados. A camada de serviço separa a lógica de acesso a dados do restante do aplicativo, garantindo a modularidade e a manutenção mais fácil.

Em resumo, o padrão MVC divide o desenvolvimento de aplicativos web em camadas distintas, permitindo uma melhor organização do código, reutilização de componentes e manutenção mais eficiente. A view lida com a interface do usuário, o roteador gerencia as URLs e as ações, o controller processa solicitações e a camada de serviço cuida do armazenamento e recuperação de dados. Isso resulta em um código mais organizado e uma estrutura mais escalável para aplicativos web.

Figura 01: Exemplificação da arquitetura adotada no projeto



Fonte: elaborado pelo autor

A figura acima ilustra o funcionamento das camadas, onde o cliente através do seu desktop, acessa a camada “view” de visualização a qual corresponde a interface gráfica que por sua vez realiza as requisições ao “router” responsável por receber as requisições feitas, que passa pelo “controller” o qual faz todo o controle e validações necessárias e que acessa a camada “service” a qual possui conexão com o banco de dados.

6.3 TECNOLOGIAS USADAS

Neste capítulo, exploraremos as tecnologias essenciais que foram empregadas no processo de desenvolvimento do protótipo. As ferramentas fundamentais para a concretização desse projeto incluem Delphi, JavaScript, UML (Unified Modeling Language) e PostgreSQL.

Cada uma dessas tecnologias desempenha um papel crucial, proporcionando um conjunto robusto e integrado de recursos para a concepção e implementação eficiente do protótipo.

6.3.1 Docker

O artigo de Fernando Furtado, publicado em 2017, aborda o Docker como um sistema de virtualização não convencional em comparação com abordagens convencionais de virtualização, como aquelas encontradas no VirtualBox, VMWare e Parallels. Nas virtualizações convencionais, um software instalado no host gerencia máquinas virtuais, cada uma contendo uma instalação completa do sistema operacional e virtualizando seu próprio hardware.

O Docker difere ao adotar o conceito de containers. O autor compara esse conceito ao transporte de cargas, destacando a revolução que os containers trouxeram para essa área, tornando o carregamento e descarregamento eficientes e eliminando problemas como perdas e trabalho manual extenso. O Docker busca aplicar esse princípio aos softwares, permitindo o transporte seguro e eficiente de aplicações entre diferentes ambientes.

Ao utilizar containers do Docker, o autor destaca os benefícios, especialmente em cenários como a transferência de software do ambiente de desenvolvimento para produção. Ele destaca que, ao encapsular o software em um container, é possível transportá-lo para outro ambiente sem se preocupar com a instalação de pré-requisitos, versões do sistema operacional, permissões e outros detalhes. Isso reduz significativamente as disparidades entre os ambientes, proporcionando uma abordagem mais consistente e eficaz para o gerenciamento de software em diferentes contextos.

6.3.1.1 Container e imagem

O artigo da AWS (2023) discute as tecnologias de implantação de aplicações oferecidas pelo Docker, centrando-se em imagens e contêineres. Antes, a execução de aplicações exigia a instalação da versão correspondente ao sistema operacional da máquina, mas o Docker permite criar contêineres que funcionam em diferentes dispositivos e sistemas

operacionais. O Docker é uma plataforma que empacota software em contêineres, sendo as imagens modelos somente leitura que contêm instruções para criar um contêiner.

Os contêineres do Docker podem ser executados em qualquer máquina com o mecanismo do Docker instalado, independentemente da arquitetura subjacente do sistema. Um contêiner é um ambiente de runtime que inclui todos os componentes necessários para executar uma aplicação, sem depender das bibliotecas da máquina host. Para implantar e escalar contêineres de maneira eficaz, especialmente em ambientes distribuídos, é recomendado o uso de uma plataforma de orquestração como o Kubernetes.

Uma imagem do Docker é um arquivo executável independente usado para criar um contêiner. Essa imagem é portátil e compartilhável, permitindo a implantação em vários locais simultaneamente. Além disso, o artigo destaca a possibilidade de armazenar imagens em registros, como o Docker Hub, que contém uma variedade de imagens para sistemas operacionais, linguagens de programação, bancos de dados e editores de código.

6.3.2 Banco de Dados

A expressão "Banco de Dados" teve sua origem no termo inglês "Databanks", o qual foi substituído pela palavra "Databases" - ou "Base de Dados" - por ser mais apropriada (SETZER; CORRÊA DA SILVA, 2005, p. 1).

No que concerne à definição de um banco de dados, DATE (2004, p. 10) afirma que se trata de "uma coleção de dados persistentes, usada pelos sistemas de aplicação de uma determinada empresa". Em outras palavras, um banco de dados consiste em um repositório onde dados necessários para as operações de uma organização são armazenados, servindo como fonte de informações para as aplicações atuais e futuras.

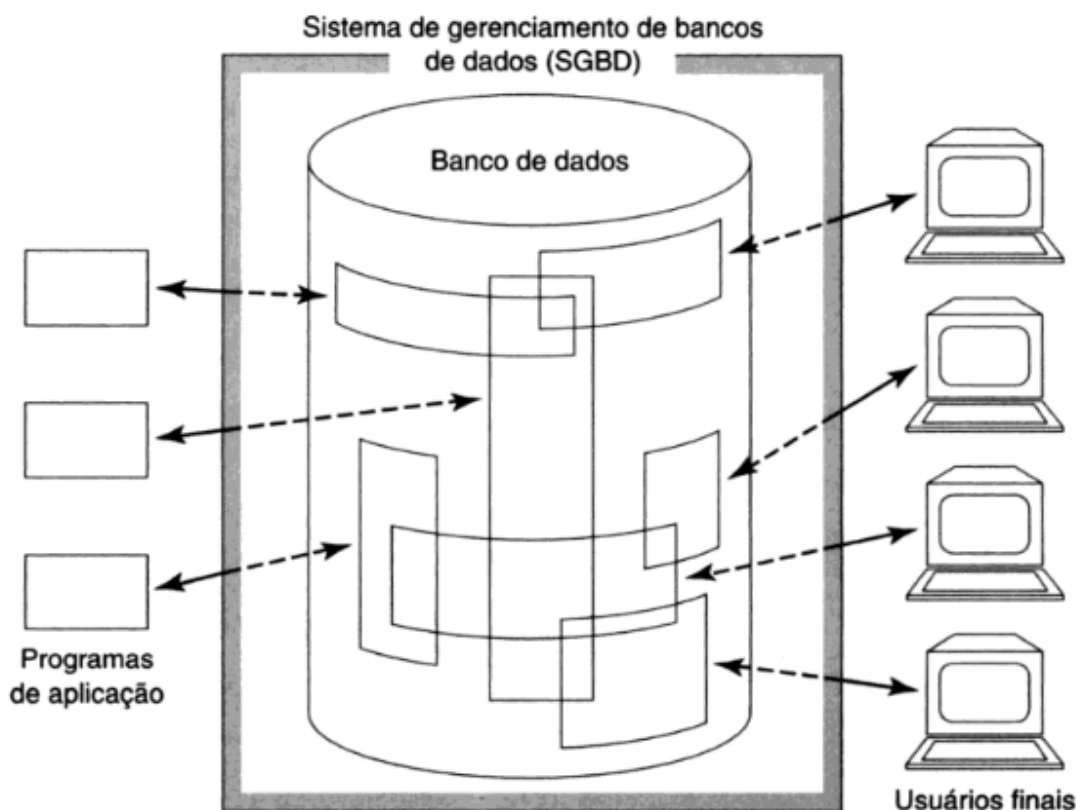
De acordo com ELMASRI e NAVATHE (2011, p. 3), a expressão "Banco de Dados" implica nas seguintes propriedades:

Um banco de dados representa algum aspecto do mundo real, às vezes chamado de minimundo ou de universo de discurso (UoD – Universe of Discourse). As mudanças no minimundo são refletidas no Banco de Dados. Um banco de dados é uma coleção logicamente coerente de dados com algum significado inerente. Uma variedade

aleatória de dados não pode ser corretamente chamada de banco de dados. Um banco de dados é projetado, construído e populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados.

Conforme DATE (2004, p. 6), um sistema de banco de dados é definido como "um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitarem". O autor enfatiza que um sistema de banco de dados é constituído por elementos como dados, hardware, software e usuários.

Figura 02: Ilustração de um SGBD



Fonte: DATE, 2004

Nesse contexto, o termo "Dados" se refere ao próprio conteúdo do banco de dados, ou seja, ao conjunto de informações. É relevante, no entanto, apresentar uma conceituação prévia do termo "dado".

Conforme SETZER e CORRÊA DA SILVA (2005, p. 2), o termo "dado" é definido como "uma representação simbólica, ou seja, feita por meio de símbolos, quantificada ou

quantificável". Além disso, ELMASRI e NAVATHE (2011, p. 3) complementam essa definição, afirmando que os dados são "fatos conhecidos que podem ser registrados e possuem significado implícito".

Portanto, um exemplo de dado seria o número de matrícula de um funcionário, visto que é um fato conhecido para a empresa, possui uma semântica, é representado por símbolos (algarismos de 0 a 9, seguindo o sistema numérico de base 10, logo é quantificável) e, portanto, pode ser registrado.

O Hardware compreende os elementos físicos que compõem o sistema de banco de dados, como as mídias de armazenamento, os canais de entrada/saída, entre outros.

Já o software, segundo DATE (2004, p. 8), o Sistema Gerenciador de Banco de Dados (SGBD) é um conjunto de programas que atua como intermediário entre o banco de dados armazenado e os usuários. Conforme SILBERSCHATZ, KORTH e SUDARSHAN (1999, p. 1), o principal objetivo de um SGBD é fornecer um ambiente conveniente e eficiente para a recuperação e armazenamento das informações do banco de dados.

Para alcançar esse propósito, o Sistema Gerenciador de Banco de Dados oferece recursos que englobam a definição, construção, manipulação, compartilhamento, proteção e manutenção de bancos de dados, como afirmado por ELMASRI e NAVATHE (2011, p. 3). A definição refere-se à especificação das estruturas de armazenamento, incluindo os elementos e os tipos de dados que compõem os registros. A construção envolve o armazenamento dos dados, abrangendo registros e relacionamentos. A manipulação abarca a recuperação e a atualização de dados, como inclusões, exclusões e alterações. O compartilhamento diz respeito à permissão de acesso simultâneo a um banco de dados. A proteção envolve a segurança contra falhas de hardware, software e acessos não autorizados. Por fim, a manutenção compreende o suporte para o crescimento contínuo do banco de dados (ELMASRI; NAVATHE, 2011, p. 4).

6.3.2.1 PostgreSQL

O PostgreSQL é um sistema de base de dados de código aberto que se destaca por suas extensas funcionalidades. Originariamente desenvolvido no ambiente acadêmico, conseguiu estabelecer uma ampla comunidade de programadores ao longo de sua duradoura

história. De acordo com Pavel Luzanov, Egor Rogov, Igor Levshin (traduzido por Liudmila Mantrova) em seu artigo "PostgreSQL: The First Experience," o PostgreSQL atualmente oferece todas as funcionalidades necessárias para a maioria dos usuários e é amplamente utilizado em todo o mundo na criação de sistemas críticos de alta carga.

O PostgreSQL demonstra eficiência ao explorar a arquitetura moderna de CPUs multi-core, com um desempenho que cresce de forma praticamente linear à medida que o número de núcleos aumenta. De acordo com Pavel Luzanov, Egor Rogov, Igor Levshin (traduzido por Liudmila Mantrova) em seu artigo "PostgreSQL: The First Experience," o PostgreSQL possui a capacidade de paralelizar consultas e alguns comandos, como a criação de índices e a aspiração. Nesse modo, operações de leitura e junção são executadas por vários processos simultâneos, e a compilação just-in-time (JIT) de consultas contribui para acelerar as operações, aproveitando melhor os recursos de hardware. A cada nova versão, o PostgreSQL introduz aprimoramentos adicionais na capacidade de paralelização.

No que diz respeito ao escalonamento horizontal, o PostgreSQL oferece opções de replicação física e lógica, permitindo a construção de clusters baseados em PostgreSQL que oferecem alto desempenho, tolerância a falhas e geodistribuição.

6.3.3 Linguagem de programação

Uma linguagem de programação é um método padronizado utilizado para expressar instruções em um programa destinado a um computador programável, seguindo um conjunto de regras sintáticas e semânticas para definir o referido programa. As regras sintáticas estão relacionadas à forma de escrita, enquanto as regras semânticas dizem respeito ao conteúdo. De acordo com Reginaldo Gotardo em "Linguagem de Programação I," publicado no Rio de Janeiro em 2015, por meio da especificação de uma linguagem de programação, é possível determinar os dados que o computador utilizará, como esses dados serão tratados, armazenados e transmitidos, e quais ações devem ser executadas em diferentes circunstâncias.

Quando se utiliza uma linguagem de programação, é gerado o chamado "Código Fonte", que consiste em um conjunto de palavras escritas em conformidade com as regras sintáticas e semânticas da linguagem.

6.3.3.1 Delphi

O Delphi é uma ferramenta de desenvolvimento de aplicativos baseada na linguagem Object Pascal, originalmente desenvolvida pela empresa Borland Software Corporation em 1995. Ele foi concebido sob o paradigma RAD (Rapid Application Development), inicialmente voltado para o desenvolvimento desktop. No entanto, atualmente, o Delphi oferece um ambiente de desenvolvimento mais integrado, permitindo a edição de códigos, teste, depuração, correção de erros e compilação para execução em diversos sistemas operacionais.

Nas versões mais recentes, mantidas pela empresa Embarcadero Technologies, é possível utilizar o mesmo código-fonte para compilar aplicativos que rodem em várias plataformas, incluindo Windows, macOS, iOS, Android e Linux (EMBARCADERO, 2023).

6.3.3.2 JavaScript

Conforme Flanagan (2013), o JavaScript teve sua origem na Netscape, nos primórdios da Web, e é formalmente uma marca registrada licenciada pela Sun Microsystems (agora Oracle). Inicialmente, a Netscape enviou a linguagem para a ECMA (European Computer Manufacturer's Association) para padronização, o que levou à denominação "ECMAScript" devido a questões de marca registrada. A Microsoft também desenvolveu sua versão chamada "JScript". No entanto, a linguagem é popularmente conhecida como JavaScript.

De acordo com Flanagan (2013), o JavaScript não fornece funcionalidades de entrada e saída diretamente, pois essas responsabilidades são delegadas ao ambiente hospedeiro, geralmente um navegador da Web.

6.3.4 UML

A UML (Unified Modeling Language) é uma família de notações gráficas apoiada por um metamodelo único, que auxilia na descrição e no projeto de sistemas de software, especialmente aqueles desenvolvidos com o estilo orientado a objetos (OO). Segundo (FOWLER, 2014), a UML desempenha um papel fundamental na engenharia de software, embora sua definição possa variar de acordo com diferentes perspectivas e históricos, uma vez que várias linguagens gráficas de modelagem têm sido usadas na indústria de software.

A necessidade de linguagens gráficas de modelagem surge da limitação das linguagens de programação em fornecer um nível de abstração adequado para discutir projetos de software. Apesar de sua longa existência, a utilização dessas linguagens na indústria de software gera controvérsias que afetam a percepção do papel da UML.

A UML é um padrão relativamente aberto, controlado pelo OMG (Object Management Group), um consórcio de empresas que busca estabelecer padrões para sistemas orientados a objetos, sendo também conhecido por seus padrões, como o CORBA (Common Object Request Broker Architecture). Ela surgiu da unificação das diversas linguagens gráficas de modelagem orientadas a objetos que surgiram nas décadas de oitenta e noventa, solucionando uma espécie de "Torre de Babel" na área de modelagem de sistemas. (FOWLER, 2014) destaca a importância desse serviço, pelo qual muitos desenvolvedores são profundamente gratos.

6.3.5 Ferramentas de desenvolvimento

Neste tópico, serão discutidas as ferramentas empregadas durante o processo de criação do protótipo, destacando suas funções, importância e contribuição para o desenvolvimento do projeto. Exploraremos como essas ferramentas desempenham um papel crucial na concepção, teste e aprimoramento do protótipo, proporcionando uma compreensão abrangente do ambiente de desenvolvimento utilizado. Além disso, será enfatizado como a escolha criteriosa e a integração eficaz dessas ferramentas podem influenciar positivamente a eficiência e a qualidade do protótipo resultante.

6.3.5.1 RAD Studio

Segundo a Embarcadero (2023), o RAD Studio é conhecido por sua capacidade de desenvolver aplicativos nativos para várias plataformas, incluindo Windows, macOS, Android e iOS, usando uma única base de código. Ele oferece suporte a várias linguagens de programação, incluindo Delphi e C++, e fornece componentes e bibliotecas para ajudar os desenvolvedores a criar aplicativos com uma ampla gama de recursos e funcionalidades.

- Desenvolvimento multiplataforma: Permite aos desenvolvedores criar aplicativos para várias plataformas a partir de uma única base de código.
- Ambiente de desenvolvimento integrado: Oferece uma IDE poderosa com ferramentas de design visual, depuração e teste.
- Componentes e bibliotecas: Fornece um conjunto rico de componentes e bibliotecas para acelerar o desenvolvimento e adicionar funcionalidades aos aplicativos.
- Conectividade a bancos de dados: Oferece suporte a uma ampla variedade de bancos de dados, facilitando a criação de aplicativos com recursos de banco de dados.
- Desenvolvimento de aplicativos móveis: Permite criar aplicativos móveis nativos para Android e iOS.
- Desenvolvimento de aplicativos desktop: Permite criar aplicativos nativos para Windows e macOS.
- Desenvolvimento de aplicativos web: Permite criar aplicativos da web com Delphi e C++.
- Suporte a IoT (Internet das Coisas): Oferece recursos para desenvolver aplicativos para dispositivos IoT.
- Gerenciamento de ciclo de vida de software: Fornece ferramentas para gerenciar todo o ciclo de vida do desenvolvimento de software.

6.3.5.2 Visual Studio Code

O Visual Studio Code (VS Code) é um editor de código-fonte desenvolvido pela Microsoft e amplamente utilizado por desenvolvedores de software e programadores para escrever, depurar e editar código de diversas linguagens de programação. É uma ferramenta de código aberto e gratuita que oferece uma ampla gama de recursos e extensões que auxiliam os desenvolvedores em suas tarefas diárias.

O Visual Studio Code é conhecido por sua alta flexibilidade e extensibilidade, permitindo que os usuários personalizem o ambiente de desenvolvimento de acordo com suas necessidades específicas. Ele oferece suporte a diversas linguagens de programação, integração com controle de versão, depuração de código, sugestões de código (intellisense), entre outras funcionalidades. Além disso, é uma ferramenta multiplataforma, disponível para Windows, macOS e Linux, o que o torna acessível a uma ampla variedade de desenvolvedores.

7 ANÁLISE E IMPLEMENTAÇÃO DO PROTÓTIPO

No processo de implementação da presente monografia de conclusão de curso, foram empregadas as seguintes tecnologias e metodologias: a linguagem de programação Delphi foi escolhida para desenvolver a parte do front-end da aplicação, enquanto a linguagem JavaScript foi adotada para o desenvolvimento do back-end. Além disso, a linguagem de modelagem UML desempenhou um papel fundamental na definição da estrutura e do fluxo do sistema.

A abordagem adotada no desenvolvimento desta monografia foi embasada em extensas pesquisas bibliográficas, que permitiram compreender as melhores práticas e conceitos relevantes para a construção do protótipo de um software para gestão contábil e fiscal.

7.1 DESCRIÇÃO DO PROTÓTIPO

O eGerir nasce com a premissa de ser um protótipo de software inovador para a gestão eficiente de entregas fiscais e contábeis. A plataforma terá uma abordagem visual utilizando o método Kanban para a gestão de tarefas, sendo projetada para atender a diferentes entidades, desde grupos de empresas até contadores individuais. Com dashboards personalizados e recursos de colaboração, o eGerir visa simplificar e aprimorar a gestão contábil e fiscal, proporcionando uma experiência fluida e transparente para os usuários.

7.2 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

A seguir, apresentam-se os requisitos funcionais e não funcionais do software de gestão contábil e fiscal, delineando as funcionalidades que os usuários requerem que o sistema desempenhe.

7.2.1 Requisitos funcionais

O conjunto de funcionalidades propostas pelo sistema é essencial para atender às necessidades do contratante e dos usuários da melhor maneira possível. Essas funcionalidades representam as ações e capacidades que o software oferece para resolver os desafios específicos. Sabendo disso os requisitos funcionais levantados são os seguintes:

- Controle por Grupo de Empresa:
 - Cadastro e manutenção de grupo de empresas
 - Essa configuração irá permitir uma visão mais sintética dos dados apurados nos dashboards por grupo. Para atender os diferentes níveis de organização e hierarquia de informação
- Controle por Empresa:
 - Cadastro e manutenção de empresas.
 - Essa configuração irá permitir uma visão das entregas realizadas por empresa. Para atender os diferentes níveis de organização e hierarquia de informação.
- Controle por Filial:
 - Cadastro e manutenção de filiais.
 - Essa configuração irá permitir uma visão mais analítica das entregas realizadas a fim de proporcionar uma abertura por filial.
- Controle por contador:
 - Cadastro e manutenção dos contadores do sistema.
 - Esse controle permitirá uma visão mais analítica dos dados a fim de gerar indicadores de entrega por contador.
- Gerenciamento das demandas/entregas:
 - Cadastro e manutenção das entregas/atividades a serem desenvolvidas.
 - Gestão das demandas através de status de andamento e informações de entrega.
- Relatórios apurando as entregas realizadas e não realizadas:
 - Relatório de entregas realizadas por grupo.
 - Relatório de entregas realizadas por empresa.
 - Relatório de entregas realizadas por filial.
 - Relatório de entregas realizadas por contador.

- Índice de entregas realizado x esperado.

7.2.2 Requisitos não funcionais

Os requisitos não funcionais, também conhecidos como requisitos de qualidade, são especificações que descrevem atributos ou características do sistema de software que não estão diretamente relacionados às funcionalidades específicas do sistema, mas que são fundamentais para a qualidade, desempenho, segurança e usabilidade do software. Eles complementam os requisitos funcionais, que se concentram no que o sistema deve fazer. Os requisitos não funcionais abordam como o sistema deve fazer essas tarefas e como deve se comportar em vários aspectos. Sabendo disso os requisitos não funcionais são os seguintes:

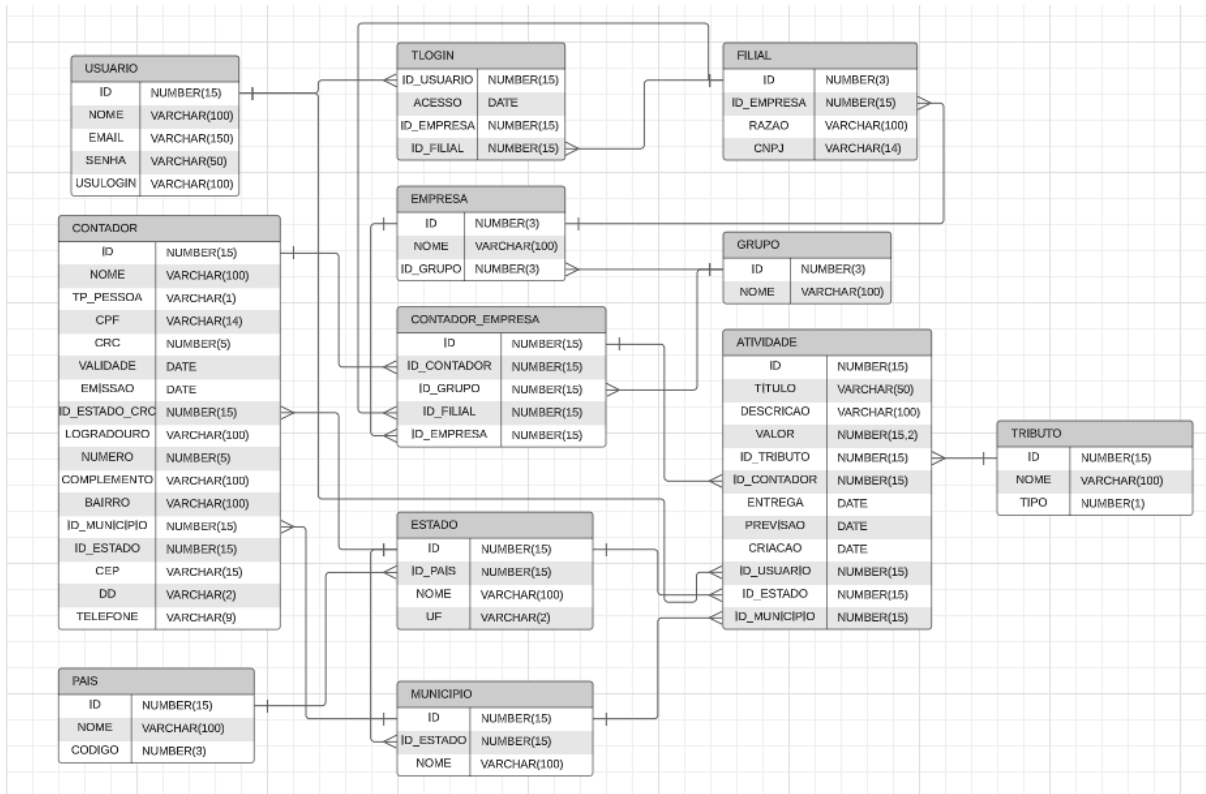
- **Praticidade:** A praticidade de um software muitas vezes se revela como um facilitador essencial para a execução de diversas tarefas. A capacidade de simplificar processos complexos e proporcionar uma experiência de usuário intuitiva torna-se evidente em soluções tecnológicas modernas. Essa praticidade se manifesta através de interfaces amigáveis e respostas rápidas, permitindo que os usuários alcancem seus objetivos de maneira eficaz. A agilidade proporcionada por um software prático não apenas economiza tempo, mas também promove uma integração suave em diferentes contextos, contribuindo para a eficiência operacional e a satisfação do usuário.
- **Escalabilidade:** Um sistema escalável é projetado para lidar com um aumento significativo no volume de dados, usuários ou operações sem comprometer seu desempenho ou funcionalidade. Essa característica é crucial para garantir que o software possa evoluir de forma flexível, atendendo às necessidades em constante mudança do ambiente em que está inserido.
- **Segurança:** O software deve incorporar diversas práticas de segurança para garantir a integridade, confidencialidade e disponibilidade das informações. A abordagem proativa em relação à segurança é fundamental para mitigar riscos e garantir que os usuários possam confiar no software para proteger suas informações de maneira eficaz.

- Desempenho: O desempenho de um software desempenha um papel crucial na experiência do usuário, influenciando diretamente a eficiência e a eficácia das operações realizadas. Quando um programa opera de maneira ágil e responsiva, os usuários podem executar tarefas de forma mais eficiente, resultando em maior produtividade e satisfação.

7.3 DIAGRAMA DE ENTIDADE E RELACIONAMENTO

Segundo Nogueira (1988), um diagrama entidade relacionamento (ER) é um tipo de fluxograma que ilustra como "entidades", que podem ser pessoas, objetos ou conceitos, se relacionam entre si dentro de um sistema. Genong et al. (2010) e Kawabata (2015) destacam que os Diagramas ER são amplamente utilizados para o projeto e depuração de bancos de dados relacionais em campos como Engenharia de Software, sistemas de informações empresariais, educação e pesquisa. Esses diagramas, também conhecidos como DERs ou modelos ER, empregam um conjunto específico de símbolos, como retângulos, diamantes, ovais e linhas de conexão, para representar a interconectividade entre entidades, relacionamentos e seus atributos (Sordi et al., 2009; Genong et al., 2010). Conforme destacado por Silva et al. (2008), os diagramas ER refletem estruturas gramaticais, onde as entidades são tratadas como substantivos e os relacionamentos como verbos.

Figura 03: Exemplificação do diagrama ER do projeto



Fonte: elaborado pelo autor

A ilustração apresentada na Figura 3 oferece um exemplo prático da aplicação do diagrama de entidade e relacionamento no contexto do software em fase de desenvolvimento.

7.4 DIAGRAMA DE CLASSE

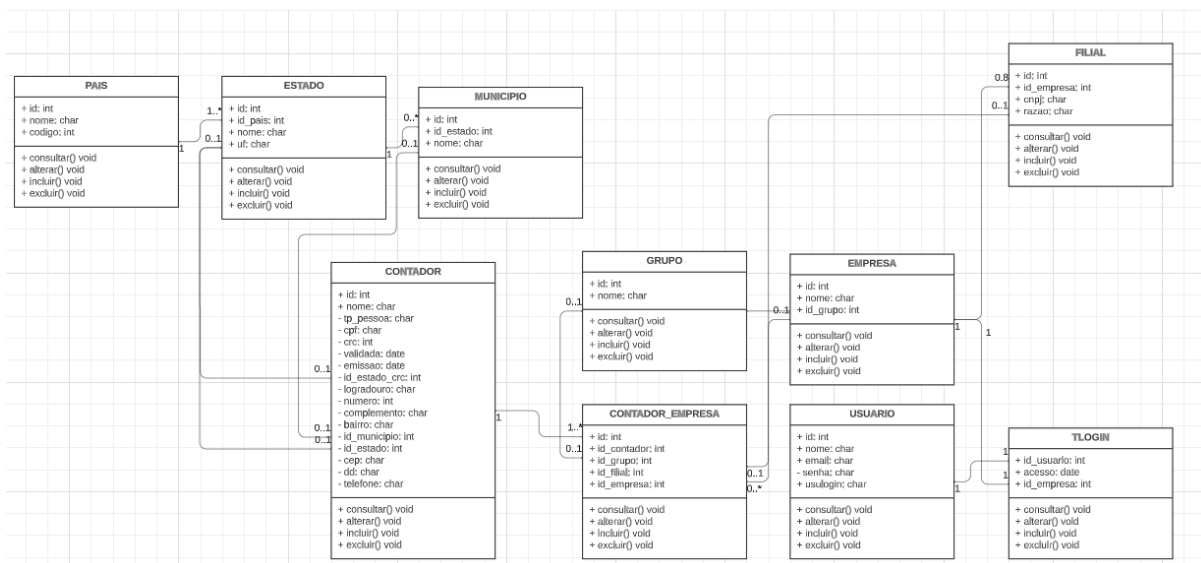
Os diagramas de classe são uma parte fundamental da UML, uma linguagem de modelagem, e são utilizados para representar a estrutura estática de sistemas. Dependendo da complexidade do sistema, é possível utilizar um único diagrama de classe para modelar o sistema inteiro ou vários diagramas para representar componentes individuais. Eles descrevem os objetos que compõem o sistema, seus relacionamentos e suas funcionalidades (“Diagramas de Classes”, 2021).

Esses diagramas desempenham um papel crucial em várias etapas do processo de desenvolvimento de sistemas, desde a análise até a implementação. No estágio de análise, ajudam a entender os requisitos do problema e a identificar componentes-chave. Durante o projeto de software orientado a objetos, os diagramas de classe evoluem para classes e

objetos reais. Posteriormente, são usados para capturar o funcionamento preciso do sistema e os relacionamentos entre componentes em diferentes níveis (“Diagramas de Classes”, 2021).

Além disso, os diagramas de classe servem para visualizar, especificar e documentar a estrutura dos modelos, incluindo classes, relacionamentos, atributos e operações. Eles podem ser usados em várias fases do desenvolvimento de software, desde a criação dos modelos até a conversão dos modelos em código e vice-versa (“Diagramas de Classes”, 2021).

Figura 04: Exemplicação de diagrama de classe produzido para o projeto



Fonte: elaborado pelo autor

A figura 4 apresentada anteriormente ilustra a aplicação de um diagrama de classe no processo de desenvolvimento de software. Este tipo de representação visual é essencial para descrever a estrutura e as relações entre as classes presentes no sistema em desenvolvimento.

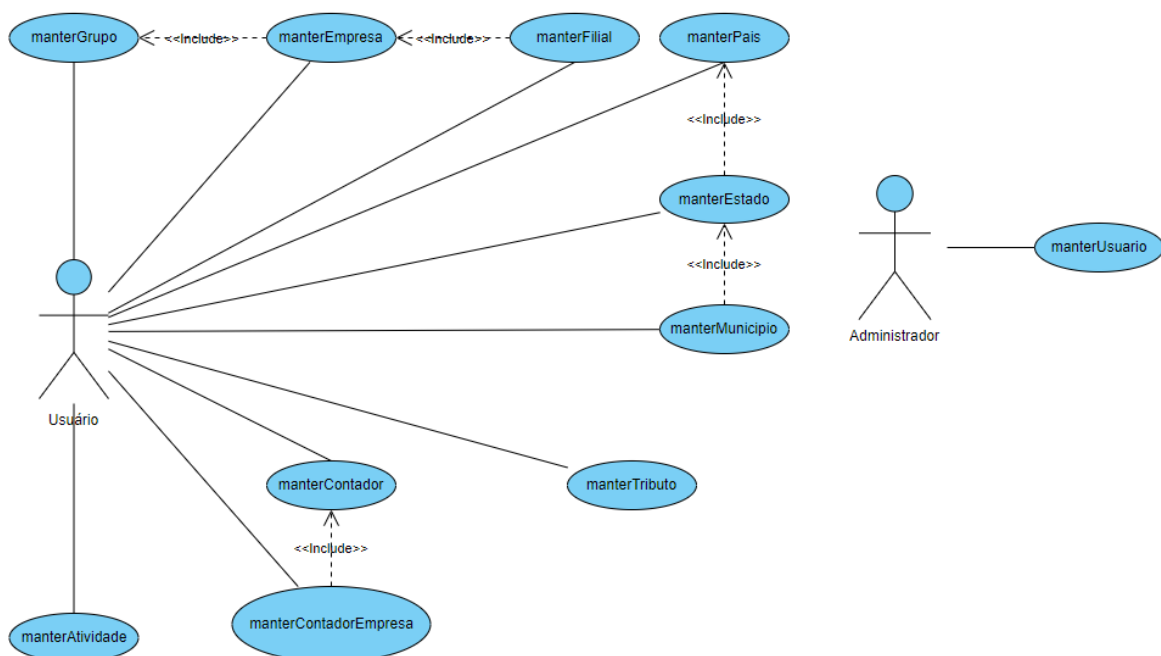
7.5 DIAGRAMA DE CASO DE USO

Os diagramas de caso de uso na UML são usados para modelar o comportamento de um sistema e capturar seus requisitos. Eles descrevem as funções de alto nível do sistema e suas interações com os agentes, sem entrar nos detalhes da operação interna do sistema. Os diagramas de caso de uso podem ser usados para representar sistemas completos ou partes importantes deles, sendo úteis em diversas fases do desenvolvimento de projetos (“Diagramas de Caso de Uso”, 2021).

Eles desempenham um papel crucial em várias situações, como a modelagem de negócios antes do início do projeto, a coleta de requisitos, a identificação de classes durante as fases de análise e design, e a definição de testes durante a fase de teste. Os principais elementos em um diagrama de caso de uso incluem casos de uso, agentes e subsistemas, que representam as funções do sistema, os usuários que interagem com ele e as unidades comportamentais independentes (“Diagramas de Caso de Uso”, 2021).

Além disso, os relacionamentos em diagramas de caso de uso desempenham um papel crucial na definição das conexões entre elementos de modelo, enriquecendo o entendimento do sistema e sua estrutura (“Diagramas de Caso de Uso”, 2021).

Figura 05: Exemplificação de diagrama de caso de uso



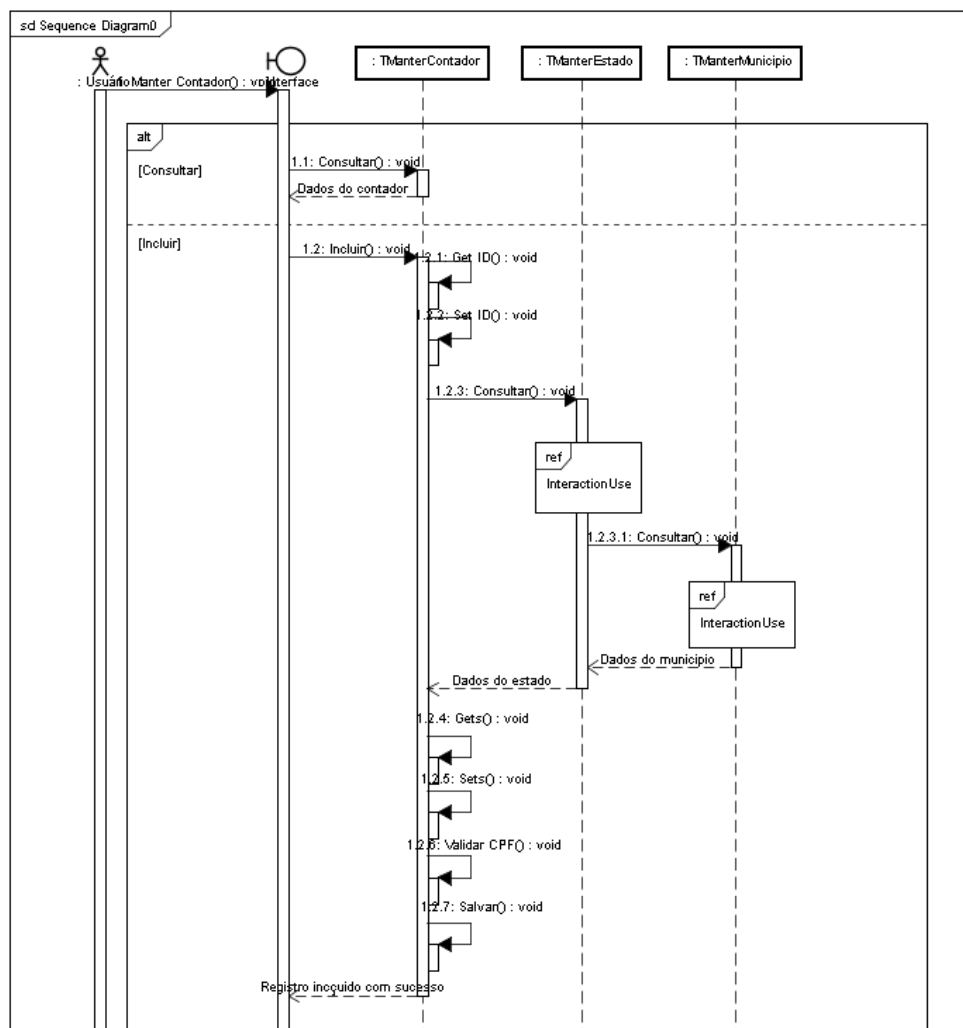
Fonte: elaborado pelo autor

A imagem apresentada na Figura 5 ilustra a aplicação do diagrama de caso de uso no contexto do desenvolvimento de software. Este diagrama é uma representação visual que descreve as interações entre os atores (usuários ou sistemas externos) e as funcionalidades do sistema em questão. Ele oferece uma visão clara e concisa de como os diferentes elementos do software se relacionam, contribuindo para uma compreensão abrangente do sistema em desenvolvimento.

7.6 DIAGRAMA DE SEQUÊNCIA

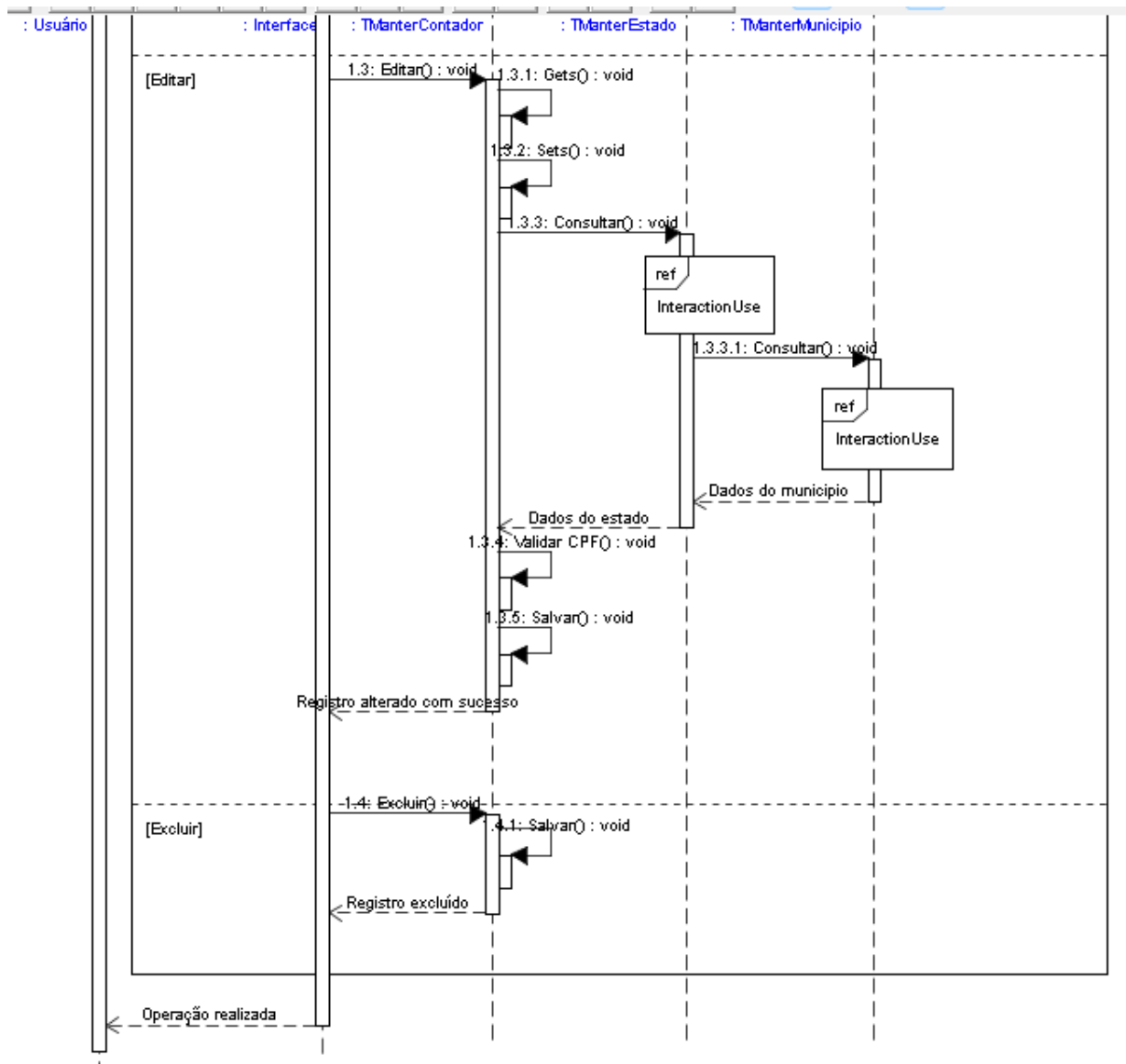
Um diagrama de sequência UML ilustra a ordem das mensagens trocadas entre objetos em uma interação. Ele consiste em objetos representados por linhas de vida e as mensagens que eles compartilham durante a interação. O diagrama descreve a sequência das mensagens e as estruturas de controle entre os objetos, facilitando a compreensão das interações. Linhas de vida representam os objetos envolvidos, mensagens indicam as comunicações e fragmentos combinados são usados para agrupar lógica condicional que afeta o fluxo de mensagens. Além disso, os usos de interação permitem referenciar interações existentes para criar sequências mais complexas a partir de interações menores (“Diagramas de Sequência”, 2021).

Figura 06: Exemplicação de diagrama de sequência



Fonte: elaborado pelo autor

Figura 07: Exemplificação de diagrama de sequência



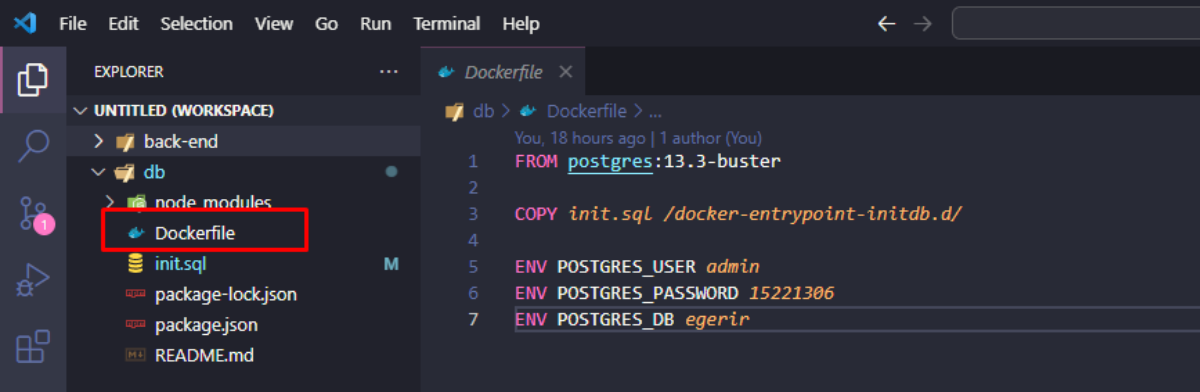
Fonte: elaborado pelo autor

As figuras 6 e 7 fornecem exemplos práticos do emprego do diagrama de sequência no contexto do software em fase de desenvolvimento. Essas representações gráficas são ferramentas valiosas para ilustrar a interação entre os diferentes elementos do sistema, destacando a ordem e o fluxo das operações.

7.7 BACKEND

A construção do backend do software foi iniciada com a criação de um container da imagem do postgres no docker através da criação de um dockerfile.

Figura 08: Exemplificação de criação da imagem do banco de dados postgres



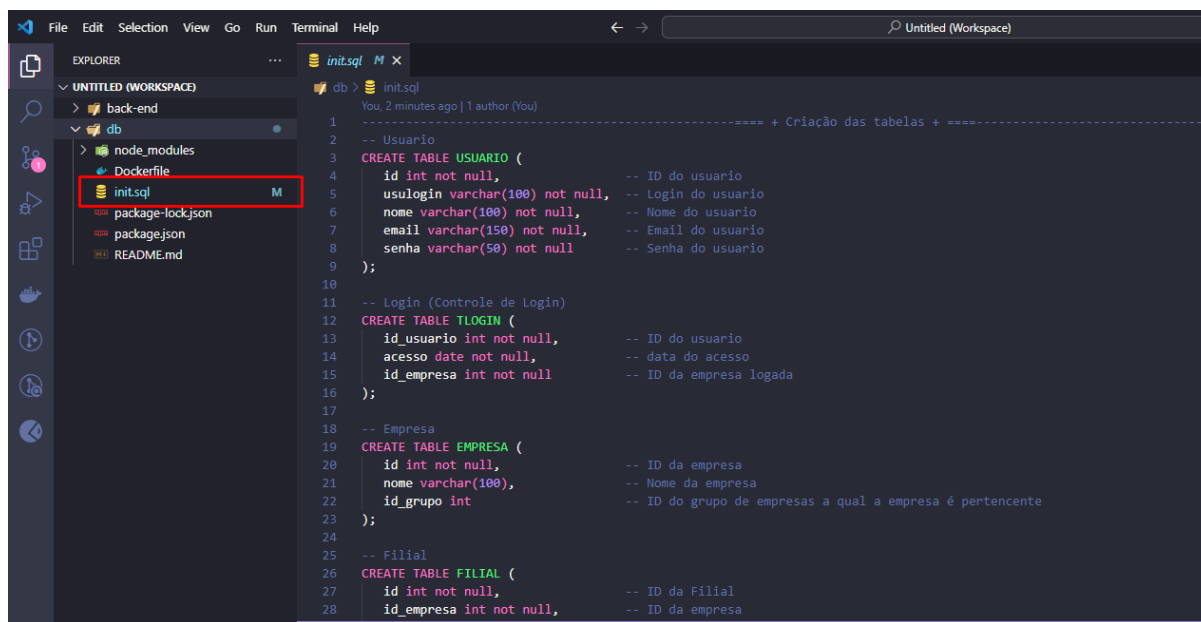
```
File Edit Selection View Go Run Terminal Help
EXPLORER
UNTITLED (WORKSPACE)
  back-end
  db
    node_modules
      Dockerfile
    init.sql
    package-lock.json
    package.json
    README.md
Dockerfile
db > Dockerfile > ...
You, 18 hours ago | 1 author (You)
1 FROM postgres:13.3-buster
2
3 COPY init.sql /docker-entrypoint-initdb.d/
4
5 ENV POSTGRES_USER admin
6 ENV POSTGRES_PASSWORD 15221306
7 ENV POSTGRES_DB egerir
```

Fonte: elaborado pelo autor

A figura acima ilustra a criação da imagem do docker com as informações de acesso ao banco de dados.

A definição das estruturas das tabelas foi adicionada ao arquivo init.sql (também referenciada na criação da imagem do postgres) para que o banco já “suba” com a estrutura de tabelas definidas.

Figura 09: Exemplificação de criação das tabelas



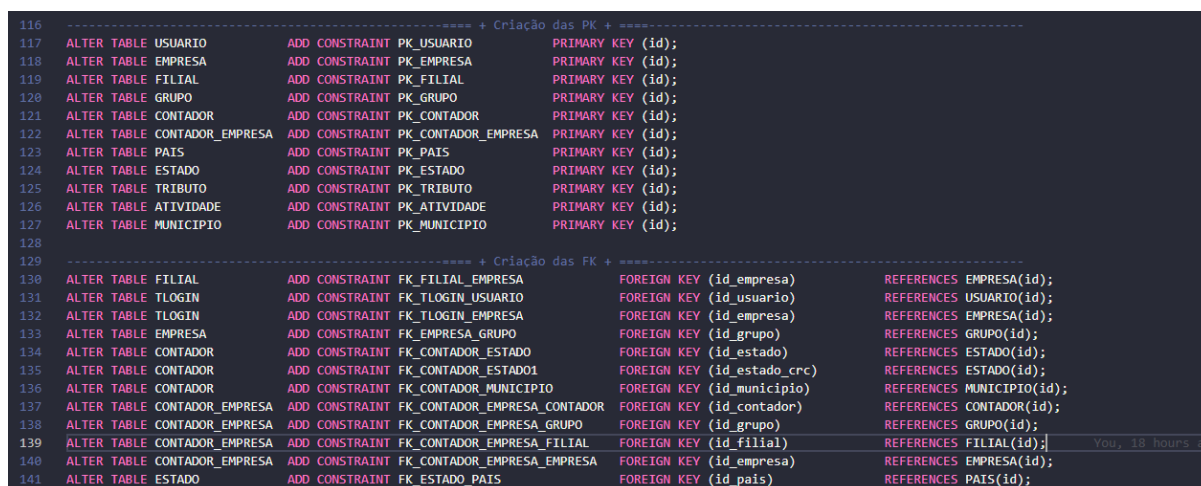
```

1 ----- + Criação das tabelas + -----
2 -- Usuario
3 CREATE TABLE USUARIO (
4     id int not null, -- ID do usuario
5     usulogin varchar(100) not null, -- Login do usuario
6     nome varchar(100) not null, -- Nome do usuario
7     email varchar(150) not null, -- Email do usuario
8     senha varchar(50) not null -- Senha do usuario
9 );
10
11 -- Login (Controle de Login)
12 CREATE TABLE TLOGIN (
13     id_usuario int not null, -- ID do usuario
14     acesso date not null, -- data do acesso
15     id_empresa int not null -- ID da empresa logada
16 );
17
18 -- Empresa
19 CREATE TABLE EMPRESA (
20     id int not null, -- ID da empresa
21     nome varchar(100), -- Nome da empresa
22     id_grupo int -- ID do grupo de empresas a qual a empresa é pertencente
23 );
24
25 -- Filial
26 CREATE TABLE FILIAL (
27     id int not null, -- ID da Filial
28     id_empresa int not null, -- ID da empresa

```

Fonte: elaborado pelo autor

Figura 10: Exemplificação de criação das chaves primárias e estrangeiras



```

116 ----- + Criação das PK + -----
117 ALTER TABLE USUARIO ADD CONSTRAINT PK_USUARIO PRIMARY KEY (id);
118 ALTER TABLE EMPRESA ADD CONSTRAINT PK_EMPRESA PRIMARY KEY (id);
119 ALTER TABLE FILIAL ADD CONSTRAINT PK_FILIAL PRIMARY KEY (id);
120 ALTER TABLE GRUPO ADD CONSTRAINT PK_GRUPO PRIMARY KEY (id);
121 ALTER TABLE CONTADOR ADD CONSTRAINT PK_CONTADOR PRIMARY KEY (id);
122 ALTER TABLE CONTADOR_EMPRESA ADD CONSTRAINT PK_CONTADOR_EMPRESA PRIMARY KEY (id);
123 ALTER TABLE PAIS ADD CONSTRAINT PK_PAIS PRIMARY KEY (id);
124 ALTER TABLE ESTADO ADD CONSTRAINT PK_ESTADO PRIMARY KEY (id);
125 ALTER TABLE TRIBUTO ADD CONSTRAINT PK_TRIBUTO PRIMARY KEY (id);
126 ALTER TABLE ATIVIDADE ADD CONSTRAINT PK_ATIVIDADE PRIMARY KEY (id);
127 ALTER TABLE MUNICIPIO ADD CONSTRAINT PK_MUNICIPIO PRIMARY KEY (id);
128
129 ----- + Criação das FK + -----
130 ALTER TABLE FILIAL ADD CONSTRAINT FK_FILIAL_EMPRESA FOREIGN KEY (id_empresa) REFERENCES EMPRESA(id);
131 ALTER TABLE TLOGIN ADD CONSTRAINT FK_TLOGIN_USUARIO FOREIGN KEY (id_usuario) REFERENCES USUARIO(id);
132 ALTER TABLE TLOGIN ADD CONSTRAINT FK_TLOGIN_EMPRESA FOREIGN KEY (id_empresa) REFERENCES EMPRESA(id);
133 ALTER TABLE EMPRESA ADD CONSTRAINT FK_EMPRESA_GRUPO FOREIGN KEY (id_grupo) REFERENCES GRUPO(id);
134 ALTER TABLE CONTADOR ADD CONSTRAINT FK_CONTADOR_ESTADO FOREIGN KEY (id_estado) REFERENCES ESTADO(id);
135 ALTER TABLE CONTADOR ADD CONSTRAINT FK_CONTADOR_ESTADO1 FOREIGN KEY (id_estado_crc) REFERENCES ESTADO(id);
136 ALTER TABLE CONTADOR ADD CONSTRAINT FK_CONTADOR_MUNICIPIO FOREIGN KEY (id_municipio) REFERENCES MUNICIPIO(id);
137 ALTER TABLE CONTADOR_EMPRESA ADD CONSTRAINT FK_CONTADOR_EMPRESA_CONTADOR FOREIGN KEY (id_contador) REFERENCES CONTADOR(id);
138 ALTER TABLE CONTADOR_EMPRESA ADD CONSTRAINT FK_CONTADOR_EMPRESA_GRUPO FOREIGN KEY (id_grupo) REFERENCES GRUPO(id);
139 ALTER TABLE CONTADOR_EMPRESA ADD CONSTRAINT FK_CONTADOR_EMPRESA_FILIAL FOREIGN KEY (id_filial) REFERENCES FILIAL(id);
140 ALTER TABLE CONTADOR_EMPRESA ADD CONSTRAINT FK_CONTADOR_EMPRESA_EMPRESA FOREIGN KEY (id_empresa) REFERENCES EMPRESA(id);
141 ALTER TABLE ESTADO ADD CONSTRAINT FK_ESTADO_PAIS FOREIGN KEY (id_pais) REFERENCES PAIS(id);

```

Fonte: elaborado pelo autor

As figuras 9 e 10 apresentadas anteriormente servem como exemplos concretos da elaboração da estrutura de tabelas e das respectivas conexões, conforme detalhado no diagrama de entidade e relacionamento mencionado anteriormente. Essas ilustrações oferecem uma representação visual da implementação prática das relações entre entidades, destacando a forma como as tabelas são organizadas e interligadas para refletir a modelagem proposta no diagrama.

Com as estruturas de banco definidas basta criar a imagem do banco no docker e executá-la, para que o banco esteja “rodando”.

Para isso no terminal executamos os comando listados abaixo:

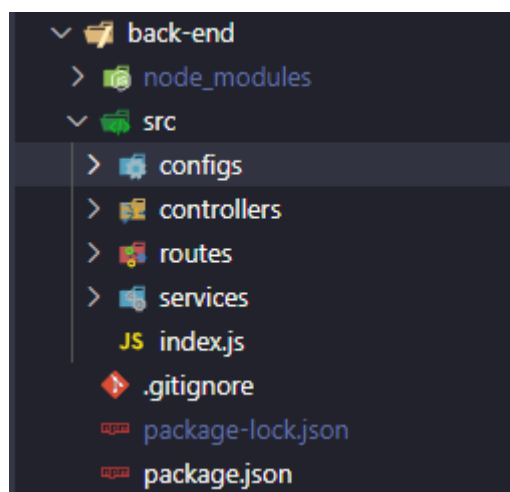
- `docker build -t egerir . ;`
- `docker run -p 5432:5432 -d egerir;`

O “docker build” é responsável por criar a imagem do banco de dados baseado no arquivo Dockerfile definido no repositório “db” do projeto.

O “docker run” irá executar a imagem “egerir” anteriormente criada e transformá-lo em um container, após sua execução o container já estará acessível e uma instância do banco estará criada e rodando, já sendo possível o acesso ao banco de dados.

Com o banco criado e instanciado, passamos a criar os micro-serviços, que serão responsáveis pela comunicação do backend com o frontend, os quais irão efetivar, manter e persistir os dados no banco de dados. Para isso o projeto do “backend” foi dividido em 3 seções, sendo elas o router, controller e services (anteriormente mencionados e explicados no tópico da arquitetura).

Figura 11: Exemplificação da criação da estrutura de diretórios do backend



Fonte: elaborado pelo autor

A imagem 11 fornece uma representação visual da organização de pastas e do fluxo operacional do projeto. Através dessa ilustração, é possível entender a disposição hierárquica das pastas e como as diferentes partes do projeto interagem entre si.

Cada rota possuirá um método em service, controller e router, mas para que o projeto fique mais organizado cada tabela do banco de dados possui um arquivo service que é responsável por manter e persistir os dados no banco de dados.

Figura 12: Exemplificação de criação de arquivo service

```

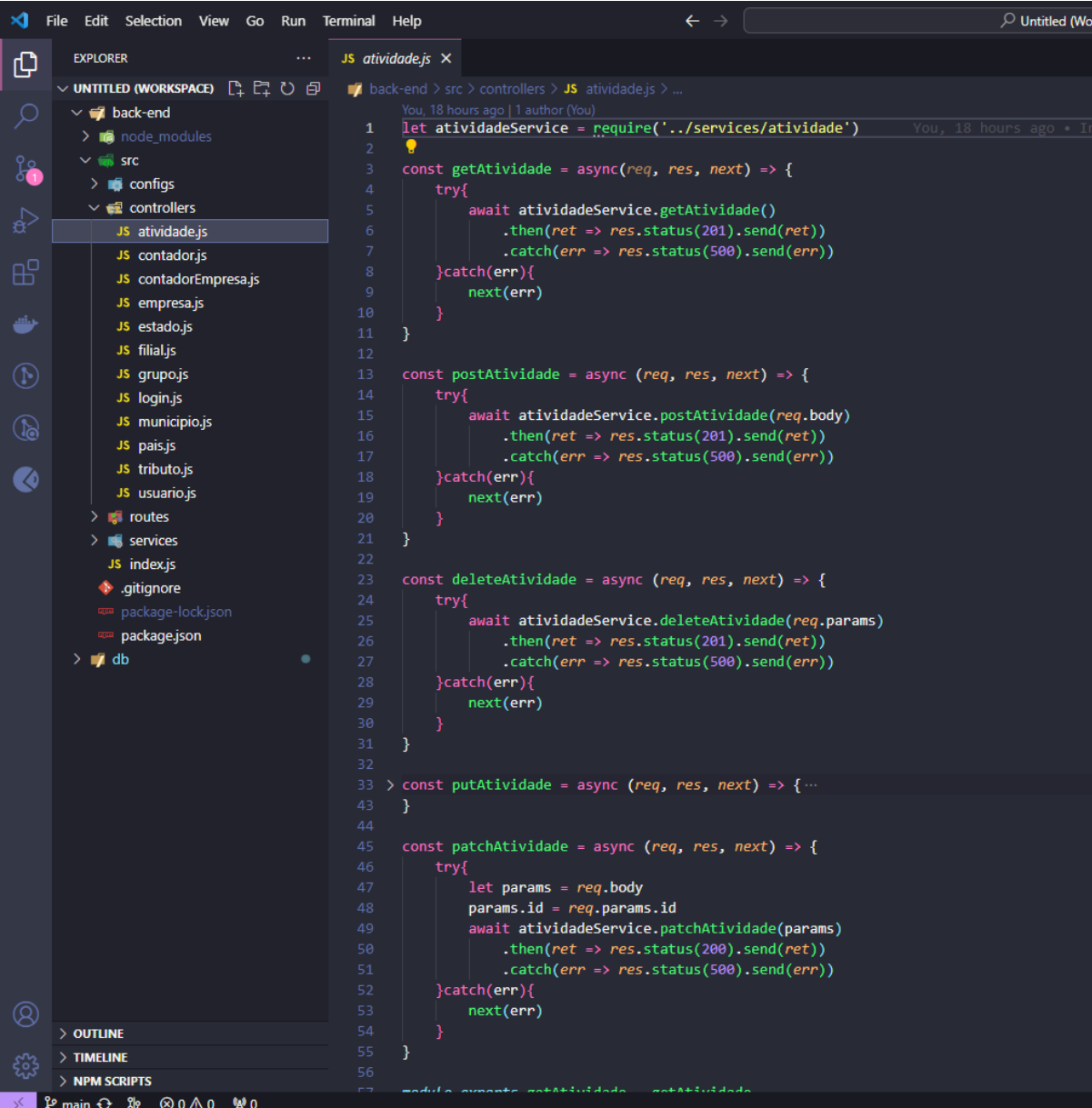
40  const cPatch =
41    `update atividade
42      set `
43
44  const getAtividade = async (params) => {
45    let atividade = {}
46    await db.query(cGet)
47      .then(ret => atividade = {total: ret.rows.length, atividade:ret.rows})
48      .catch(err => atividade = err.stack)
49    return atividade
50  }
51
52  const postAtividade = async (params) => {
53    const { id, titulo, descricao, valor, id_tributo,
54      id_contador_empresa,
55      entrega, previsao, criacao, id_usuario,
56      id_estado, id_municipio } = params
57    await db.query(cPost, [id, titulo, descricao, valor, id_tributo,
58      id_contador_empresa,
59      entrega, previsao, criacao, id_usuario,
60      id_estado, id_municipio])
61  }
62
63  const deleteAtividade = async (params) => {
64    const { id } = params
65    await db.query(cDelete, [id])
66  }
67
68  > const putAtividade = async (params) => { ...
69  }
70
71  const patchAtividade = async (params) => {
72    let fields = ''
73    let binds = []
74    binds.push(params.id)
75    let countParams = 1
76    if (params.titulo){
77      countParams++
78      fields += ` titulo = ${countParams}`
79      binds.push(params.titulo)
80    }else if (params.descricao){
81      countParams++
82      fields += ` descricao = ${countParams}`
83      binds.push(params.descricao)
84    }else if (params.valor){
85      countParams++
86      fields += ` valor = ${countParams}`
87      binds.push(params.valor)
88    }
89  }

```

Fonte: elaborado pelo autor

A imagem 12 ilustra o processo de criação de um arquivo de service para as APIs.

Figura 13: Exemplicação de um arquivo controller

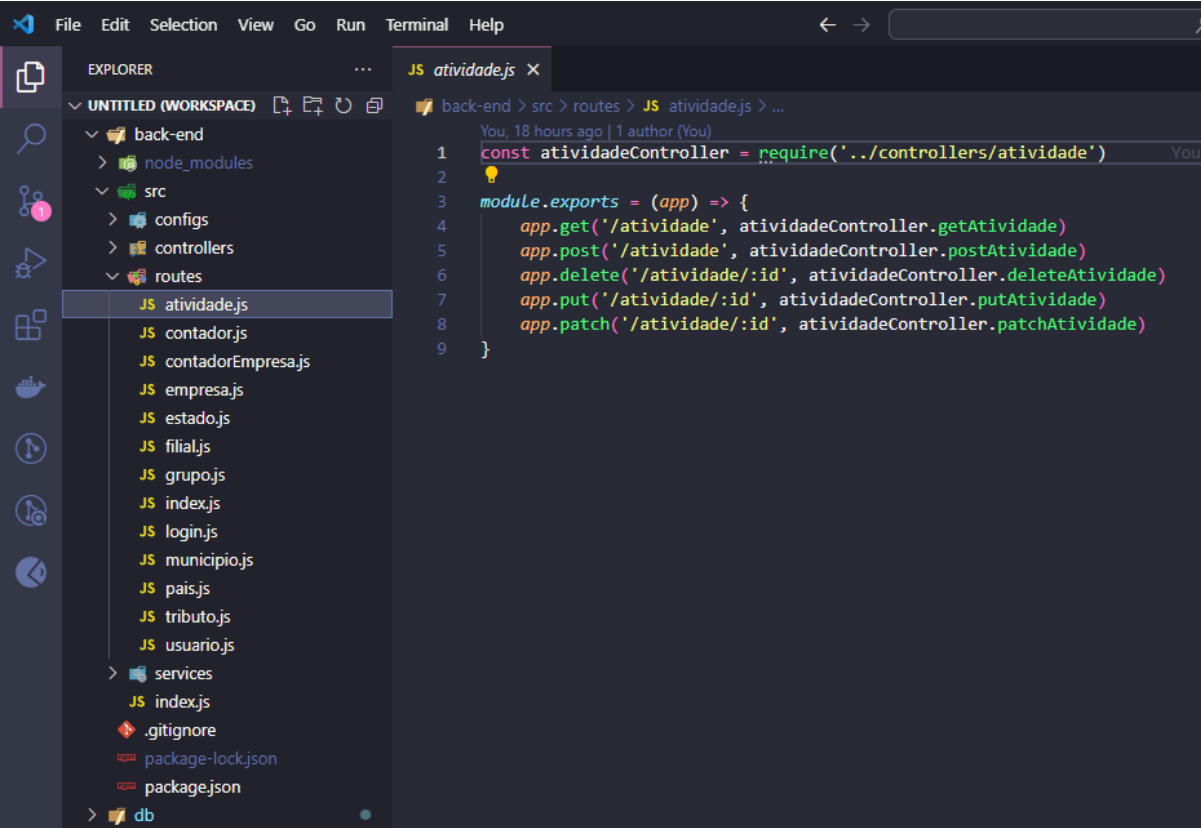


```
1 let atividadeService = require('../services/atividade')
2
3 const getAtividade = async (req, res, next) => {
4   try{
5     await atividadeService.getAtividade()
6     .then(ret => res.status(201).send(ret))
7     .catch(err => res.status(500).send(err))
8   }catch(err){
9     next(err)
10  }
11 }
12
13 const postAtividade = async (req, res, next) => {
14   try{
15     await atividadeService.postAtividade(req.body)
16     .then(ret => res.status(201).send(ret))
17     .catch(err => res.status(500).send(err))
18   }catch(err){
19     next(err)
20   }
21 }
22
23 const deleteAtividade = async (req, res, next) => {
24   try{
25     await atividadeService.deleteAtividade(req.params)
26     .then(ret => res.status(201).send(ret))
27     .catch(err => res.status(500).send(err))
28   }catch(err){
29     next(err)
30   }
31 }
32
33 > const putAtividade = async (req, res, next) => { ...
34 }
35
36 const patchAtividade = async (req, res, next) => {
37   try{
38     let params = req.body
39     params.id = req.params.id
40     await atividadeService.patchAtividade(params)
41     .then(ret => res.status(200).send(ret))
42     .catch(err => res.status(500).send(err))
43   }catch(err){
44     next(err)
45   }
46 }
47
48 >
49
50
51
52
53
54
55
56
57
```

Fonte: elaborado pelo autor

A representação na Figura 13 ilustra o processo de elaboração de um arquivo controller para as APIs. Esse componente desempenha um papel crucial na arquitetura de um sistema, atuando como um intermediário entre as requisições recebidas e a lógica de negócios subjacente. Por meio desse arquivo, é possível gerenciar e direcionar as solicitações vindas das APIs para as funções apropriadas

Figura 14: Exemplificação de arquivo router



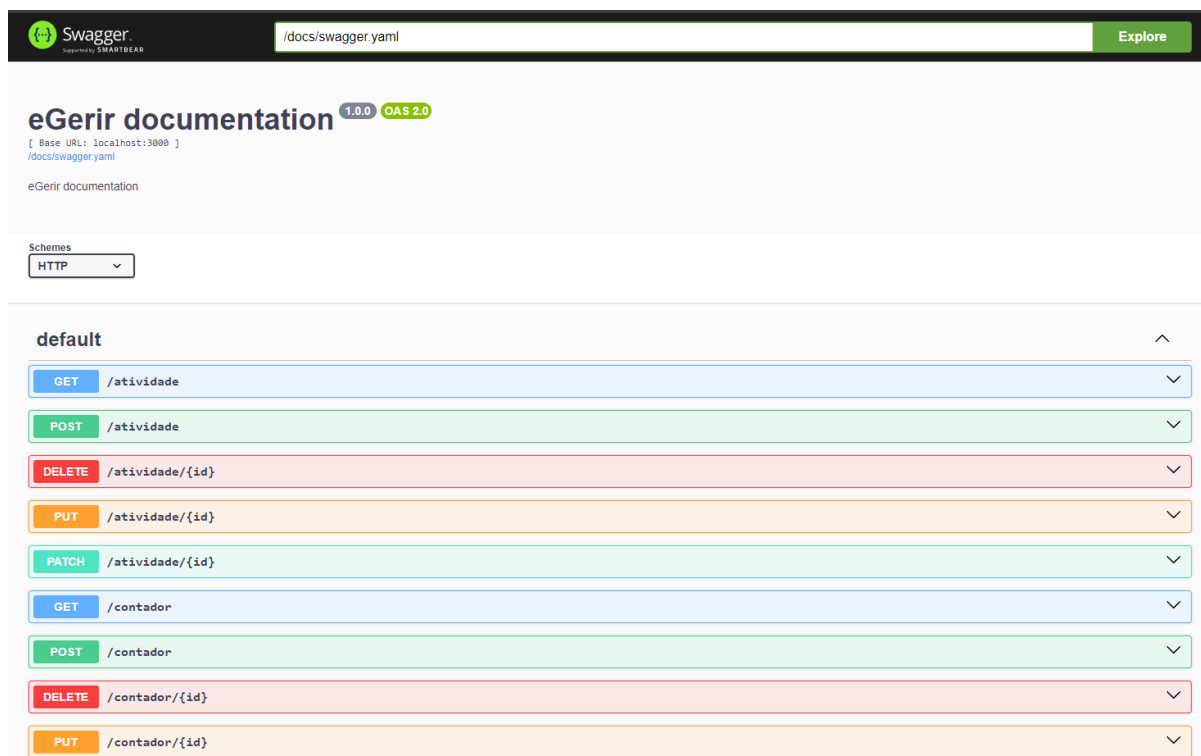
```
1 const atividadeController = require('../controllers/atividade')
2
3 module.exports = (app) => {
4   app.get('/atividade', atividadeController.getAtividade)
5   app.post('/atividade', atividadeController.postAtividade)
6   app.delete('/atividade/:id', atividadeController.deleteAtividade)
7   app.put('/atividade/:id', atividadeController.putAtividade)
8   app.patch('/atividade/:id', atividadeController.patchAtividade)
9 }
```

Fonte: elaborado pelo autor

A figura 14 ilustra o processo de desenvolvimento de um arquivo router para as APIs. Nesse contexto, a criação do arquivo router desempenha um papel crucial na organização e na gestão eficiente das APIs.

Para se manter um software saudável e de fácil manutenção é importante manter a documentação do código, nesse caso foi utilizado o framework “swagger” para criar a documentação das rotas.

Figura 15: Exemplificação da documentação das rotas feitas no swagger



Fonte: elaborado pelo autor

A imagem 15 exibida acima ilustra a documentação das APIs elaboradas por meio do framework Swagger para o atual projeto. Essa documentação fornece uma visão abrangente das APIs desenvolvidas, facilitando a compreensão e implementação por parte dos desenvolvedores. O uso do Swagger neste contexto não apenas simplifica a documentação, mas também promove a interoperabilidade e a comunicação eficiente entre as diferentes partes do sistema, contribuindo para a robustez e manutenção eficaz do projeto.

7.8 FRONTEND

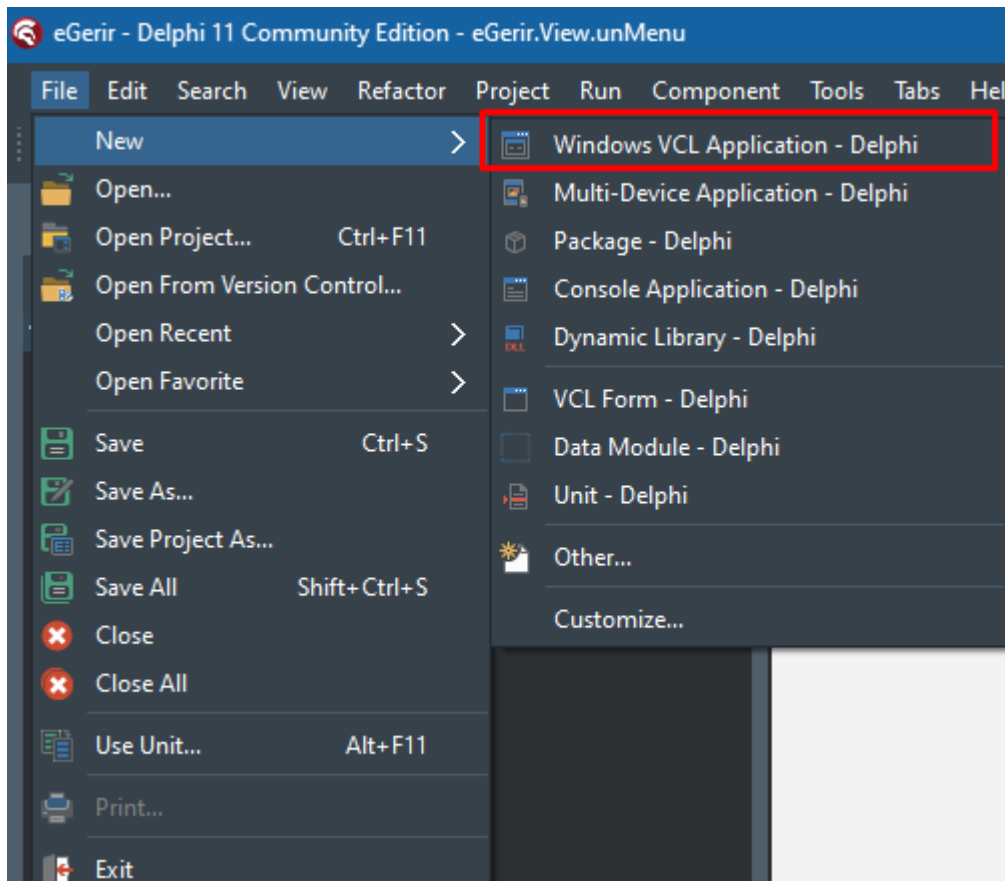
O Frontend desenvolvido em Delphi CE (11) foi feito inteiramente com recursos da ferramenta, não tendo sido utilizado nenhum framework externo em seu desenvolvimento.

7.8.1 VCL

A aplicação Windows VCL foi desenvolvida como o projeto principal, e suas estruturas de arquivos e pastas foram organizadas de acordo com alguns padrões organizacionais da arquitetura MVC.

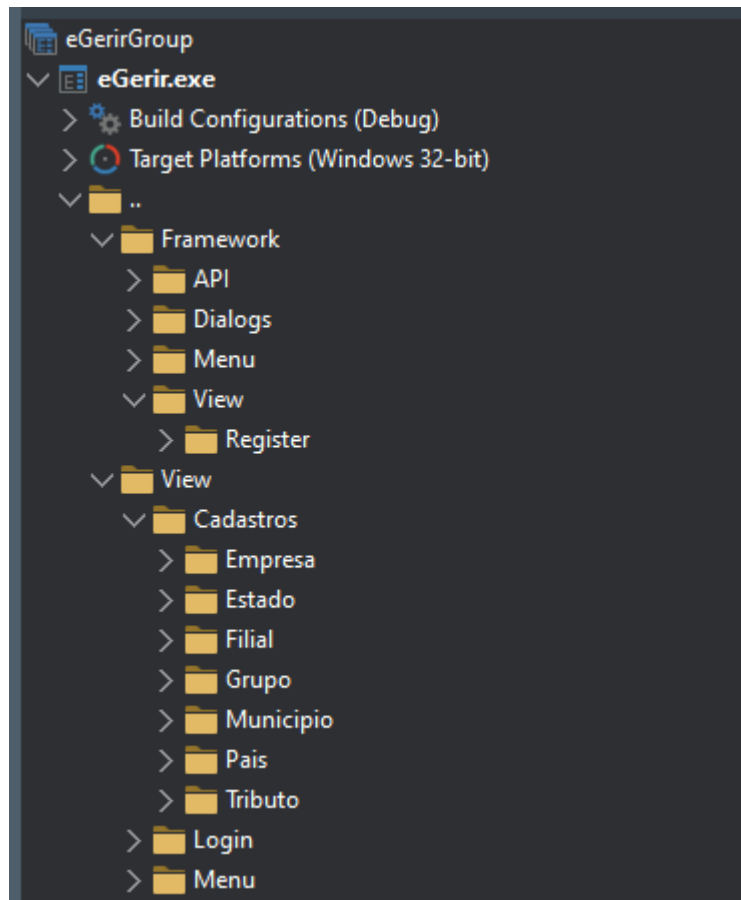
Essa abordagem proporciona uma separação clara das responsabilidades, com modelos, visualizações e controladores devidamente distribuídos. Tal organização facilita a manutenção, escalabilidade e compreensão do código-fonte, promovendo boas práticas de desenvolvimento.

Figura 16: Exemplificação de criação de um projeto VCL



Fonte: elaborado pelo autor

A Figura acima ilustra a criação de um projeto VCL Delphi.

Figura 17: Exemplificação da estrutura dos diretórios no frontend

Fonte: elaborado pelo autor

A Figura 17 representa de forma organizada a estrutura de arquivos e diretórios do projeto, proporcionando uma visão clara das relações entre os componentes. Essa representação visual é crucial para a compreensão rápida da hierarquia do sistema.

Visando a entrega de um software desacoplado e escalável foram desenvolvidos alguns recursos próprios denominados de “Framework”, a qual se trata de uma framework própria do projeto. Além disso, a estrutura da framework foi projetada levando em consideração os princípios de design orientado a objetos, favorecendo a criação de componentes independentes que podem ser interconectados de maneira eficiente.

Outro ponto relevante é a implementação de padrões de design consagrados, como o padrão de Injeção de Dependência, que promove a separação de preocupações e facilita a substituição de componentes sem impactar negativamente as demais partes do sistema. A utilização desses padrões contribui para a manutenção da coesão e baixo acoplamento entre os módulos, tornando o software mais resistente a mudanças e mais fácil de evoluir ao longo do tempo.

Framework.Menu.Forms: É uma classe crucial no sistema, desempenhando um papel fundamental no tratamento e na gestão de formulários externos. Sua principal responsabilidade reside na chamada desses formulários no menu principal, e ela foi projetada com o objetivo de eliminar o acoplamento indesejado entre formulários externos e o menu principal do projeto.

Figura 18: Estrutura da classe TFrwMenuForms

```
unit Framework.Menu.Forms;
interface
uses
  System.TypeInfo, Vcl.ExtCtrls;
type
  IMenu = interface
    ['{4CD1620C-F5F2-40E5-8907-2EAA37EE8488}']
    procedure OpenForm;
    procedure CloseFMXForms;
  end;
  TFrwMenuForms = class(TInterfacedObject, IMenu)
  strict private
    FMenu: string;
    FParent: TPanel;
  private
    constructor Create(AMenu: string; AParent: TPanel); reintroduce;
    procedure OpenForm;
    procedure CloseFMXForms;
  public
    class function New(AMenu: string; AParent: TPanel): IMenu;
  end;
```

Fonte: elaborado pelo autor

A estrutura da classe é ilustrada de maneira abrangente na Figura 18, proporcionando uma visão clara e concisa de como ela está organizada internamente.

Framework.API.Controls: Essa classe tem como responsabilidade a comunicação do frontend com o backend passando todas as requisições dos apis por essa classe.

Figura 19: Estrutura da classe TFrwAPIControl

```

unit Framework.API.Control;

interface

uses
  REST.Client, REST.Json;

type
  IAPIControl = interface
    ['{FD8FA5FD-BBAF-44B7-97DA-D021D31E6B0E}']
    function AddMenu(AMenu: string): IAPIControl;
    function AddID(AID: string): IAPIControl;
    function AddJson(AJson: string): IAPIControl;
    function PostAction: IAPIControl;
    function PatchAction: IAPIControl;
    function DeleteAction: IAPIControl;
    function NewMaxSequence: Integer;
  end;

  TFrwAPIControl = class(TInterfacedObject, IAPIControl)
  strict private
    FMenu: string;
    FJson: string;
    FID: string;
    Client: TRESTClient;
    Request: TRESTRequest;
    Response: TRESTResponse;
  private
    constructor Create; reintroduce;
    function AddMenu(AMenu: string): IAPIControl;
    function AddID(AID: string): IAPIControl;
    function AddJson(AJson: string): IAPIControl;
    function PostAction: IAPIControl;
    function PatchAction: IAPIControl;
    function DeleteAction: IAPIControl;
    function NewMaxSequence: Integer;
  public
    class function New: IAPIControl;
  end;

```

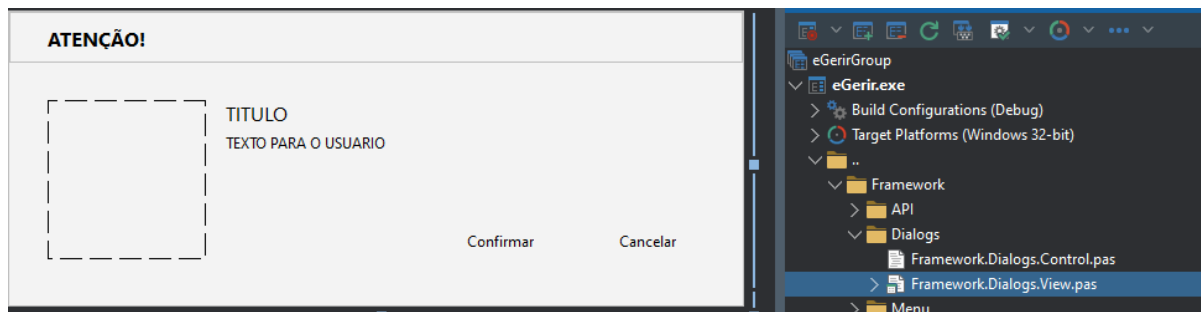
Fonte: elaborado pelo autor

A Figura 19 fornece uma visão estrutural da classe, destacando sua importância e relação com outros componentes do framework. Essa abordagem promove coesão, manutenibilidade e facilita a implementação de novos recursos, contribuindo para um design modular e eficiente.

A classe Framework.Dialogs.Control centraliza o uso de mensagens do sistema, unificando chamadas para um único formulário modal padrão, representado pela classe Framework.Dialogs.View. Essa abordagem simplifica a interação com o usuário, proporcionando uma experiência coesa e consistente. A classe controla e exibe modais, promovendo uma separação clara entre a lógica de controle e a apresentação, facilitando a

manutenção e permitindo futuras extensões do sistema de mensagens. Em resumo, contribui para um design modular e escalável, melhorando a usabilidade do aplicativo.

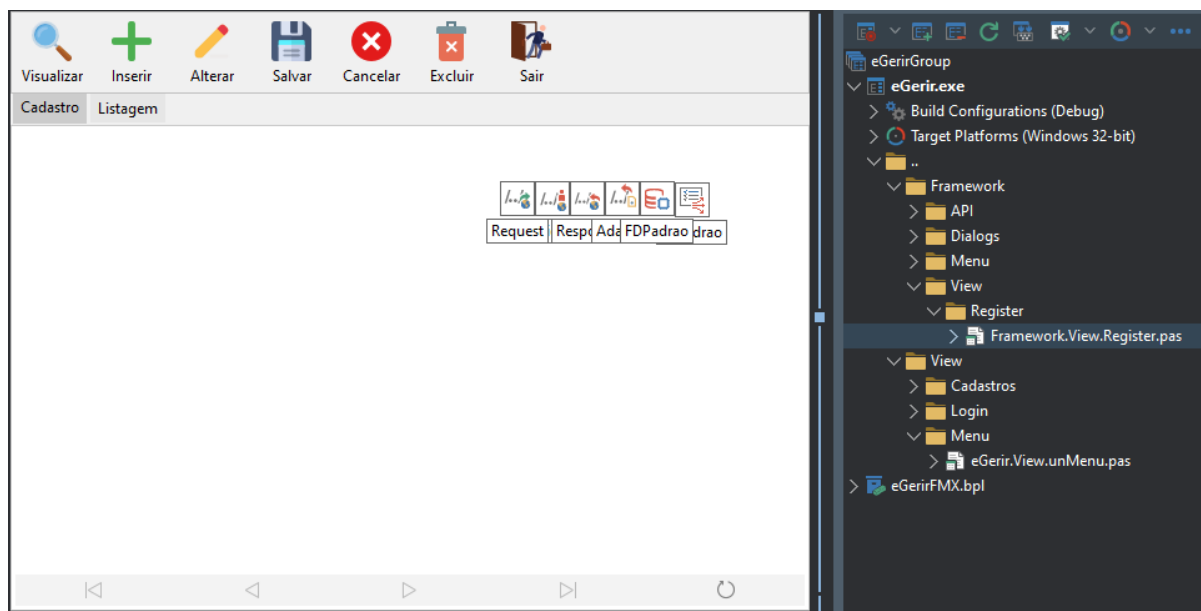
Figura 20: Ilustra o formulário padrão de mensagens do sistema



Fonte: elaborado pelo autor

A Figura 20 representa visualmente o modal padrão de mensagens mencionado. Essa ilustração oferece uma visão gráfica do formulário modal associado à classe `Framework.Dialogs.View`. Essa representação visual inclui elementos como caixas de diálogo, botões, e imagens relevantes para a exibição e interação com as mensagens do sistema. A presença de uma figura para ilustrar o modal destaca a importância do design visual na experiência do usuário.

`Framework.View.Register`: é utilizado como o formulário padrão para telas de cadastro. Esse formulário estabelece um conjunto de características comuns que serão herdadas por todas as telas de cadastro, proporcionando uma abordagem unificada para o tratamento de dados.

Figura 21: Ilustração da herança da tela de cadastro

Fonte: elaborado pelo autor

A Figura 21, conforme mencionado, ilustra visualmente como essa herança funciona, destacando a eficiência e agilidade que ela traz para o processo de desenvolvimento de cadastros. Ao unificar os tratamentos em um formulário padrão, alcança-se uma coerência na aplicação, facilitando a compreensão e manutenção do código. Esse enfoque também pode contribuir para a consistência na interface do usuário e para a redução de possíveis erros, uma vez que as funcionalidades comuns estão centralizadas em um único local.

Empregando o princípio da herança, e adotando métodos centralizados para as telas de cadastro a serem desenvolvidas, é possível simplificar significativamente a implementação, exigindo apenas um método adicional, além dos componentes específicos que serão utilizados. A Figura 22 abaixo exemplifica esse conceito, destacando a eficiência e a organização proporcionadas por essa abordagem na criação de interfaces de cadastro.

Figura 22: Implementação da tela de cadastro de empresa

```

· unit eGerir.View.Cadastro.unEmpresa;
·
· interface
·
· uses
·   Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
·   Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Framework.View.Register, Data.DB,
·   REST.Types, FireDAC.Stan.Intf, FireDAC.Stan.Option, FireDAC.Stan.Param,
·   FireDAC.Stan.Error, FireDAC.DatS, FireDAC.Phys.Intf, FireDAC.DApt.Intf,
10  FireDAC.Comp.DataSet, FireDAC.Comp.Client, REST.Response.Adapter, REST.Client,
·   Data.Bind.Components, Data.Bind.ObjectScope, Vcl.Grids, Vcl.DBGrids,
·   Vcl.DBCtrls, Vcl.ComCtrls, Vcl.Buttons, Vcl.ExtCtrls, Vcl.StdCtrls, Vcl.Mask;
·
· type
·   TfrmEmpresa = class(TFrwViewRegister)
·     edtID: TDBLabeledEdit;
·     edtNome: TDBLabeledEdit;
·     DBLabeledEdit3: TDBLabeledEdit;
·   private
20   { Private declarations }
·   public
·     { Public declarations }
·   protected
·     procedure InitializeForm; override;
·   end;
·
·   var
·     frmEmpresa: TfrmEmpresa;
·
30  implementation
31  {$R *.dfm}
·   { TfrmEmpresa }
·
·   procedure TfrmEmpresa.InitializeForm;
·   begin
·     FMenu := 'empresa';
·     inherited;
·   end;
40
· end.

```

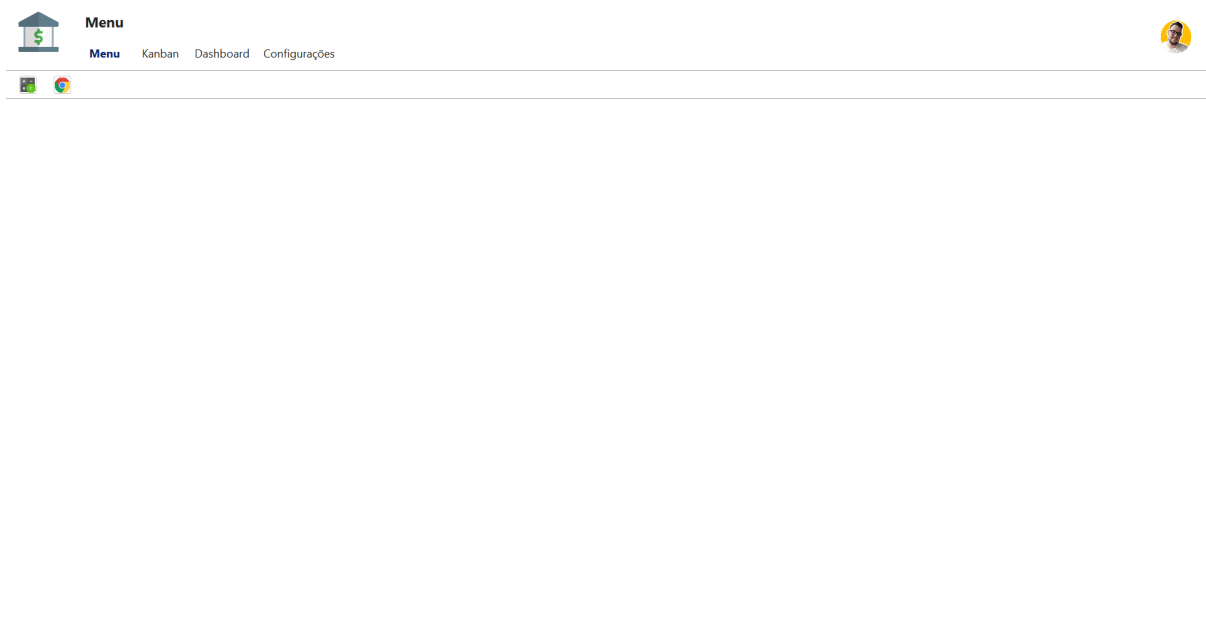
Fonte: elaborado pelo autor

As telas-chave do sistema, como a de login, menu principal e configurações/cadastros, foram desenvolvidas utilizando a VCL (Visual Component Library). Essa escolha proporcionou benefícios como eficiência no design de interfaces gráficas, resultando em telas intuitivas, consistentes e visualmente atrativas. O uso da VCL contribuiu para uma experiência de usuário mais fluida, facilitando a interação e otimizando a usabilidade do sistema como um todo.

Figura 23: Tela de login do sistema

Fonte: elaborado pelo autor

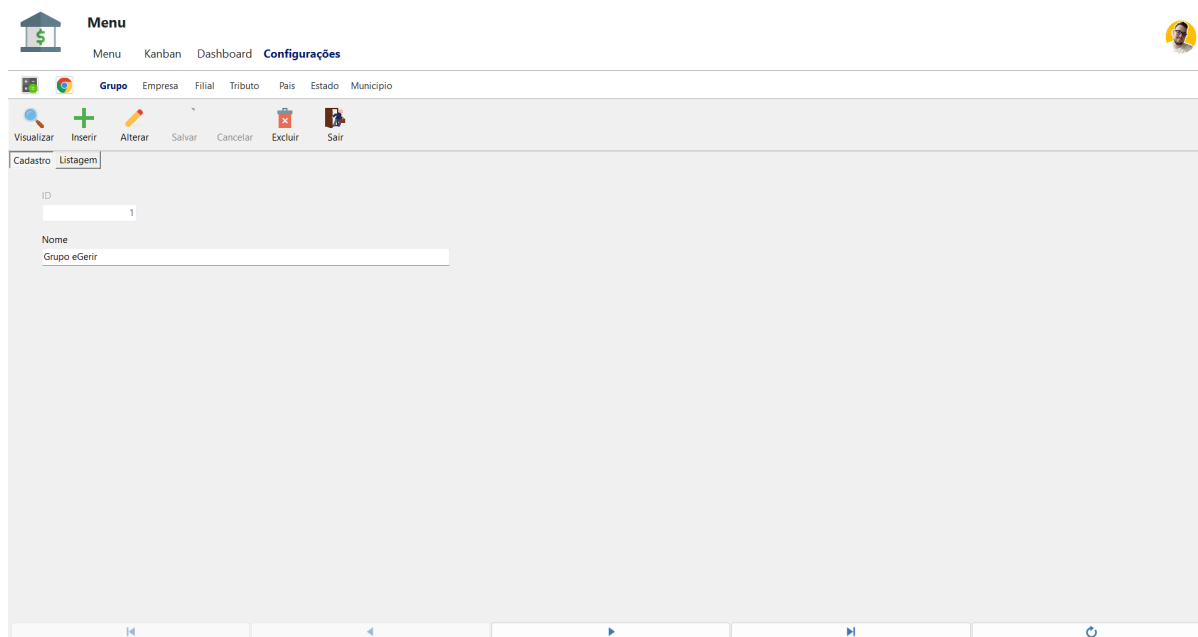
A Figura 23 representa a tela de login do sistema desenvolvido, sendo a entrada principal para os usuários. Projetada com uma interface intuitiva, destaca-se pela eficiência e amigabilidade. Essa interface não é apenas um ponto de entrada, mas também um componente crucial na experiência do usuário, estabelecendo o tom para a interação subsequente com o sistema. Sua concepção reflete preocupações com usabilidade, acessibilidade e segurança, contribuindo para uma experiência positiva e eficiente.

Figura 24: Menu principal do sistema

Fonte: elaborado pelo autor

Na figura 24, é possível visualizar a interface da tela de menu do sistema, proporcionando aos usuários uma visão clara e organizada das opções disponíveis. Essa representação gráfica destaca-se pela sua usabilidade, permitindo que os usuários naveguem intuitivamente pelas diversas ações oferecidas. A disposição ordenada dos itens no menu facilita a identificação e seleção das funcionalidades desejadas, contribuindo para uma experiência do usuário mais eficiente e agradável. O design da tela reflete o cuidado em fornecer uma interface amigável, otimizando a interação entre o usuário e o sistema, e promovendo uma utilização eficaz das capacidades oferecidas.

Figura 25: Tela de cadastro



Fonte: elaborado pelo autor

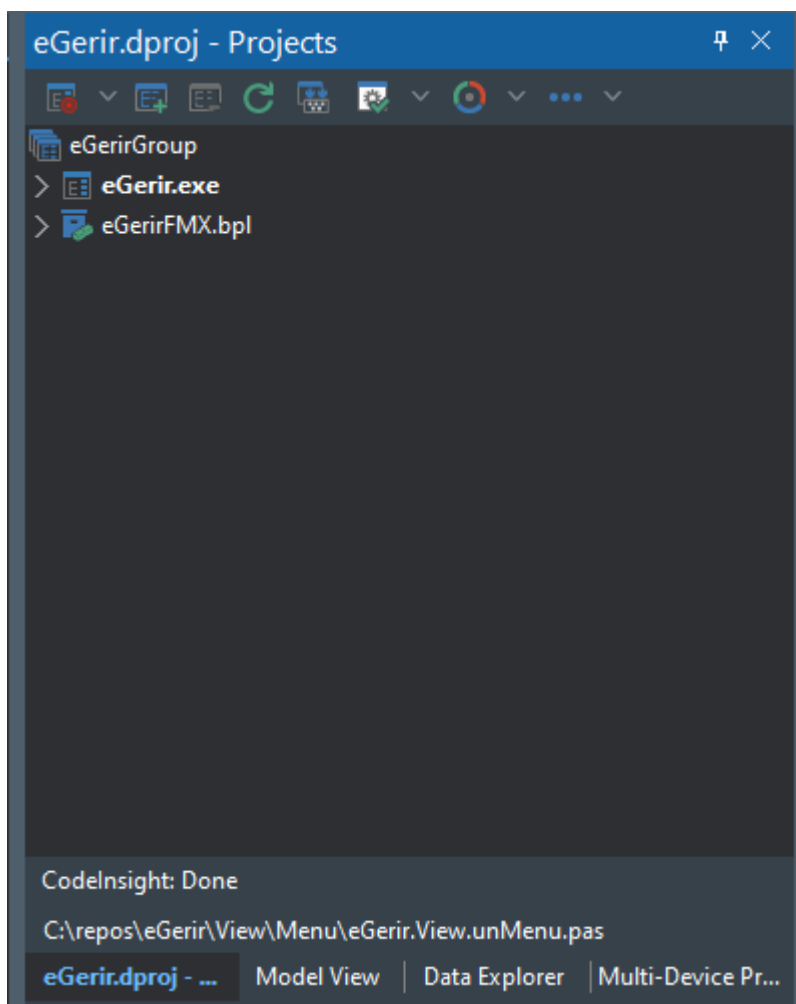
Na figura 25, podemos observar uma representação visual elucidativa da interface de cadastro no sistema. É evidente que a uniformidade é mantida, uma vez que as telas de cadastro aderem consistentemente ao padrão estabelecido pela framework desenvolvida. As variações notáveis residem nos campos destinados ao preenchimento, os quais se ajustam conforme a natureza específica de cada cadastro. Essa abordagem coesa não apenas confere uma estética visualmente harmoniosa, mas também facilita a navegação do usuário ao garantir uma experiência intuitiva e familiar, independentemente do contexto de cadastro em questão. Essa consistência na estrutura de cadastro não apenas otimiza a usabilidade, mas também contribui para a eficiência operacional, tornando o sistema mais acessível e fácil de aprender para os usuários.

7.8.2 FMX

A FMX (FireMonkey) surgiu como um pacote adicional (BPL) integrado ao nosso projeto, ampliando as capacidades visuais e de componentes do Delphi. Essa framework foi estrategicamente incorporada para fornecer recursos não disponíveis na VCL. Para integrar harmoniosamente essas duas frameworks, desenvolvemos um pacote que encapsula esses

recursos específicos e um grupo de projeto que engloba ambas as aplicações. Essa abordagem permitiu uma sinergia eficiente entre as tecnologias, agregando valor ao nosso desenvolvimento.

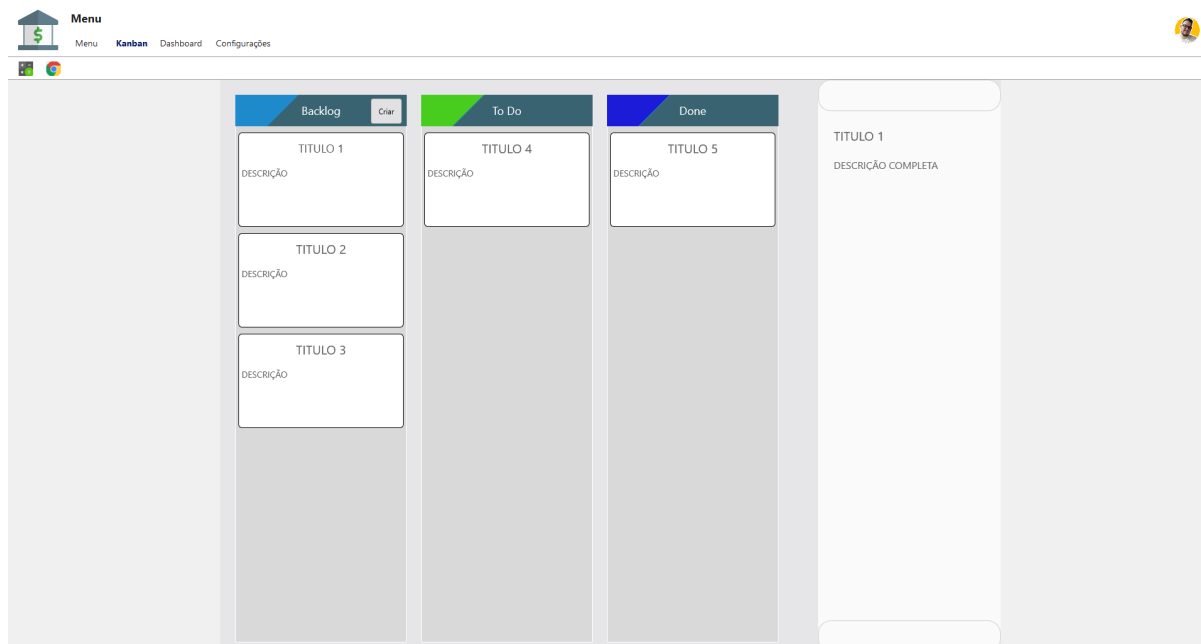
Figura 26: Projeto VCL e FMX adicionado ao grupo



Fonte: elaborado pelo autor

Neste conjunto FMX, foi incorporada uma interface de Kanban dedicada à administração de entregas e obrigações acessórias. Essa adição visa proporcionar uma abordagem mais eficiente e visual para o acompanhamento e organização de tarefas, promovendo maior agilidade no gerenciamento de projetos. A tela de Kanban oferece uma visão intuitiva do progresso, permitindo aos usuários monitorar de forma simplificada o status de cada entrega e obrigação acessória. Isso não apenas facilita a identificação de áreas que requerem atenção imediata, mas também otimiza o fluxo de trabalho, contribuindo para uma gestão mais eficaz e centrada nas metas estabelecidas.

Figura 27: Imagem da tela de Kanban do projeto



Fonte: elaborado pelo autor

A Figura 27 apresenta a aplicação do método Kanban em um contexto específico, dividindo o quadro em três colunas: "*Backlog*", "*To Do*" e "*Done*". No cenário de uma equipe de contabilidade, a coluna "*Backlog*" lista as tarefas e obrigações tributárias pendentes, enquanto a coluna "*To Do*" representa o trabalho em andamento, com atribuição clara de responsabilidades. A coluna "*Feito*" destaca as tarefas concluídas. Essa abordagem visual facilita a gestão, priorização e cumprimento de prazos, sendo crucial para o ambiente contábil, onde há demanda por transparência, colaboração e conformidade. A Figura 27 também destaca a importância do Kanban na identificação de bloqueios, na promoção da melhoria contínua e na criação de um processo mais eficiente e adaptável.

9 CONSIDERAÇÕES FINAIS

Em conclusão, esta monografia representou uma significativa contribuição para a compreensão e otimização da gestão de entregas contábeis e fiscais, bem como para a aquisição de um profundo conhecimento acerca da complexidade da carga tributária brasileira. O extenso estudo realizado abordou não apenas os aspectos tributários, mas também englobou conceitos fundamentais de engenharia de software, banco de dados e UML, fundamentais para embasar a parte teórica do projeto.

A escolha do ambiente de programação Delphi e do banco de dados Postgre, aliados ao uso de Javascript, permitiu não apenas o desenvolvimento do protótipo, mas também proporcionou uma imersão prática nos desafios enfrentados nesse contexto. A estruturação do projeto, iniciando pelo backend com o desenvolvimento de APIs, seguido pelo frontend com a criação de um framework próprio, demonstrou eficiência ao possibilitar a elaboração de telas de cadastro em tempo reduzido, destacando a agilidade no processo de desenvolvimento.

Os resultados obtidos não apenas atenderam ao objetivo geral estabelecido, mas também trouxeram benefícios tangíveis para a compreensão aprofundada do sistema de informação e do cenário tributário brasileiro. A aprendizagem adquirida durante a pesquisa bibliográfica e o desenvolvimento do protótipo consolidaram conceitos fundamentais, contribuindo não apenas para a conclusão desta monografia, mas também para a formação acadêmica e profissional do autor. Este estudo reforça a importância do contínuo aprimoramento na interseção entre tecnologia da informação e legislação tributária, evidenciando a relevância prática e teórica da monografia desenvolvida.

9.1 TRABALHOS FUTUROS

Com os resultados obtidos, são nítidas as oportunidades de melhorias intrínsecas nas premissas do projeto sendo algumas delas as seguintes:

- Explorar a viabilidade e implementar a funcionalidade de realizar entregas diretamente pelo sistema, proporcionando uma experiência mais completa e conveniente para os usuários;
- Desenvolver e incorporar novos indicadores de desempenho que possam fornecer insights mais abrangentes sobre o funcionamento do sistema, ajudando na tomada de decisões estratégicas;
- Identificar e automatizar tarefas rotineiras, aumentando a eficiência operacional. Isso pode incluir a automação de processos de aprovação, geração de relatórios automáticos e otimização do fluxo de trabalho;
- Desenvolver um sistema de gestão de arquivos integrado para facilitar o armazenamento, organização e recuperação eficiente de documentos importantes, como recibos, guias e outros registros relevantes;

REFERÊNCIAS BIBLIOGRÁFICAS

ARAUJO, Fernanda. **O que é o imposto de renda e para que serve?**

<<https://www.serasa.com.br/blog/o-que-e-imposto-de-renda/>> Acesso em: 09/10/2023

BARBOSA, Luiz Roberto Peroba; TORNOVSKY, Luciana M. Cossermelli. **Reforma tributária: impactos sobre as empresas**. Revista de Direito Tributário Internacional, v. 18, n. 76, p. 107-129, 2021.

BOEG, J. **Kanban em 10 Passos**. InfoQ Brasil, 2011. Disponível em: <<https://www.infoq.com/br/minibooks/priming-kanban-jesper-boeg/>> Acesso em: 03 dez. 2023.

BRASIL. Ministério da Fazenda. Estudos Tributários. **Carga Tributária no Brasil – 2017 (Análise por Tributo e Bases de Incidência)**. Brasília. Novembro. 2018.

BRASIL. Câmara dos Deputados. Proposta de Emenda Constitucional nº 45, de 03 de abril de 2019. Altera o Sistema Tributário Nacional e dá outras providências. Brasília: Câmara dos Deputados, 2019. Disponível em: <https://www.camara.leg.br/proposicoesWeb/prop_mostrarintegra?codteor=1728369&filenome=PEC%2045/2019>. Acesso em: 06 novembro. 2023

BEZERRA, J. M. L.; OLIVEIRA, J. M. C.; ARAÚJO, P. B. **Compliance Tributário e a Gestão das Obrigações Acessórias no Brasil**. Revista Brasileira de Administração, v. 1, n. 1, p. 78-91, 2021.

CAMPANELLE, DAVID MATOS. **SISTEMA TRIBUTÁRIO BRASILEIRO: UMA ANÁLISE CRÍTICA**. Artigo, 2012. 49 pág.

CARNEIRO, Claudio. **Curso de Direito Tributário e Financeiro**. 9. ed. São Paulo: Saraiva Educação, 2020. 824 p. Disponível em: <https://bdjur.stj.jus.br/jspui/bitstream/2011/84530/curso_direito_tributario_carneiro_9.ed.pdf>. Acesso em: 06 novembro. 2023.

DATE, C. J.. **INTRODUÇÃO A SISTEMAS DE BANCOS DE DADOS**. 8. ed. Rio de Janeiro: Elsevier, 2003.

DE MORAIS, FAUSTO SANTOS. **A TRIBUTAÇÃO NA ERA DIGITAL E OS DESAFIOS DO SISTEMA TRIBUTÁRIO NO BRASIL** v. 19, n. 1. Revista, 2023. 23 pág.

Diagramas de Caso de Uso. Disponível em: <<https://www.ibm.com/docs/pt-br/rsm/7.5.0?topic=diagrams-use-case>>. Acesso em: 11 jul. 2023.

Diagramas de Classes. Disponível em: <<https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=structure-class-diagrams>>. Acesso em: 7 nov. 2023.

Diagramas de Seqüência. Disponível em: <<https://www.ibm.com/docs/pt-br/rsm/7.5.0?topic=uml-sequence-diagrams>>. Acesso em: 7 nov. 2023.

ELMASRI, Ramez; NAVATHE, Shamkant B.. **SISTEMAS DE BANCO DE DADOS**. 6. ed. São Paulo: Addison Wesley, 2011.

EMBARCADERO. **Delphi - Products - Embarcadero WebSite**. Disponível em: <https://www.embarcadero.com/br/products/delphi>. Acesso em 01 nov. 2023.

FAVARIN, Armando; OLIVEIRA, Edilene Santana de. **O cenário tributário atual no Brasil e os desafios para as empresas**. Revista Eletrônica de Contabilidade, v. 13, n. 3, p. 153-170, 2021.

FUCK, Luciano Felício. **Estado Fiscal e Supremo Tribunal Federal**. São Paulo: Saraiva, 2017.

Flanagan; David, **JavaScript: O Guia Definitivo**, Porto Alegre - RS: Bookman Editora LTDA, 2013.

FOWLER, M. **UML Essencial: Um Breve Guia para Linguagem Padrao**. [s.l.] Bookman Editora, 2014.

FURTADO, Fernando. **O que é o Docker e quais as vantagens de usá-lo?** Disponível em <<https://www.alura.com.br/artigos/comecando-com-docker>> Acesso em: 10 nov. 2023.

Genong, Y., Liping, D., Zhang, B. & Wang, H. (2010). **Coordination Through Geospatial Web Service Workflow in the Sensor Web Environment**. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 3(4): 433–441.

GÜTSCHOW, F. **Escrituração Contábil Fiscal: como cumprir o prazo de entrega**. Disponível em: <<https://bhub.com/blog/escrituracao-contabil-fiscal/>>. Acesso em: 3 dez. 2023.

HENRIQUE DE CAMPOS JÚNIOR, C. Universidade do Estado do Rio de Janeiro. **Estrutura tributária brasileira: uma análise de seu impacto sobre a distribuição de renda**. Disponível em: <<https://www.btdt.uerj.br:8443/bitstream/1/18289/2/Disserta%20c3%a7%20a3o%20-%20Carlos%20Henrique%20de%20Campos%20J%20c3%banior%20-%202020%20-%20completa.pdf>>.. Acesso em: 6 nov. 2023.

Imagem do Docker e contêineres – Diferença entre tecnologias de implantação de aplicações – AWS. Disponível em: <<https://aws.amazon.com/pt/compare/the-difference-between-docker-images-and-containers/>>. Acesso em: 10 nov. 2023.

InfoMoney. **IOF: o que é, quando é cobrado e como o imposto é calculado**. <<https://www.infomoney.com.br/guias/iof/>>. Acesso em: 09/10/2023.

InfoMoney. **ITCMD: como funciona o imposto sobre heranças e quem deve pagar** <<https://www.infomoney.com.br/guias/itcmd/>> Acesso em: 09/10/2023.

Kawabata, T. (2015). **Deteção de outliers espaciais: refinamento de similaridade e desempenho**. 79 p. Dissertação (Mestrado em Ciência da Computação) –Universidade Estadual Paulista “Julio de Mesquita Filho”, São José do Rio Preto, SP.

LEMOS, F. M. L.; BRITO, L. R. S.; SANTOS, J. F. **O Cumprimento das Obrigações Acessórias no Brasil: Uma Análise dos Desafios Enfrentados pelas Empresas**. Revista Científica Multidisciplinar Núcleo do Conhecimento, v. 5, n. 9, p. 81-97, 2020.

Lunelli, Reinaldo Luiz. **A ATUALIZAÇÃO DO QUINTO**

Disponível

em:

<<https://www.portaltributario.com.br/artigos/atualizacaodoquinto.htm#:~:text=A%20reserva%20do%20%22Quinto%22%20pela,prata%20ou%20qualquer%20outro%20metal>>.

Acessado em: 11 jun. 2023.

MACIEL, RENATO DE JESUS; CARVALHO, CARLOS EDUARDO. **ANÁLISE DOS PRINCIPAIS IMPOSTOS QUE COMPÕEM A ESTRUTURA TRIBUTÁRIA BRASILEIRA**. Artigo, 2012. 11 pág.

MACHADO JUNIOR, W. et al. **Controle de estoque: gestão de processos utilizando a ferramenta Kanban com o suporte da metodologia ágil Scrum**. Research, Society and Development, v. 8, n. 1, p. e2381531, 1 Jan 2019. Disponível em: <<https://rsdjournal.org/index.php/rsd/article/view/531>>. Acesso em 03 dez. 2023.

NEVES, Thiago et al. **Complexidade do sistema tributário brasileiro: um estudo exploratório com empresas de pequeno e médio porte**. Revista de Contabilidade e Organizações, v. 15, n. 1, p. 1-18, 2021.

Nogueira, D. L. (1988). **Ferramentas automatizadas para apoio ao projeto estruturado: uma aplicação do diagrama de entidade-relacionamento**. 336f. Tese (Doutorado em Ciências em Engenharia de Sistemas e Computação) –Universidade Federal do Rio de Janeiro, Rio de Janeiro, RS.

PagBank. **O que é IPVA? Descubra essa resposta e mais detalhes sobre o imposto veicular**. <<https://blog.pagseguro.uol.com.br/o-que-e-ipva/>>. Acesso em: 09 out. 2023.

Pavel Luzanov, Egor Rogov, Igor Levshin (translated by Liudmila Mantrova), PostgreSQL: **The First Experience**, 08.2020 169 p. ISBN 075-2063-327-56-2.

QuintoAndar. **O que é ITBI: entenda como funciona o Imposto sobre a Transmissão de Bens Imóveis.** <<https://conteudos.quintoandar.com.br/itbi/>>. Acesso em: 09 out. 2023..

RAMOS, Aléssio Tony et al. **A carga tributária brasileira: uma análise da contribuição das empresas para o desenvolvimento socioeconômico do país.** Revista Brasileira de Finanças, v. 19, n. 1, p. 1-20, 2021.

Receita Federal. **Arrecadação federal alcança R\$ 1,878 trilhão em 2021, alta real de 17,36% sobre o ano anterior:**

<<https://www.gov.br/pt-br/noticias/financas-impostos-e-gestao-publica/2022/01/arrecadacao-federal-alcanca-r-1-878-trilhao-em-2021-alta-real-de-17-36-sobre-o-ano-anterior#:~:text=RECEITA%20FEDERAL-,Arrecada%C3%A7%C3%A3o%20federal%20alcan%C3%A7a%20R%24%201%20C878%20trilh%C3%A3o%20em%202021%20alta%20real,36%25%20sobre%20o%20ano%20anterior&text=O%20recolhimento%20total%20das%20Receitas,resultad o%20de%20dezembro%20de%202020>>.

Acesso em: 09 out 2023.

ROCHA, D. P.; SOUSA, J. C. **Gestão Da Qualidade: A Importância do Método Kanban como Ferramenta Gerencial.** Id on Line Rev. Mult. Psic., vol.15, n.55, p. 449-468, ISSN:1981-1179.

ROSSI, B. **PROPOSTA DE EMENDA CONSTITUCIONAL N.º 45, DE 2019.** Disponível em: <https://www.camara.leg.br/proposicoesWeb/prop_mostrarintegra?codteor=1728369&filenam>. Acesso em: 20 nov. 2023.

SANT'ANNA, Paula; PEREIRA, Luis Felipe Vital Nunes. **Compliance Tributário nas Empresas Brasileiras: Desafios e Oportunidades.** Revista de Administração Mackenzie, v. 22, n. 3, p. 1-29, 2021.

SANTOS, A. V. **Análise de modelos de séries temporais para a previsão mensal do imposto de renda.** Fortaleza: UFC, 2003

SCHWAB, Klaus. **A Quarta Revolução Industrial**. Trad. Daniel Moreira Miranda. São Paulo: Edipro, 2016.

Serasa. **IPTU: o que é e quem tem que pagar?**

<<https://www.serasa.com.br/blog/iptu-o-que-e-quem-tem-que-pagar/>>

Acesso em: 09/10/2023

Serasa. **O que é e quem paga o ISS (Imposto Sobre Serviços)?**

<<https://www.serasaexperian.com.br/blog-pme/iss/>>

Acesso em: 09/10/2023

SETZER, CORRÊA DA SILVA. **Bancos de Dados: Aprenda o que São, Melhore seu Conhecimento, Construa os Seus**. Edgard Blücher, 2005.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S.. **SISTEMA DE BANCO DE DADOS**. 3. ed. São Paulo: Makron Books, 1999.

SITECONTABIL. Sitecontabil - **ARTIGO CONTÁBIL - Impostos diretos e indiretos: quais são as diferenças e como eles afetam as empresas?** Disponível em: <https://www.sitecontabil.com.br/noticias_empresariais/ler/artigo-contabil---impostos-diretos-e-indiretos--quais-sao-as-diferencas-e-como-eles-afetam-as-empresas->. Acesso em: 6 nov. 2023.

Silva, W.C. (2008). **Uma ferramenta para geração automática de modelo conceitual de classes baseada em processamento de linguagem natural**.2008. 117p. Dissertação (Mestrado em Ciência da Computação) –Universidade Metodista de Piracicaba, Piracicaba, SP.

Sordi, J. O., Picchiali, D., Costa, M. A. M. & Sanches, M. A. (2009). **Competências críticas ao desenvolvimento de mapas cognitivos de redes interorganizacionais**. Revista de Administração Pública, 43(5): 1181-1206

TCU, S. **TCU aponta os principais problemas na gestão tributária federal entre 2015 e 2021**. Portal TCU. Disponível em: <<https://portal.tcu.gov.br/imprensa/noticias/tcu-aponta-os-principais-problemas-na-gestao-tributaria-federal-entre-2015-e-2021.htm>>.

Acesso em: 3 dez. 2023.

TÔRRES, Heleno Taveira et alli. “**Sistema Tributário e Direitos Fundamentais no Constitucionalismo Comparado**”. In: TÔRRES, Heleno Taveira (Coord.). Sistema Tributário, legalidade e direito comparado: entre forma e substância. Belo Horizonte: Fórum, 2010. p. 21-76.

Torres, Vitor. **ICMS: o que é e como calcular este imposto?**

<<https://www.contabilizei.com.br/contabilidade-online/icms/>>

Acesso em: 09/10/2023

Tesouro nacional. **Carga tributária bruta do Governo Geral atinge 33,71% do PIB em 2022**

<<https://www.gov.br/tesouronacional/pt-br/noticias/carga-tributaria-bruta-do-governo-geral-atinge-33-71-do-pib-em-2022>>

Acesso em: 09/10/2023

VARSAÑO, Ricardo. **A EVOLUÇÃO DO SISTEMA TRIBUTÁRIO BRASILEIRO AO LONGO DO SÉCULO: ANOTAÇÕES E REFLEXÕES PARA FUTURAS REFORMAS**. IPEA, 1996. 37 pág.