UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE
TECNOLOGIAS DA INFORMAÇÃO E COMUNICAÇÃO

MATEUS SCHEFFER FREITAS

Desenvolvimento de um Sistema Web para Gestão de Notificações Tributárias em um Município do Extremo Sul Catarinense

Araranguá
2024

MATEUS SCHEFFER FREITAS

Desenvolvimento de um Sistema Web para Gestão de Notificações Tributárias em um Município do Extremo Sul Catarinense

Trabalho de Conclusão de Curso submetido ao curso de graduação em Tecnologias da Informação e Comunicação do Centro de Ciências, Tecnologias e Saúde do Campus Araranguá da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Tecnologias da Informação e Comunicação.

Orientador(a): Prof. Dr, Fabrício Herpich

Araranguá

2024

Freitas, Mateus Scheffer Freitas
Desenvolvimento de um Sistema Web para Gestão de Notificações Tributárias em um Município do Extremo Sul Catarinense / Mateus Scheffer Freitas Freitas ; orientador, Fabrício Herpich, 2024.
105 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Tecnologias da Informação e Comunicação, Araranguá, 2024.

Inclui referências.

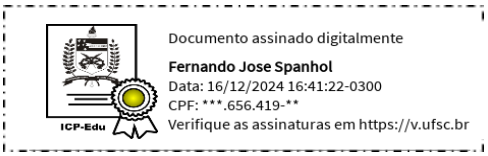
1. Tecnologias da Informação e Comunicação. 2. Sistema Web. 3. Tecnologia Tributária. 4. Notificações Tributárias. I. Herpich, Fabrício. II. Universidade Federal de Santa Catarina. Graduação em Tecnologias da Informação e Comunicação. III. Título.

Mateus Scheffer Freitas

Desenvolvimento de um Sistema Web para Gestão de Notificações Tributárias em um Município do Extremo Sul Catarinense

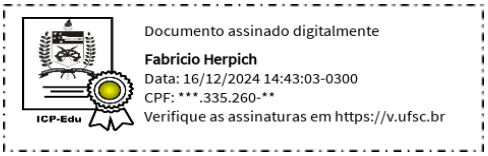
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel e aprovado em sua forma final pelo Curso de Tecnologias da Informação e Comunicação.

Araranguá, 16 de Dezembro de 2024.



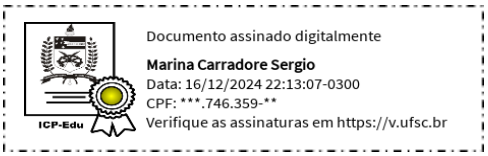
Coordenação do Curso

Banca examinadora



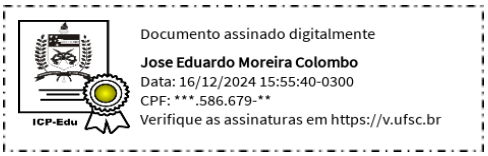
Prof. Fabrício Herpich, Dr.

Orientador



Profa. Marina Carradore Sérgio, Dra.

Universidade Federal de Santa Catarina (UFSC)



Prof. José Eduardo Moreira Colombo, Me.

Universidade Federal de Santa Catarina (UFSC)

Araranguá, 2024.

Dedico este trabalho ao meu pai, Renato Freitas, que se foi de forma precoce para os braços de Deus. Com amor de seu Filho.

AGRADECIMENTOS

Agradeço primeiramente a Deus, meu maior guia espiritual, e aos meus guias espirituais que estiveram comigo em todo momento.

Agradeço a minha família, minha mãe Silvia A. S. Freitas e ao meu querido pai Renato Freitas, minha esposa Érica Larissa de Souza Lima e a minha filha Cloe de Souza Scheffer, sem eles tudo seria em vão, e através da força deles continuei até o final.

Agradeço aos professores que ao longo destes anos me deram suporte de todas as formas, em especial ao Prof. Dr. Fabrício Herpich; Ao Prof. Dr. Marcus Vinicius e a Profa. Dra. Marina Carradore Sérgio.

Agradeço a banca Dra. Marina Carradore Sérgio e ao Me. José Eduardo Moreira Colombo, por aceitarem o convite para a banca avaliadora deste presente trabalho.

Agradeço aos seguranças terceirizados que atuam nas noites da UFSC, que me amparam e comigo geraram uma amizade.

Por fim, agradeço aos meus colegas que fizeram parte desse evento, que se chama graduação.

RESUMO

Este trabalho surge através da experiência pessoal do autor em um setor de tributos de um município catarinense, na qual se fez necessário o desenvolvimento de um sistema web para organização e registro de notificações tributárias, visando melhorar a eficiência administrativa. Tem como objetivo o desenvolvimento de um sistema web, bem como a especificação de suas funcionalidades funcionais e não funcionais, casos de uso e regras de negócio. A metodologia utilizada foi o Design Science Research Methodology (DSRM), que contém um processo bem definido em seis passos, que auxilia a criação do projeto desde seu planejamento, sua execução e comunicação. A avaliação do sistema foi realizada por meio de questionário, cujos resultados indicaram através da escala Likert com uma considerável maioria de notas como “concordo”, indicando que o sistema auxilia em maior organização, produtividade e melhoria no desempenho das atividades do setor. Apesar de algumas funcionalidades não terem sido implementadas devido a limitações de tempo e recursos, o sistema cumpriu seus objetivos e atendeu às necessidades do setor, com seus requisitos funcionais, não-funcionais e regras de negócio sendo aplicados, e uma avaliação positiva por parte da entrevistada.

Palavras-chave: Sistema Web; Tecnologia Tributária; Notificações Tributárias.

ABSTRACT

This work arises from the author's personal experience in a tax sector of a municipality in Santa Catarina, in which the development of a web system became necessary for the organization and registration of tax notifications, aiming to improve administrative efficiency. Its objective is the development of a web system, as well as the specification of its functional and non-functional functionalities, use cases, and business rules. The methodology used was the Design Science Research Methodology (DSRM), which contains a well-defined process in six steps that supports the creation of the project from its planning, execution, and communication. The system's evaluation was conducted through a questionnaire, whose results indicated, through the Likert scale, a significant majority of responses such as "agree," indicating that the system contributes to greater organization, productivity, and improvement in the sector's performance. Although some functionalities were not implemented due to time and resource limitations, the system achieved its objectives and met the sector's needs, with its functional requirements, non-functional requirements, and business rules applied, and a positive evaluation from the interviewee.

Keywords: Web System; Tax Technology; Tax Notifications.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
CETIC	Centro de Estudos sobre as Tecnologias da Informação e da Comunicação
CSS	Cascading Style Sheets
DSRM	Design Science Research Methodology
FEBRABAN	Federação Brasileira de Bancos
HTML	HyperText Markup Language
MTV	Model, Template & View
SQL	Structured Query Language
TIC	Tecnologia da Informação e Comunicação
W3C	World Wide Web Consortium
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

SUMÁRIO

1. INTRODUÇÃO

O sistema tributário nacional caracteriza-se por sua complexidade e baixa transparência. Com base nessa perspectiva, foi constatada, em uma experiência profissional realizada em uma prefeitura localizada no extremo sul do estado de Santa Catarina, a existência de uma pendência relacionada à organização, ao acesso e ao registro de notificações no setor de tributos. Diante desse contexto, identificou-se a necessidade de desenvolver um sistema web para solucionar o problema identificado.

Cada ente da federação tem competências sobre algum grupo de tributos definido pela constituição federal (BRASIL, 1988), bem como a existência de profissionais para a fiscalização que averigua e garante o pagamento destes tributos.

A notificação tributária se deriva de um processo de fiscalização que encontrou algum tipo de pendência, limitando um prazo para o pagamento desta. Cada ente tem seus próprios regulamentos para o tipo de notificação, como se notifica, prazos e para que tipos de pendências.

No século XXI, os governos devem se atualizar e buscar novas tecnologias, pois “ [...] cresce o desenvolvimento de políticas que usam as chamadas Tecnologias da Informação e estabelecem padrões que visam a construção de uma arquitetura na qual o acesso à informação pelos cidadãos é facilitado[...]” (Comunitas, 2024, p.18). Sendo assim, o sistema web trouxe o acesso transparente para o contribuinte e a organização da própria administração.

1.1 OBJETIVOS

1.1.1. Objetivo Geral

Desenvolver um sistema web para organização, acesso e registro de notificações tributárias para um setor de tributos de um município do extremo sul catarinense.

1.1.2. Objetivos Específicos

- Identificar requisitos funcionais e não funcionais para o desenvolvimento de um sistema web de notificações tributárias;
- Elaborar as regras de negócios aplicáveis ao sistema, garantindo conformidade com as normas tributárias e a adequação aos processos da entidade foco;
- Desenvolver e integrar funcionalidades de cadastro, consulta, edição e exclusão de notificações tributárias, assegurando a consistência dos dados e a rastreabilidade das informações registradas;

.

1.2 JUSTIFICATIVA

O Sistema Tributário Nacional em todas as esferas são complexos e muitas vezes não entendidos por todos, dentro deste sistema se tem diferentes partes de atuação e de legislação, uma delas é a notificação tributária que se molda às suas particularidades conforme o estado e ainda mais importante o município. Como diz Testa *et al*. (2014):

> “O Sistema Tributário Nacional, então, é formado pelas normas que cercam a instituição, fiscalização e arrecadação dos direitos, e se relacionam com as normas que asseguram os direitos e as garantias fundamentais do contribuinte, estabelecendo, inclusive, limitações ao poder do Estado de tributar” (Testa *et al*., 2014)

A notificação de lançamento é um documento que formaliza a cobrança de crédito tributário (‘impostos’) e a aplicação de penalidade (‘multas’) (RECEITA FEDERAL, 2024). Conforme cada código tributário de cada esfera pode se ter a separação entre notificação e autuação, que seria a realização da cobrança de multa, bem como particularidades na forma de notificar.

Entretanto, em muitos municípios não há uma organização e nem uma forma transparente de acesso para a pessoa notificada. Na experiência profissional do autor desse projeto em um setor de tributos de uma determinada prefeitura de um

município catarinense, observou-se a falta de organização administrativa para catalogação e acompanhamento, bem como a falta de transparência e validação de um documento oficial por parte do contribuinte. Assim como Comunitas (2024, p. 45) colocam em sua cartilha:

> “No Brasil, especificamente, para se compreender o cenário atual da administração pública e o nível de maturidade em utilização de ferramentas de tecnologia da informação, os conceitos como governança e accountability ganham importância. O processo de redemocratização, estimulado pela Constituição Federal de 1988, a melhoria da governança pública instigou o aumento da pressão da sociedade por maior transparência e qualidade dos serviços prestados aos cidadãos.“ (Comunitas, 2024, p.45).

Segundo a Febraban (2023), o acesso a internet dos brasileiros chegaram em 2023 a 90%, sendo assim, se fez necessário a construção de um sistema web para a organização, catalogação e acesso de notificações tributárias, na qual se levou em conta que o acesso a internet traz a transparência necessária que o serviço público pode se adequar.

Nesta perspectiva, este projeto visa promover a transparência e a fiscalização de ambas as partes envolvidas, tanto o contribuinte como o setor de tributos da prefeitura. Através deste projeto, almeja-se facilitar não apenas regularizar o tempo devido, mas também monitorar o aumento de receita tributária para a administração e a diminuição de dívidas ao longo dos anos, referente a cada contribuinte.

1.3 ESTRUTURA DO TRABALHO

Este trabalho é estruturado em diversas formas que buscam explicitar a completitude da obra, e é organizado da seguinte forma: Na seção 2 se tem o Referencial teórico, em que se explicita conceitos e ferramentas a partir de um estudo bibliográfico; Na seção 3 se tem a parte da Metodologia, na qual se expõe qual a metodologia utilizou-se como base para construção deste trabalho; Na Seção 4, temos o Projeto e Desenvolvimento, na qual será analisado e exposto as particularidades do sistema, seus requisitos, suas regras e seus projetos; Na Seção 5, na qual se é exposto os Resultados e Demonstração, demonstrando os artefatos gerados a partir do projeto; Na seção 6, a Avaliação do trabalho será exposta a partir de testes e conceitos; Na Seção 8, se encontra o fechamento do trabalho com as considerações finais.

2. REFERENCIAL TEÓRICO

2.1. INTERNET E WORLD WIDE WEB

Em toda história humana um fator primordial nos fez evoluir em todos nossos aspectos, que é a comunicação, da mais primordial até a mais complexa, nos trouxe novas formas de consumir e criar informação. “Em linhas gerais, os pré-requisitos para que a comunicação exista são a presença do emissor, do meio, da mensagem e do receptor. Sem qualquer um desses fatores o ciclo não se completa, portanto, a comunicação não acontece” (ALVES, 2013).

A internet é a transferência da comunicação para a área digital, mantendo ainda a linha de emissor, meio e receptor. Segundo a *University System of Georgia* (2024), a internet é mais do que a World Wide Web, e sim todos os protocolos atuantes do sistema criado.

A rede mundial funciona por meio de padronização, que são chamados de protocolos, na qual se tem um conjunto de respostas e perguntas, que são feitas durantes as comunicações, bem como a aplicação de uma regra rígida, para não haver perda de contato com a rede e manter a transferência de dados constante.

A World Wide Web é a parte interativa de todo o sistema “internético”, em que o dinamismo se encontra com a padronização. Criada em 1989, a WWW (World Wide Web) se beneficiou dos protocolos de rede da internet e ao mesmo tempo de tecnologias de hipertexto(textos ricos em imagens, vídeos e simbologias).

A WWW se fez não só possível como continuou expandindo, por causa das tecnologias acessórias que foram criadas para dinamismo e o segmento de multi-conteúdo, aumentando a possibilidade de imagens, vídeos, áudios, gifs e muitas outras formas. O uso de tecnologias como linguagens de programação, linguagens de marcação e linguagens de folha de estilo, mudaram a maneira que interagimos com o digital e com a vida real.

2.2. LINGUAGEM DE MARCAÇÃO

Segundo a Encyclopedia Britannica (2024), a linguagem de marcação é aquela que utiliza símbolos que formatam um documento de texto para controlar seu formato, estrutura ou relacionamentos.

Este tipo de linguagem não pode ser confundida com linguagem de programação, pois apenas formatam um documento multimídia, enquanto a última tem como objetivo executar instruções diretamente no computador para que faça alguma tarefa em específico, tratando dados e alocação de memória.

As linguagens de marcação “possuem um ancestral comum chamado SGML (Standard Generalized Markup Language), um padrão internacional independente de sistemas e máquinas para a definição de métodos de representação de textos em formato eletrônico” (Furgeri, 2006) , que surgiu na década de 80, e foi utilizado de base para as novas tecnologias de marcação, alguns exemplos são HTML, XHTML, XML, XSLT.

2.2.1. HTML

A linguagem de marcação mais complexa é o HTML, chamada de linguagem de marcação de hipertexto, “é o bloco de construção mais básico da web. Define o significado e a estrutura do conteúdo da web.” (MOZILLA, 2024). É uma das linguagens de marcação mais importantes da história, não só criando como fez parte da web ainda no século XX.

Segundo a Boston University (2024) a linguagem foi criada em 1991 por Tim Berners-Lee, na versão 1.0, junto com outras ferramentas compôs a criação da Web . Versões e variações surgiram ao longo do tempo, que contém as versões 2.0, 3.0, 4.0 e 5.0, bem como a variação XHTML, que se combina com outra linguagem, o XML.

Seu funcionamento ocorre através de tags, onde cada uma serve para delimitar algum tipo de característica em uma documentação, cada uma dessas tags trazem uma instrução de como o navegador deve interpretar, assim podendo definir o que é texto, bem como as características destes, o que é imagem, vídeos e outros tipos de dados.

O HTML5, sua última versão, trouxe uma linguagem de marcação mais dinâmica, com algumas funcionalidades que vai além de uma demarcação de documento, “ele visa proporcionar suporte total e integração com CSS e Javascript e eliminar a necessidade de plugins para muitas situações comuns, incorporando

nativamente vários recursos que antes dependiam de um plugin” (MOZER; LOPER; SILVA, 2014).

Como não há mais a necessidade de plugins externo, foi necessário trazer tecnologias que a nova web precisava, as novidades vieram cheio de inovações, como os elementos canvas e svg, que respectivamente permite a criação de gráficos de forma dinâmica, e a criação de vetores através de outros atributos internos. Bem como a criação de elementos para armazenamento interno, ou seja, no próprio navegador.

Sendo assim, a linguagem tornou-se atrativa e complexa, se adequando a um novo tipo de web e a um público cada vez mais ligado a tecnologia digital e da internet.

2.3. LINGUAGEM DE FOLHA DE ESTILO

A linguagem de folha de estilo, assim como a linguagem de marcação, não é considerada uma linguagem de programação, mas sim uma linguagem estética que cria uma apresentação visual de uma documentação.

Segundo a W3C (2024) esse estilo de linguagem é um grande avanço para os designers de web, pois no início da internet não haveria a preocupação com a parte visual, e sim o conteúdo que a rede trazia, mas conforme pessoas de outros âmbitos começaram a navegação, se fez necessário um visual com melhor usabilidade.

O funcionamento deste tipo de linguagem se utiliza da estrutura criada pela linguagem de marcação e acrescenta características visuais, que são instruções para o navegador interpretar e apresentar um visual mais acessível ao sentido humano, é possível também utilizar em conjunto com linguagens de programação web ou em frameworks para mobile.

2.3.1. CSS

O CSS é uma das linguagens de estilo mais utilizadas no mundo, chamada de folhas de estilo em cascata, e é “usada para descrever a apresentação de um documento escrito em HTML ou em XML [...]” (MOZILLA, 2022).

A linguagem contém três versões, sendo a última, CSS3 a mais completa e usada em toda estrutura da web. A sintaxe do CSS se utiliza de um seletor, que é aquele elemento que recebe as estilizações. Se tem as propriedades que são quais características serão modificadas, sendo que cada uma pode ser associada com um valor específico, modificando visualmente o seletor escolhido.

Conforme Verou (2015) o CSS tem suas padronizações criadas por um grupo de trabalho interno, chamado de CSS WG, na qual se tem 98 membros no total, sendo 86 membros de companhias membras da W3C, ou seja, membros de companhias tecnológicas, vendedores de browsers, websites populares e entre outros.

A linguagem foi uma revolução para o design e para usabilidade na web, foi através dela que foi possível organizar visualmente a documentação na internet “primitiva” e a partir disso se deu a possibilidade de tornar a internet popular e acessível a todos os públicos, e é por isso que o “CSS é descrito como uma tecnologia fundamental de a World Wide Web (WWW), além de HTML e JS” (Kuparinen, 2023). Sendo a base da internet, ela continua em constante evolução, buscando cada vez mais não só trazer uma internet esteticamente favorável, como também a busca de uma acessibilidade visual que dá o direito para que todos a usem.

2.4. LINGUAGEM DE PROGRAMAÇÃO

Conforme a Encyclopedia Britannica (2024), a linguagem de programação é aquela que é capaz de indicar um conjunto de instruções para o computador. Ao longo do tempo, as linguagens foram se moldando de diferentes formas, mas todas têm como um princípio facilitar a formulação de instruções para o sistema operacional e posteriormente para o hardware.

“[...]Em particular, uma linguagem de programação é a realização sintática de um ou mais modelos computacionais[...]” (Aaby, 2004). Pode ser interpretada ou compilada, a primeira se refere às linguagens que são interpretadas em tempo real por um interpretador, sendo esta interpretação independente do sistema operacional usado.

A forma compilada é aquela que necessita de uma tradução prévia para um estilo de arquitetura computacional ou sistema operacional específico, sendo assim se tem limitações quanto a sua portabilidade. Com advento das novas tecnologias e arquiteturas, essa limitação vem cada vez mais se tornando inexistente.

A linguagem de programação se utiliza ativamente da matemática, dos componentes de hardwares, dos processos dos sistemas operacionais e da estrutura de dados. Quando usamos uma linguagem ela deixa como artefato o código fonte, que “[...] é um conjunto de palavras escritas de acordo com as regras sintáticas e semânticas de uma linguagem” (Gotardo, 2015).

Sendo assim, as linguagens são uma das ferramentas tecnológicas mais importantes da era contemporânea, através delas foi possível evoluir a computação em saltos cada vez menores, e também democratizando o acesso para pessoas de todos os espectros sociais e de culturas diferentes a computação, assim criando sistemas e programas cada vez mais úteis ao dia a dia e a sociedade que está em contexto.

2.4.1. Javascript

Javascript é uma linguagem script utilizada principalmente para o desenvolvimento web, “é uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe” (MOZILLA, 2024).

A linguagem foi criado por Brendan Eich em 1995 através da Netscape Corporation sendo lançado no netscape navigator 2.0 ainda no mesmo ano, com o passar do tempo suas funções começaram a ser aumentada, e o que estava em fase de crescimento acabou se popularizando, a Microsoft lançou um dialeto da linguagem, chamado de JScript, mas que levou a problemas entre os programadores, pois não havia uma padronização do javascript o que prejudicava a interoperabilidade na internet, que é um dos seus pilares.

A organização proprietária do javascript buscou profissionalizar cada vez mais a linguagem e torná-la um padrão de acesso, sendo assim “em 1996, o processo de padronização para JavaScript foi iniciado sob os auspícios da organização de padrões Ecma International. A primeira especificação padrão oficial para a linguagem foi emitida em 1997 sob o nome 'ECMAScript' (Brock; Eich, 2020),

diminuindo assim os problemas de interoperabilidade e de falta de padronização que prejudicava a atuação de navegadores.

Atualmente é frequentemente utilizado no front-end, “JavaScript faz parte da tríade de tecnologias que todos os desenvolvedores Web devem conhecer: HTML, para especificar o conteúdo de páginas Web; CSS, para especificar a apresentação dessas páginas; e JavaScript, para especificar o comportamento delas” (Flanagan, 2013, p.19).

Além de aplicações em outras esferas, como no back-end e no mobile através de frameworks e bibliotecas. Bem como um interpretador da linguagem para funcionamento diretamente no lado do servidor, chamado de Node.js, que traduz a linguagem para o servidor se comunicar.

Com todos esses usos, a partir da década de 2010, o Javascript começou a se tornar popular e multifacetado, como os citados backend e mobile, até como linguagem para uso em projetos de estatísticas e probabilidades, sendo uma das mais utilizadas no mercado e de fácil aprendizado.

2.4.2. Python

A linguagem de programação Python foi criada por Guido van Rossum a partir de 1989. Sendo "[...] uma linguagem de programação interpretada, interativa e orientada a objetos. O mesmo incorporou módulos, exceções, tipagem dinâmica, tipos de dados dinâmicos de alto nível e classes[...]” (PYTHON, 2024).

Apesar de se ter três versões da linguagem, as duas primeiras apresentavam algumas limitações. Sendo assim, “em 2008 foi lançada a versão 3. Essa versão é a primeira reescrita da linguagem sem muita preocupação em manter compatibilidade com versões anteriores, possibilitando uma limpeza e mudanças sem passar pelo tradicional processo de depreciação da versão 2. Melhorias na biblioteca padrão devem ocorrer apenas na versão 3”. (Luz, 2017).

Se usa Python em propósito geral, sem qualquer tipo de limitação, algumas áreas se tornaram mais dependentes da linguagem, como a ciência de dados, inteligência artificial e sua subárea machine learning.

Uma das principais áreas de seu uso é a inteligência artificial, um tipo de tecnologia que se tornou uma tendência mundial, uma das ferramentas mais

utilizadas para a criação dessa nova área, e a principal “vantagem que Python tem sobre outras linguagens é sua versatilidade. Python é uma linguagem portátil e multiplataforma — o que significa que você pode escrever e executar código Python em qualquer sistema operacional com um interpretador Python” (Miller, 2024, tradução nossa).

Sendo assim, o Python é uma das linguagens mais requisitadas para o mundo atual, com suas múltiplas bibliotecas e frameworks se tem um ambiente amplo de situações na qual pode ser usado.

2.5. FRAMEWORK

Um desenvolvimento de sistema pode ser demorado e complexo, “um framework é um design reutilizável de toda ou parte de um sistema, representado por um conjunto de classes abstratas e pela forma como suas instâncias interagem” (Johson,1997) Utiliza como base uma linguagem de programação, marcação ou de estilo, e costuma expandir o horizonte de uso para outros sistemas ou arquiteturas, facilitando sua construção.

Contém bibliotecas, padrões de código e design, bem como ferramentas específicas para o desenvolvimento, desta maneira se otimiza tempo e recursos para produção de qualquer projeto de software ou sistema web.

Segundo o AWS (2024) frameworks tem como benefícios de seu uso, a melhora na qualidade do código, redução do tempo para desenvolver, segurança mais desenvolvida e eficiente, flexibilidade e revisão mais eficiente.

Como um framework expande as funções e o uso de uma linguagem de programação acaba criando categorias necessárias, como para a web, para o mobile ou para o desktop. Utilizando a linguagem python como exemplo, temos os seguintes exemplos, o Django para a criação de sistemas web; Kivy para a criação de sistemas mobile; Tornado para a criação de redes e comunicação com servidor, entre outros.

Em um mundo da informação constante e em tempo real, o desenvolvimento de tecnologias digitais deve acompanhar o mesmo ritmo, sem o uso de frameworks, metade das inovações de curto prazo seriam inviáveis e o acesso a serviços digitais seria deficitário.

2.5.1. Bootstrap

Bootstrap é um framework HTML, CSS e Javascript que desenvolve ‘‘um ambiente responsivo e um site amigável ao mobile” (Gaikwad; Adkar, 2019). Contém recursos de tipografia, cores, formulários, botões e outros,utilizado de forma modularizada, ou seja, fragmentos podem ser utilizados de maneira separada e espalhada pelo projeto, sem seguir completamente um layout.

“Antes de ser uma estrutura de código-fonte aberto, o Bootstrap era conhecido como Twitter Blueprint” (ALURA, 2023), pois foi desenvolvido para a empresa Twitter inicialmente, por meio de um hack week outros funcionários do twitter se juntaram na contribuição e em 2011 foi lançado como código aberto e renomeado para Bootstrap.

Seu maior foco é em estilos responsivos e no conceito mobile first, que implementa o site focado inicialmente ao mobile, com os layouts funcionando bem em todos navegadores de uso geral e as principais larguras de telas usadas no mercado.

Pode ser aliado com outros frameworks ou técnicas, tornando dinâmico o seu uso e aumentando a rapidez de sua utilização. Se tornando utilizado por vários programadores e empresas pelo mundo inteiro o “Bootstrap permite um desenvolvimento rápido e responsivo que é consistente e bem apoiado pelo desenvolvimento e comunidade de design” (Gaikwad; Adkar, 2019).

2.5.2. Django

O framework tem como foco a reutilização de um modelo de código completo ou parcial, o “django é uma estrutura web Python de alto nível que incentiva o desenvolvimento rápido e um design limpo e pragmático” (DJANGO), com uma comunidade massiva e uma quantidade de conteúdo disponibilizado considerável. É uma das ferramentas que se criou para a construção de sistemas web completos.

Segundo a Alura (2023) o framework foi criado em 2003 pela Lawrence Journal World, uma empresa jornalística, através de sua equipe de desenvolvedores,

mas só em 2005 foi adequado para a licença BSD, se tornando aberto ao público final. Foi lançado com vários recursos client-side e server-side. Segundo Luz (2017):

> “O Django segue o padrão de desenvolvimento chamado MTV, Model-View-Template. Similar ao modelo mais famoso, chamado de MVC, Model View Controller. Ambos os conceitos seguem a premissa de separar claramente o desenvolvimento em camadas. Isso é importante para evitar misturar o código que é responsável por mostrar os dados do código que é responsável por acessar dados ou do código responsável por implementar as regras de negócio.” (Luz, 2017).

Qualquer sistema desenvolvido neste framework é disposto como um projeto, e cada projeto tem uma gama de aplicações, parte que será de fato codificada. As aplicações têm os recursos necessários para a construção, como views, models, tests, templates, urls, admin e afins.

Fornece uma gama de funções de segurança cibernética, para a proteção do sistema web, deixa-se o desenvolvedor livre para apenas configurar os recursos necessários. “O Django ativa a proteção contra muitas vulnerabilidades por padrão, incluindo SQL injection (injeção de SQL), cross-site scripting, cross-site request forgery (Falsificação de solicitações entre sites), e clickjacking (furto de click)” (MOZILLA, 2024).

Por todas essas inovações e funções, o framework se tornou um dos mais usados no planeta, segundo a Statista (2024) 12% dos desenvolvedores utilizam o Django em seus projetos, mostrando seu futuro promissor e com melhorias esperadas para outras atualizações.

2.6. SISTEMA WEB

Sistema web, também conhecido como aplicação web ou web apps, “[...] é um software que é executado em um navegador da Web” (AWS, 2024).Pode ser acessado por meio da internet e é altamente escalável, usufruindo da tecnologia Cloud para aumentar a banda de tráfego e de banco de dados. “Sistemas e aplicações baseados na Web (WebApps) produzem uma complexa matriz de conteúdo e funcionalidade para ampla população de usuários finais” (Ribeiro *et a*l, 2015).

Uma aplicação web foge do tradicional site estático ou dinâmico, em que o primeiro apenas se configura em uma página web com recursos inertes, não

atualizados em tempo real e sem interação com o usuário final complexa; Os sites dinâmicos se aproximam dos sistemas web por se ter uma interação mais complexa, a sua principal diferença é que se pode não usar tecnologias cloud, e seu dinamismo são focados em partes específicas do site, funcionando como um site estático com pequenas funções dinâmicas.

Este tipo de sistema foi criado para eliminar a lacuna entre software e sites estáticos. Oferece funcionalidades e elementos de usuário interativos da mesma forma que o software, mas que eram entregues usando um URL de navegador da Web (AWS, 2024).

Ao levar o funcionamento de um software ou aplicação para a internet, muitos serviços acabam sendo democratizados e passam a ter opções suficientes para o mercado não estar dependente de aplicações de softwares de grandes empresas e de depender que pessoas com conhecimentos específicos, como técnicos de informática, sejam necessitados para que instalem em suas máquinas.” Como não é necessário instalar nenhum programa além do navegador, os sistemas web tornam-se altamente acessíveis, já que os usuários só necessitam de um computador (ou celular) e conexão à internet” (EESCJR, 2024).

Ao transformar a web de um pequeno meio de diretório de informações para um campo de aplicações, se abriu as possibilidades do que a internet pode fazer, criando um campo de inovações e revoluções na indústria.

2.7. TICS NO DEPARTAMENTO DE TRIBUTOS MUNICIPAIS

Como diz no artº 24 da lei nº 12.965 de 2014 no inciso X indica que o poder público deve garantir a “prestação de serviços públicos de atendimento ao cidadão de forma integrada, eficiente, simplificada e por múltiplos canais de acesso, inclusive remotos” (CONGRESSO NACIONAL, 2014). Sendo assim os serviços públicos de todas as esferas, tem o dever de informatizar todo seu governo, com transparência e de múltiplas formas.

Segundo o Cetic (2023) em uma pesquisa sobre TICs nas prefeituras municipais do país, foi demonstrado que 92% das prefeituras contém um website, porém é visto que mesmo com essa bom número de informatização, os websites

servem em primeiro grau mais para download de formulários e documentos, do que serviços de fato online, como cadastros e atendimentos diretamente por esse portal.

Neste projeto o foco é a criação de um sistema para notificação tributária, na qual mostra a mesma pesquisa que apenas 57% das prefeituras tem um portal que permite ao contribuinte acompanhar qualquer processo administrativo ou judicial contra ele, ou seja, a notificação emitida para uma empresa em 43% dos municípios não se tem um portal para acompanhamento depois de notificado, e para acompanhar o caso se torna mais difícil e menos transparente.

Quando ocorre o não pagamento do tributo o contribuinte fica em estado de irregularidade, sendo assim, os governos de cada esfera emitem uma infração ou notificação, conforme legislação de cada ente. Cada departamento que notifica tem suas próprias regras internas para garantir o controle de prazos e regularidade, em muitos casos por meios não eficientes, como documentos de texto, e na maioria dos casos o notificado não tem acesso a esse acompanhamento.

Segundo o Art. 145º da Constituição Federal (BRASIL, 1988), o tributos são instituídos na forma de impostos, taxas e contribuição de melhoria, o não pagamento deles, é passível de alguma medida administrativa, como multa ou fechamento do negócio(se for empresa), mas para isso é direito do contribuinte ser previamente avisado e notificado. Para essa notificação é preciso fiscalização ativa e recorrente.

“A fiscalização tributária se materializa em atos de verificação do cumprimento de obrigações tributárias, quer sejam principais, quer sejam acessórias[...]” (Sabbag, 2012). Para cada organização governamental e esfera federal existe regulamentação e regramento em geral para as práticas dos fiscais. Toda ação de um fiscal e de recolhimento de tributos deve ser verificável, ou seja, um departamento tributário deve ter acesso a um sistema que contenha informações dos contribuintes, dos seus tributos e das ações que o fiscal realiza sobre estes.

Portanto, as TICs são necessárias para o serviço público não só para a democratização de acesso aos contribuintes, mas também para a organização, emissão e acompanhamento de casos referentes a ações do poder público, a notificação tributária é uma das partes essenciais para a lei se valer.

2.7.1. Notificação Tributária

Quando ocorre a irregularidade, as esferas têm suas regras referente a ação a ser tomada, havendo a aplicação da multa ou mesmo a notificação prévia à multa, bem como medidas judiciais, “a notificação é o último ato do procedimento de constituição formal do crédito tributário que o torna oponível ao contribuinte” (Enciclopédia Jurídica PUCSP, 2024).

O planejamento fiscal anual é feito pelas esferas no ano anterior, para isso é baseado nos recebimentos de repasses de outros entes, no caso do município e estado, e no recebimento dos tributos. Quando não se há uma fiscalização, é possível que o montante recebido não tenha sido de acordo com o necessário previsto para os gastos, assim os entes precisam sempre estar em fiscalização ativa constante para manter não só a responsabilidade do contribuinte, mas a própria condição financeira.

Cada município tem sua forma de notificar e seus prazos, seguindo também o regramento geral do estado e da união, como exemplo o município de Passo de Torres no seu artº 127 do Código Tributário Municipal (Passo de Torres, 1995) diz “Verificado o descumprimento das obrigações tributárias, será expedida a notificação contra o infrator, para que, no prazo máximo de 30 (trinta) dias, seja regularizada a infração, ou ofereça defesa escrita”.

Visto que as notificações acompanham com dados essenciais, bem como prazo, é viável a área de TIC estar atenta a esse setor e para esta questão específica, trazendo inovações e tecnologia para tornar eficiente.

3. METODOLOGIA

A metodologia adotada para este trabalho consiste na Design Science Research Methodology (DSRM), que traz uma abordagem metodológica focada no desenvolvimento de um artefato na área de tecnologia.

Na qual é um método que busca ser aplicável e gerar um conhecimento para solucionar um problema, através de uma abordagem flexível e sistemática, trazendo uma construção iterativa e avaliação constante. Segundo Peffers *et al.* (2007) as etapas definidas são identificação do problema e motivação, definição de objetivos para uma solução, projeto e desenvolvimento, demonstração, avaliação e por fim a comunicação.

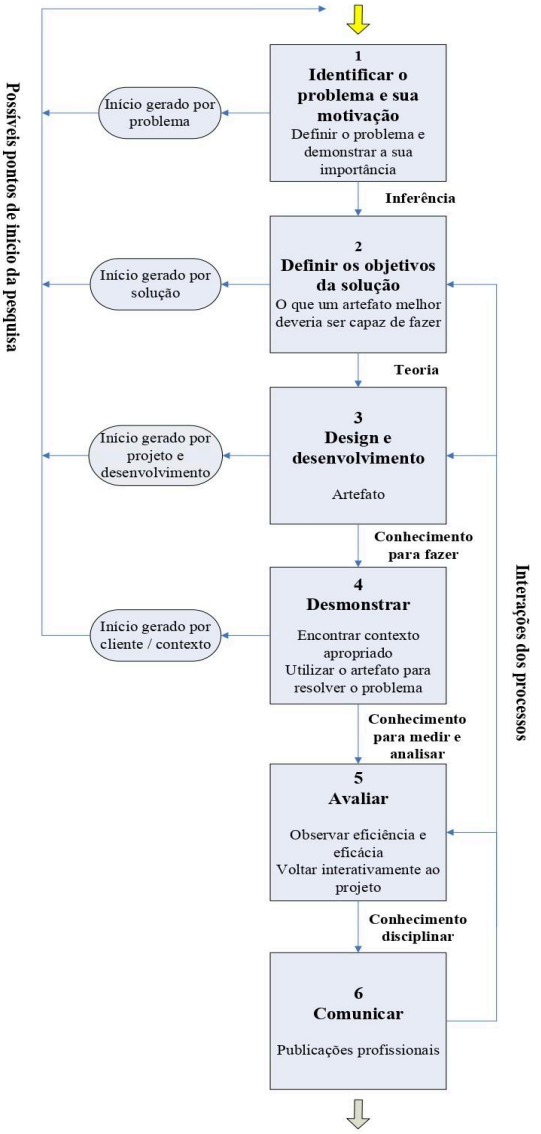
A etapa de identificação do problema e motivação traz uma determinação do problema, que por meio de uma pesquisa anterior da área, é possível delimitar o problema, suas soluções e justificativa.

A definição dos objetivos para uma solução é onde se avalia, a partir do problema, quais os objetivos dos artefatos e do processo até a finalização do desenvolvimento do sistema. Esses objetivos são definidos através de uma visão do que é possível e viável.

O projeto e desenvolvimento é a parte prática e teórica da criação do artefato. Levando em consideração as etapas anteriores, é delimitado um campo de atuação, trazendo elementos como engenharia de requisitos e as ferramentas utilizadas. A demonstração é a etapa que explica o projeto de maneira mais abrangente e aplica uma simulação sobre o caso de uso.

A avaliação é a etapa que traz a percepção do autor e de um usuário do sistema, a partir de um teste para aceitação da tecnologia. Por fim, a comunicação é a parte que, em si própria, trata da divulgação e da publicação da pesquisa realizada. Como podemos ver na *figura 1* que contém todas partes desta metodologia e seu ciclo.

Figura 1 - Representação da Metodologia DSRM



Fonte: Jappur (2014).

Nesta perspectiva, observando as etapas previstas na DSRM, abaixo são descritas as etapas previstas para este projeto.

3.1 IDENTIFICAÇÃO DO PROBLEMA E MOTIVAÇÃO

Esta etapa busca delimitar o problema e trazer a motivação para ser utilizado na etapa de projeto como o artefato final, através de pesquisas anteriores ou a partir de vivência própria.

O problema vivenciado se deu através de uma vivência profissional do autor em um setor de tributos de uma cidade do extremo sul catarinense, na qual foi percebido um déficit sobre a falta de um sistema informático[1] mais dinâmico para anotação de notificações tributárias. Para encontrar a delimitação do problema e a motivação foi utilizado de pesquisa bibliográfica, pesquisa de mercado, bem como interação acerca deste problema com outros colegas do mesmo ambiente profissional.

A pesquisa bibliográfica sobre o tema se deu no referencial teórico, abordando as especificidades do projeto. A justificativa se deu em conjunto a partir da pesquisa bibliográfica e da interatividade com colegas de profissão.

3.2 DEFINIÇÃO DE OBJETIVOS PARA UMA SOLUÇÃO

Nesta etapa se buscou contextualizar e delimitar o problema em metas mais fechadas e assertivas, buscando a eficiência do uso de recursos e de habilidades. O DSRM é flexível em algum nível, sendo assim, a definição de objetivos pode adentrar outras etapas de um projeto.

Neste projeto os objetivos aparecem de forma abstrata na introdução, justificativa e com um direcionamento mais concreto na etapa de objetivos, levando aos poucos para uma delimitação de problema real, é visto que segundo a experiência do autor, o principal objetivo é o desenvolvimento de um sistema web para notificações tributárias.

Ao longo do projeto é definido quais objetivos são viáveis para o momento que é construído, partindo da realidade do autor, da realidade do contexto social atual, das habilidades e ferramentas empregadas.

3.3 DESIGN E DESENVOLVIMENTO

Nesta etapa traz a conceitualização do projeto do sistema de forma mais técnica e explícita, sem o uso de uma teoria densa e sim detalhada nas suas

[1] Sistema informático é um conjunto de ferramentas e gadgets de tecnologia que se utilizam das TICS em geral.

necessidades. Aparecem como parte necessária duas grandes áreas, de projeto e de desenvolvimento.

Na parte de projeto irá ser analisado os requisitos do sistema, de forma mais teórica e visual, onde será analisada a engenharia de requisitos que é a ciência que busca a definição e gerência dos requisitos de um sistema, abrangendo os funcionais, não funcionais, regras de negócios e casos de uso.

Nos requisitos funcionais, é elencados aqueles ações que os usuários podem praticar; Nos requisitos não funcionais são critérios que não dependem de uma ação do usuário, mas que caracterizam o sistema; As regras de negócio, são aquelas que limitam a ação do sistema levando em consideração regras da organização, leis de mercado, leis do governo e afins; E por fim que os casos de uso, que não é uma parte que elenca novos requisitos, mas utilizada sim, uma prototipação de possibilidade de uso de cada usuário abstrato, ou seja, o administrador, fiscal e o usuário final.

3.4 DEMONSTRAÇÃO

Esta etapa tem como objetivo a demonstração prática do sistema desenvolvido, mostrando sua aplicação real em um caso de uso específico. Durante essa fase, será feito um teste controlado, com um fluxo de trabalho estruturado, para demonstrar as funcionalidades principais do sistema em ação. Além disso, será realizada a identificação das partes componentes do sistema, bem como uma explicação abrangente sobre seu funcionamento finalizado, destacando os resultados obtidos e os benefícios esperados com sua utilização.

3.5 AVALIAÇÃO

A avaliação será o uso de um questionário criado através do Google Forms utilizando nas perguntas a escala de Likert, levando em consideração que dentro do departamento e para mesma função só há uma outra integrante, sendo assim será aplicado este questionário para esta colega de profissão, em que será respondida a perguntas pré definidas dividas em três grupos: Usabilidade, utilidade e impacto no

trabalho, e por fim intenção de uso futuro. Será deixado também uma pergunta final em aberto sobre recomendações ao sistema.

3.6 COMUNICAÇÃO

Nesta etapa é explicado o processo de comunicação, ou seja, como essa pesquisa/projeto será publicado e apresentado à comunidade. Este projeto por si só é uma publicação, como um Trabalho de Conclusão de Curso (TCC), sendo posto no repositório da universidade na qual se atribui. A apresentação se dará na defesa deste trabalho para uma banca de professores desta universidade.

4. PROJETO E DESENVOLVIMENTO

Nesta etapa se analisa toda parte prática do projeto, traz as particularidades do sistema, demonstra a concepção de projeto do sistema com a engenharia de requisitos. Através da prototipação do banco de dados e da interface, busca a compreensão lógica do sistema, criando um campo de visão de como funciona. Por fim, traz a parte de desenvolvimento que é a parte prática e que gera resultados que o usuário possa interagir com o sistema imaginado.

O sistema foi criado para organizar, emitir, catalogar e controlar as notificações tributárias de um município que o autor atua, o nome escolhido foi ‘Denotify’, quanto à sílaba ‘De’ vem de democracy ou democracia, e como livre tradução notify vem de notificar, onde se quer passar a sensação de uma ‘gestão democrática pública’ ou uma notificação mais aberta. O sistema funcionará com três tipos de entidades abstratas principais, o fiscal, na qual é quem notifica; a entidade, empresa ou autônomo, quem recebe a notificação, e o usuário controlador da entidade; Cada um terá uma gama de responsabilidades e de acessos ao sistema.

4.1 PARTICULARIDADES DO PROJETO

O projeto desse sistema web para notificações tributárias, tem certas peculiaridades, pois partiu de uma experiência profissional do autor dentro de um departamento de tributos de uma cidade do extremo sul catarinense. Por ser órgão público se mantém certas particularidades advindas da lei municipal e regras internas.

Deve-se entender que uma notificação neste município em questão é gerada a partir de dois problemas constatados, um é a falta de cadastro municipal e outro é a dívida/débito em aberto no sistema.

Uma das particularidades é referente ao código verificador, que deve ser gerado anteriormente, logo se notifica em branco para geração do código e logo depois edita essa notificação com os dados da empresa notificada. Isso ocorre, pois a fiscalização é imediatista, ou seja, ela percorre o município sem um objetivo inicial em vários casos, assim não se sabe qual empresa deve ser notificada até estar presencialmente nela.

Outra particularidade em nível de sistema, é a existência de uma obrigação de pedido de liberação de acesso, que tem como objetivo conectar a conta de um contribuinte com sua empresa, acessando histórico e dados da notificação desta empresa. Esse pedido serve para manter a segurança dos dados, em que um fiscal irá analisar se os dados do usuário que pediu liberação batem com os dados da empresa notificada, levando em consideração uma gama de processos burocráticos fora do sistema.

4.2 ELICITAÇÃO DE REQUISITOS

A elicitação de requisitos é a etapa que busca por meio de materiais externos, possíveis clientes e experiência do autor, requisitos para o funcionamento ativo do sistema e regras que podem vir a ter. Nesta etapa se levou em consideração a experiência profissional do autor, visto que o sistema resolve o problema deste em seu ambiente de trabalho, a partir disso, traz requisitos funcionais, requisitos não-funcionais, regras de negócios, casos de uso.

4.2.1 Requisitos Funcionais

Requisitos funcionais são aqueles que buscam trazer uma visão sobre o que o sistema tem que fazer, ou seja, as ações práticas do sistema. Listam-se abaixo na *tabela 1* os requisitos funcionais deste sistema:

Tabela 1 - Requisitos Funcionais

Nº Requisito	Descrição
RF01	O Sistema deve permitir a emissão de notificação tributária

RF02	O Sistema deve permitir a criação de pareceres sobre cada notificação tributária emitida.
RF03	O sistema deve permitir o cadastro de empresas e autônomo com detalhes compatíveis com cada tipo.
RF04	O sistema deve permitir uma busca pelas empresas cadastradas e com um filtro de pesquisa.
RF05	O sistema deve permitir o cadastro de usuário com informações de email, senha e outros detalhes.
RF06	O sistema deve permitir a busca de usuários cadastrados e ser capaz de ter um filtro de pesquisa.
RF07	O sistema deve ter um validador de notificação, que valida que aquela notificação é verdadeira e do setor correto.
RF08	O sistema deve permitir ao usuário ver o seu histórico de notificação.
RF09	O sistema deve permitir que o usuário e o fiscal peçam liberação de uma conta nova para adentrar ao aplicativo.
RF10	O sistema deve permitir que o fiscal tenha acesso a uma lista de usuários

	que pediram liberação, permitindo aceitar ou não a liberação.
RF11	O sistema deve permitir a geração de um PDF da notificação em conjunto com seu código gerador.
RF12	O sistema deve permitir anexar documentos à notificação emitida.
RF13	O sistema deve permitir a filtragem de notificações pelos seguintes filtros: totais, abertas, regulares e fora do prazo.

Fonte: Do autor.

4.2.2 Requisitos não-funcionais

Os requisitos não-funcionais são aqueles que buscam explicar o sistema a partir de suas características de desempenho, usabilidade, segurança e afins, ou seja, características que não precisam depender de uma ação do usuário diretamente, e nem depender que o requisito seja visto pelo usuário.

Abaixo na *tabela 2* se encontra uma lista de requisitos não funcionais:

Tabela 2 - Requisitos não-funcionais

Nº Requisito	Descrição
RNF01	O sistema deve proteger os dados de acesso com criptografia.

RNF02	O sistema deve ser acessível a todos navegadores de uso popular e em suas três últimas versões.
RNF03	O sistema deve em caso de falha voltar a funcionar em até 1 hora a contar do horário da instabilidade.
RNF04	O sistema deve ter documentação suficiente para que o treinamento de um fiscal seja feito em até 1 hora.

Fonte: Do autor.

4.2.3 Regras de Negócio

Regras de negócios são aquelas regras que o sistema deve seguir tanto em sua codificação, até seu funcionamento geral, levando em conta diretrizes interna da empresa, diretrizes do mercado, leis e cultura local.

Abaixo na *tabela 3* segue a lista de regras de negócio aplicado a esse sistema:

Tabela 3 - Regras de Negócio

Nº Regra	Descrição
RN01	O cadastro de usuário para acessar os dados de uma empresa notificada deve ser analisado por um funcionário

	manualmente, para averiguação e corroboração dos dados.
RN02	Um usuário que não seja staff[2] pode ter acesso a apenas uma empresa notificada e seus dados.
RN03	Uma notificação só pode ser excluída se for causada por algum tipo de erro externo, como a notificação errada.
RN04	Uma notificação será emitida em branco para a geração do código verificador, logo após essa notificação será atrelada a uma empresa, conforme a situação necessária da fiscalização.
RN05	Só quem pode cadastrar ou emitir uma notificação é um usuário com o cargo de staff.

4.2.4 Casos de Uso

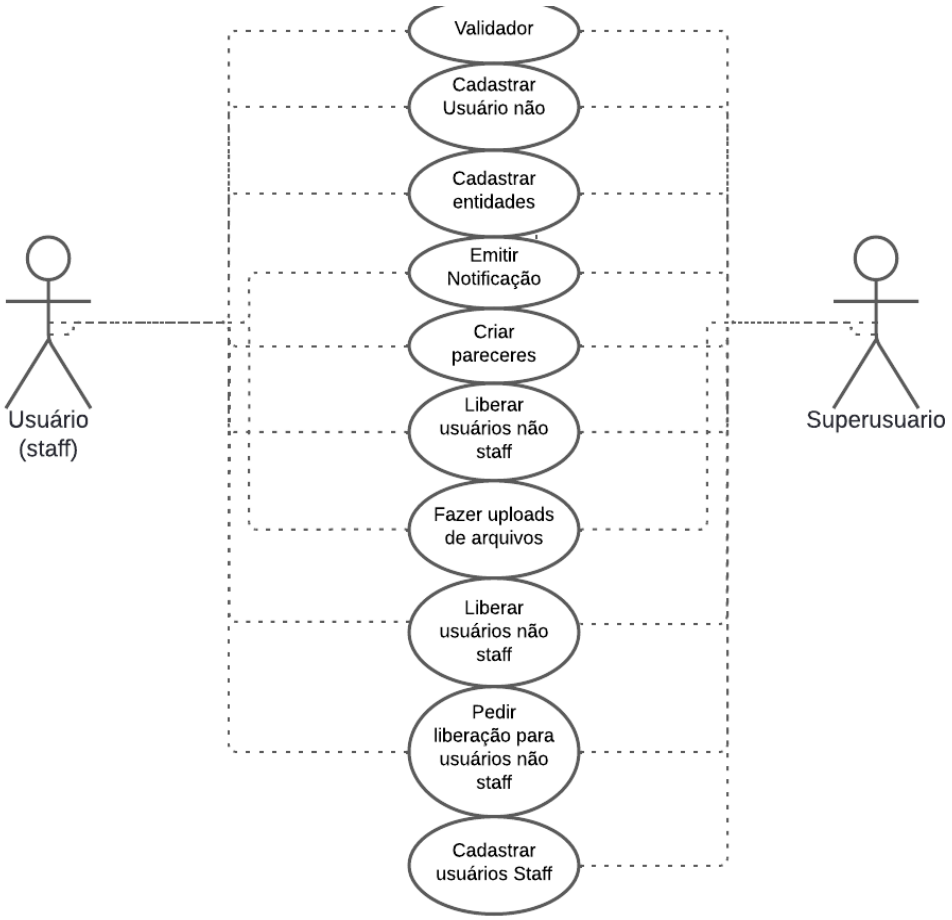
Casos de uso é uma representação visual onde inclui requisitos funcionais, atores e sua interligação entre eles. Um ator pode ser o sistema, ou um atributo do sistema que começa uma rede de acontecimentos ou utilização de funções, bem como pode ser o usuário humano, as interligações com os requisitos funcionais demonstra a quais funções este ator é permitido interagir e utilizar.

No caso de uso abaixo na *figura 2* podemos ver dois atores, o usuário staff, que é aquele que faz parte do quadro de funcionários do sistema, também podendo ser reconhecido como fiscal, e o superusuário que é o administrador, todos estes

[2] Staff é um termo padrão do Django para atribuir um cargo a um usuário, aquele que é Staff recebe acessos a um conjunto de ações que só um administrador do site pode ter.

dois fazem parte do quadro de funcionários em um modo mais geral-, mas o superusuário tem uma hierarquia de cargo acima, por ser o chefe do departamento, e assim o único que pode cadastrar outros usuários que são também funcionários.

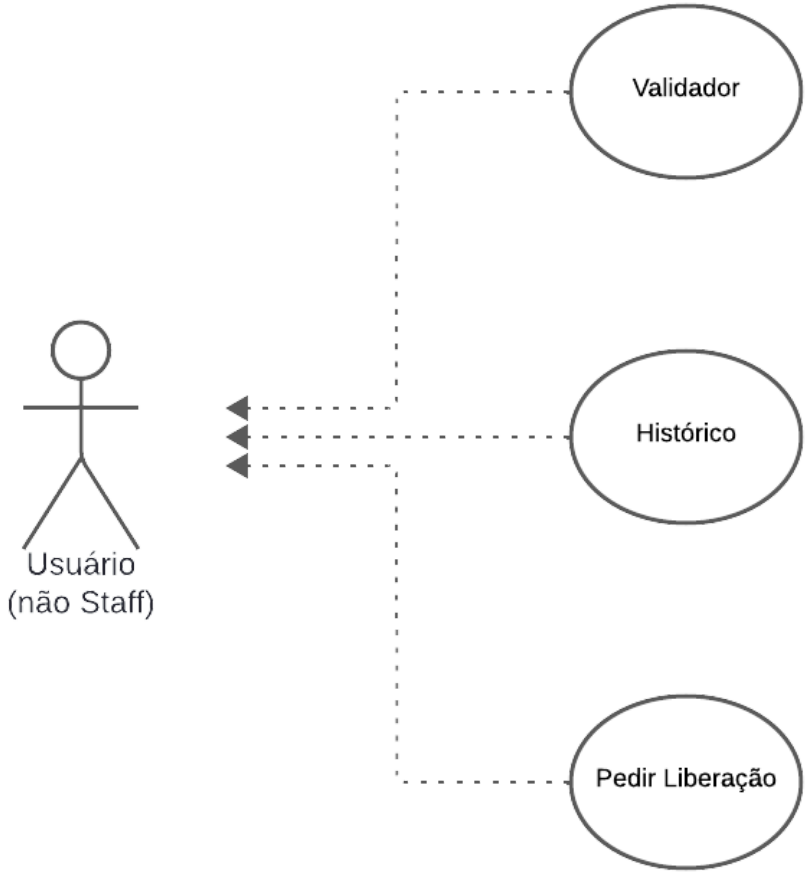
Figura 2 - Caso de uso usuário Staff e SuperUsuário



Fonte: Do autor.

No próximo caso de uso, na *figura 3,* se dá sobre o usuário não staff, ou seja, aquele que representa uma empresa ou autônomo notificado pelo sistema, suas ligações são mais limitadas e menos abrangentes, sendo permitido a conexão com a função de validador, que é usado para validar uma notificação recebida, assim permitido a confiabilidade naquele documento; Histórico, que mantém um controle sobre todas as notificações já recebidas; Além de ser permitido pedir sua própria liberação para acessar sua empresa.

Figura 3 - Caso de Uso do usuário não-Staff



Fonte: Do autor.

Por fim, é visto que temos três atores principais, o administrador,usuário staff e o usuário não staff, com acessos a diferentes tipos de funções baseado em suas permissões e regras de negócios.

4.3 PROJETO DO SISTEMA

Nesta etapa ocorre o projeto da interface gráfica e da modelagem de banco de dados que servirá como base a construção do sistema, é neste estágio que será reconhecido quais tipos de templates precisaremos, baseado em suas funções e

quais entidades de bancos de dados devem ser criados no processo de modelagem no framework Django.

4.3.1. Projeto de Interface Gráfica

O sistema usa como suporte tecnológico para geração de interface gráfico o CSS, Bootstrap e o Bootstrap Studio. Este último é um software de geração de página de sites, utilizando da técnica ‘drag and drop’, que no final pode se importar todo o código gerado, facilitando a criação de produtos visuais, e por fim padronizando em templates.

Foi pensado em interface gráfica levando em consideração três estados do sistema, o estado logado e usuário staff, o logado e usuário não staff, e o não logado. Cada tipo de estado traz semelhanças e também algumas particularidades baseada no tipo de acesso que cada usuário traz.

No Estado não logado se encontra os templates de login, cadastro e validador. No estado logado e staff os templates são dashboard, notificação, parecer, busca geral, cadastro de empresa, cadastro de usuário, validador e liberação; No estado logado e não staff aparecem os templates de dashboard, histórico e validador. No Django será usado um template base para diminuição de codificação e para massificação do CSS e alterações de design.

4.3.2 Projeto de Banco de Dados

O banco de dados do framework do Django vem como padrão o SQlite, que é um banco de dados embutido, ou seja, não precisa de um servidor ou de um sistema de nuvem, ele funciona na própria máquina na qual está sendo codificado o sistema, serve para manter o sistema em teste ou para funcionamento interno.

Posterior a codificação, e ao fazer deploy em um sistema de nuvem será utilizado o postgresql, um gerenciador de banco de dados, que funciona de maneira mais robusta e em um servidor real, não sendo embutido no próprio sistema, e dependente de funções mais avançadas.

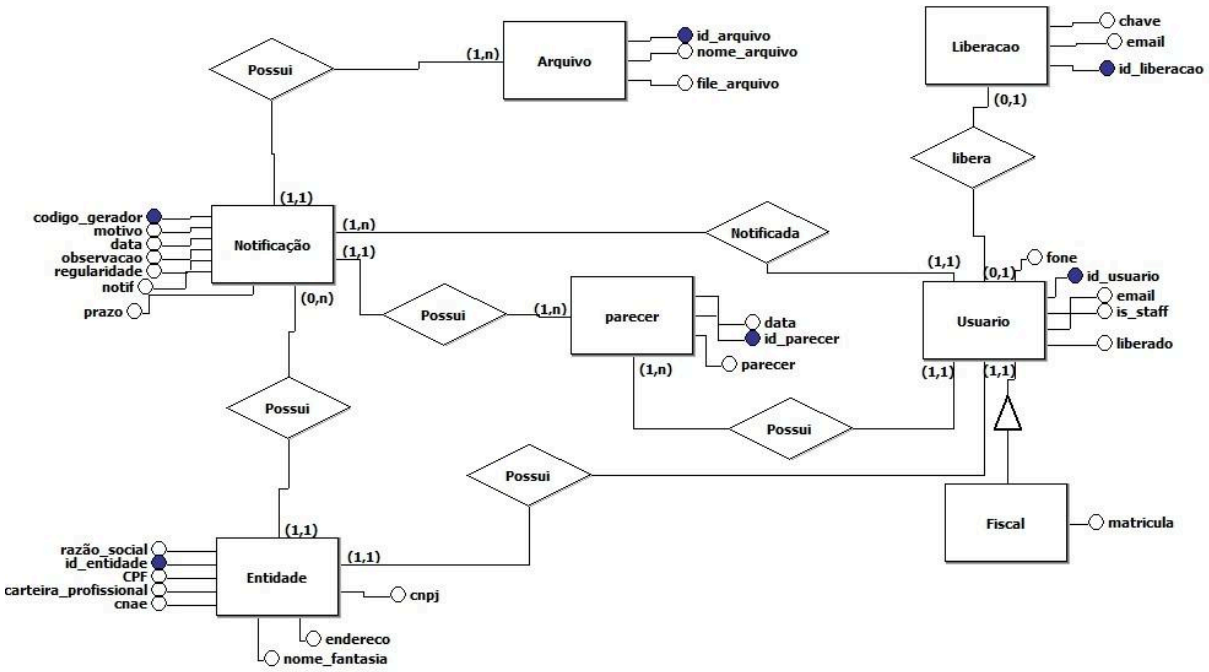
Para estruturação do banco de dados desse sistema foi utilizado a ferramenta BrModelo, que serve para modelagem de banco de dados no seu esquema conceitual que busca representar visualmente as entidades criadas, o esquema

lógico que representa as entidades visuais em lógica de maneira que mostra a conexão real entre eles e identificando o chaveamento correto, e o por fim o esquema físico que a representação na linguagem SQL que pode ser utilizado em um banco de dados ou na programação.

Levando em consideração o uso do framework Django, é utilizado neste trabalho o esquema visual e lógico, para se usar como referência, pois este framework se utiliza de criação de modelos, que é uma representação mais abstrata e feito diretamente na programação.

Abaixo na *figura 4,* se tem o primeiro esquema que é o visual na qual é representado as principais entidades deste projeto, e seus atributos, já definindo as chaves primárias e o tipo de dado de cada atributo.

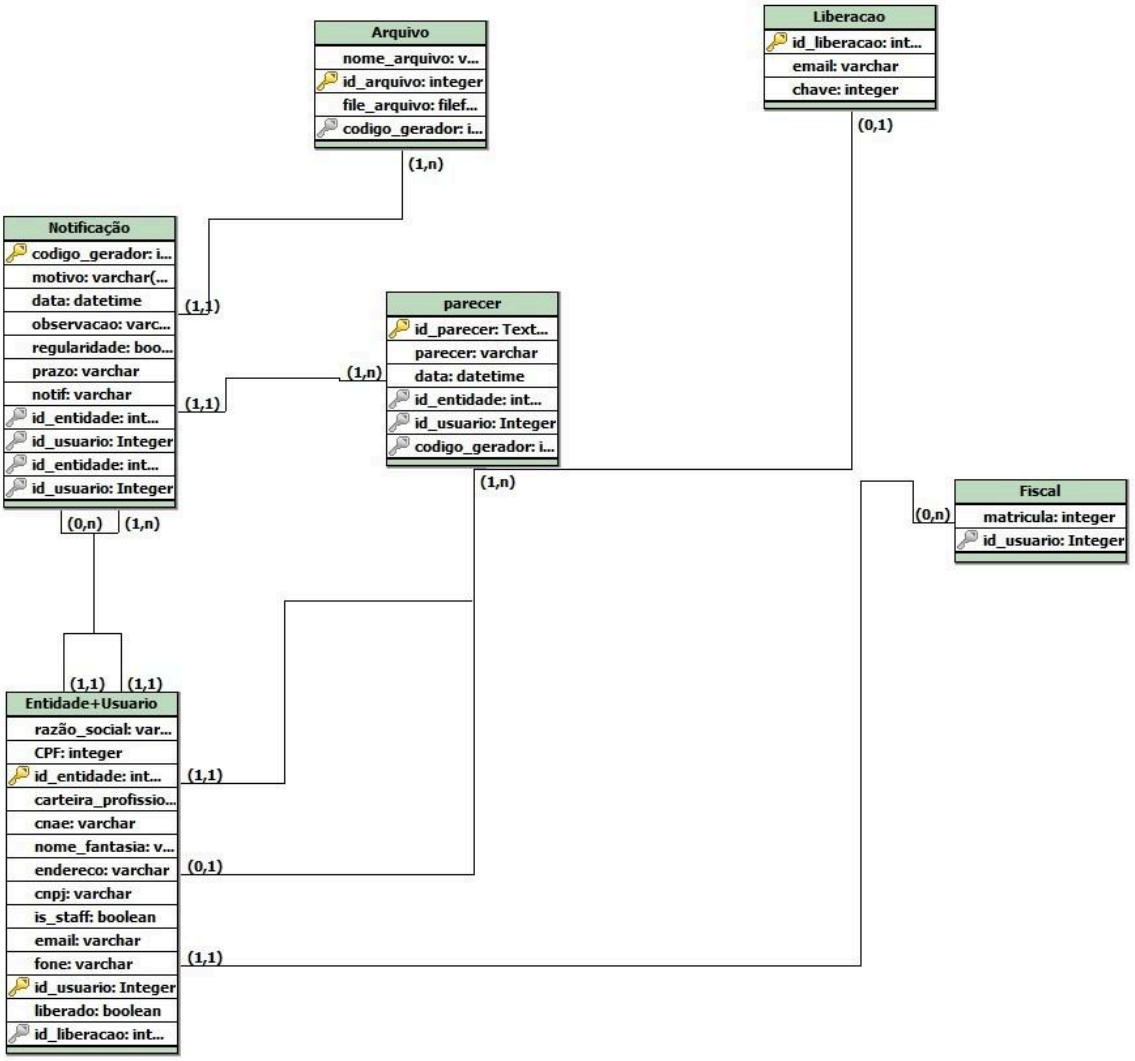
Figura 4 - Esquema visual banco de dados



Fonte: Do autor.

Na *figura 5* abaixo, se tem o esquema lógico, que é uma representação mais fiel ao pensamento de “tabelas” de um sistema de banco de dados, conseguimos visualizar de forma mais autêntica.

Figura 5 - Esquema lógico de banco de dados



Fonte: Do autor.

Portanto, é percebido que seis modelos são precisos para a construção do sistema, os modelos de usuário, liberação, notificação, parecer e arquivo, a entidade ‘fiscal’ na modelagem é uma especialização da generalização Usuário. Ou seja, é como se fosse um cargo especial dentro da faixa de usuário. A partir destes modelos é pensado a codificação e suas ligações.

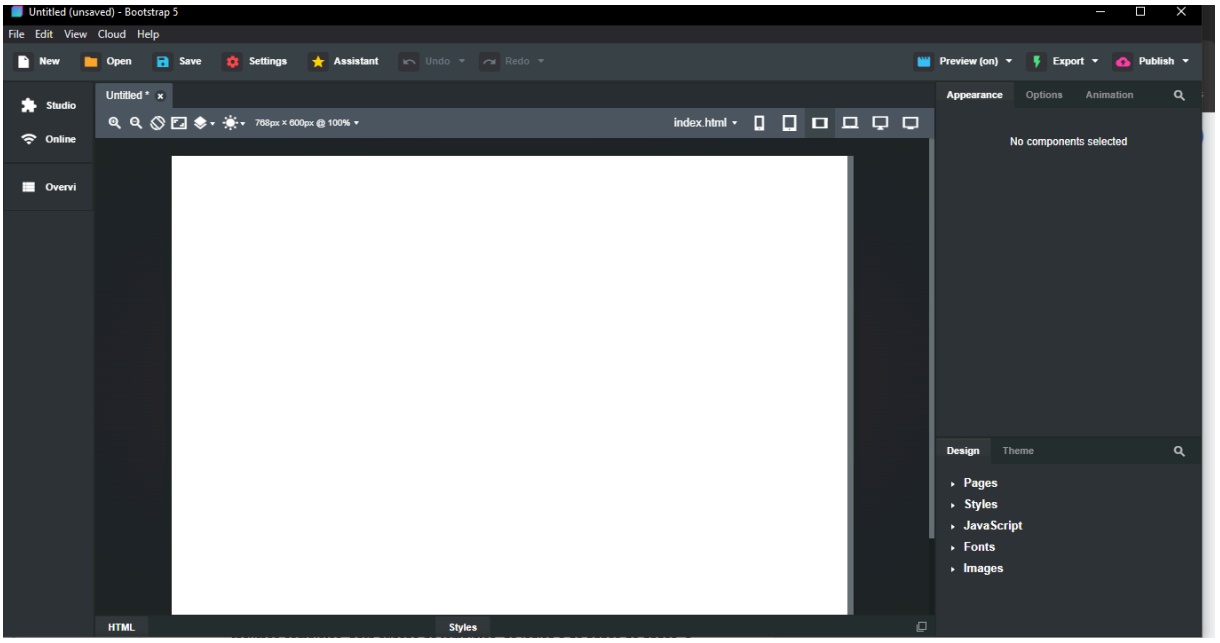
4.4 DESENVOLVIMENTO

Nesta etapa ocorre o desenvolvimento em si do sistema, onde será feito de maneira prática sua codificação e desenvolvimento de ambiente gráfico e outros. É aqui que de fato o sistema começa a ser funcional, e todo planejamento e estudo prévio se faz útil.

4.4.1 Desenvolvimento de Ambiente Gráfico

Para o desenvolvimento do ambiente gráfico foi utilizado o software bootstrap studio, bem como a linguagem de CSS para estilizar de forma mais específica. O bootstrap studio é um software para criação de templates no framework bootstrap de forma mais completa, utilizando de drag and drop, e outras configurações, retornando no final todos templates com suas respectivas codificações javascript, html e css, exemplificado na *figura 6 abaixo.*

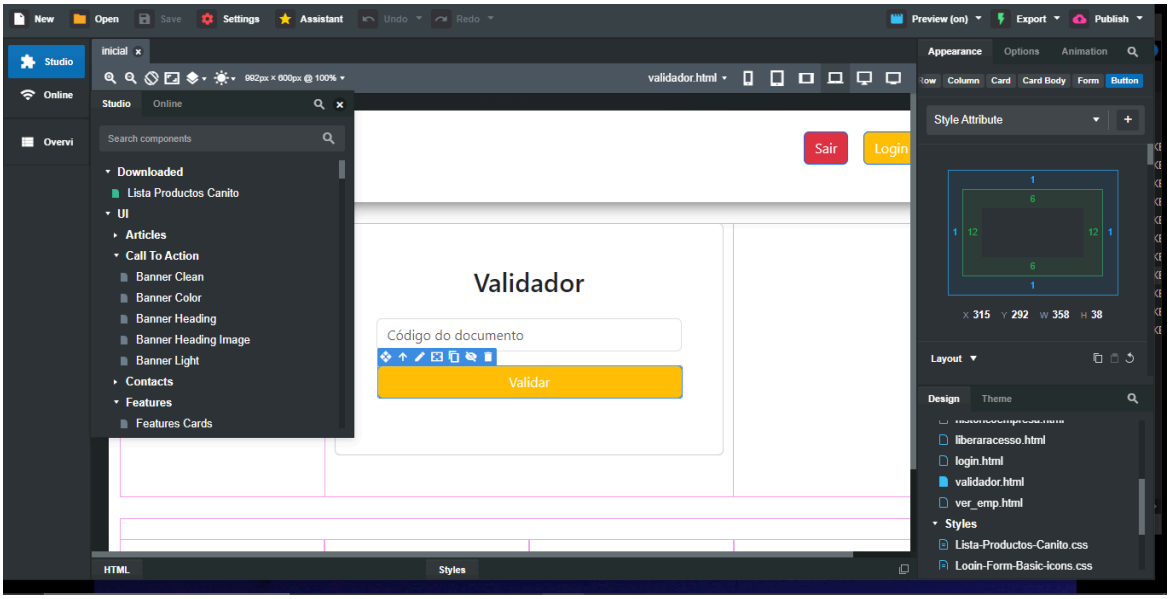
Figura 6 - Ambiente Bootstrap Studio



Fonte: Do autor.

Através desse software, é produzido todas as páginas do sistema, como um exemplo na *figura 7* abaixo, na qual se está produzindo a tela de validação.

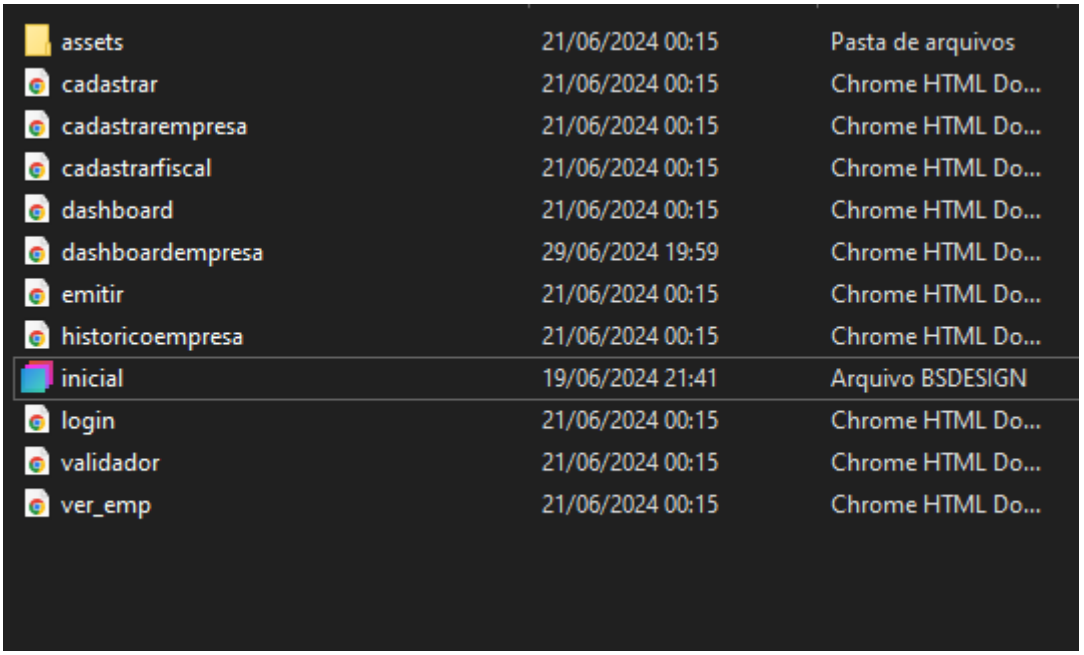
Figura 7 - Exemplo página Bootstrap Studio



Fonte: Do autor.

Ao final da diagramação do site, e sua exportação, o software faz download de todos os templates em ‘.html' e traz seus assets em conjunto como visto na *figura 8*. Os templates feitos no bootstrap ficam prontos para exportar diretamente para a pasta de templates do Django.

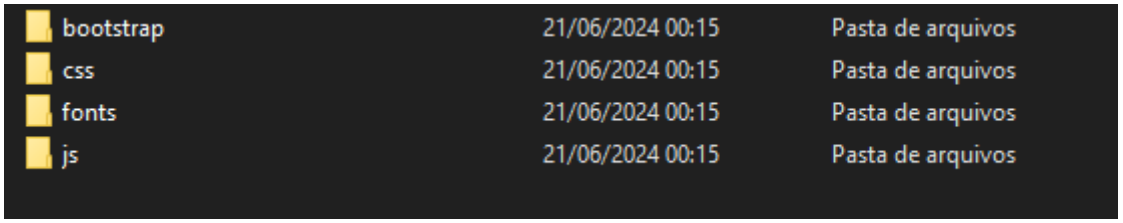
Figura 8 - Templates exportados em conjunto com assets



Fonte: Do autor.

E entrega junto a pasta ‘assets’ onde contém todos os códigos e arquivos secundários desta criação, como o javascript, css, fontes e códigos bootstrap, demonstrado na *figura 9*. Estes assets podem ser utilizados e aplicados diretamente na pasta static do framework.

Figura 9 - Assets gerados no Bootstrap Studio



Fonte: Do autor.

Por fim, como todo projeto de desenvolvimento, algumas alterações devem ser feitas de maneira mais direta e individual, conforme a necessidade. Para isso, neste projeto, foi criado um arquivo de css próprio para modificações de certos elementos de forma isolada. Como na primeira parte, na *figura 10*, a construção de códigos para classes de .form-container e form-notificacao, para modificar alguns detalhes de estilo em formulários.

Figura 10 - Primeira parte código CSS próprio

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f2f2f2;
}
.form-container {
    max-width: 600px;
    margin: 20px auto;
    padding: 6px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 20px rgba(216, 209, 209, 0.1);
}
.form-notificacao input,select{
    margin:5px;
    width:500px;
}
.form-container input,select, textarea {
    margin-bottom: 5px;
    width:580px;
    border-radius: 4px;;
}
.form-container input[type=checkbox]{
    width:50px;
}
```

Fonte: Do autor.

Bem como a alteração de formulário da entidade, demonstrado na *figura 11 e* 12, modificando algumas características individuais como o input e label, visualmente adicionando cores de fundo, alinhamento dos elementos bem como alterações nas bordas.

Figura 11 - Segunda parte código CSS próprio

```css
.form-container input[type=submit] {
    background-color: #0aa303;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}
.form-container input[type=submit]:hover {
    background-color: #0cad05;
}
.form-entidade {
    max-width: 600px;
    margin: 10px auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);
}
.form-entidade form {
    display: flex;
    flex-direction: column;
}
.form-entidade label {
    margin-bottom: 10px;
    font-weight: bold;
}
```

```css
.form-entidade input[type=text],
.form-entidade input[type=number],
.form-entidade textarea {
    width: 100%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
    font-size: 16px;
}
```

Fonte: Do autor.

Figura 12 - Terceira parte código CSS próprio

```
}
.form-entidade select {
    width: 100%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
    font-size: 16px;
    appearance: none;
    -webkit-appearance: none;
    -moz-appearance: none;
    background-image: url('data:image/svg+xml;utf8,<svg fill="%23333" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg"><path d="M7 10l5 5 5-5H7z"/></svg>')
    background-repeat: no-repeat;
    background-position-x: calc(100% - 12px);
    background-position-y: center;
}
.form-entidade input[type=checkbox] {
    margin-top: 5px;
}
```

Fonte: Do autor.

Sobre o formulário do usuário, com algumas mudanças e padronizações necessárias, para adequar alguns visuais parecidos a todos formulários do sistema, contida na imagem da *figura 13.*

Figura 13 - Quarta parte código CSS próprio

```
}
.form-usuario {
    max-width: 400px;
    margin: 10px auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);
}
.form-usuario form {
    display: flex;
    flex-direction: column;
}


.form-usuario label {
    margin-bottom: 10px;
    font-weight: bold;
}
.form-usuario input[type=text],
.form-usuario input[type=password] {
    width: 100%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
    font-size: 16px;
}
```

```css
.form-usuario input[type=submit] {
    background-color: #0aa303;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}
.form-usuario input[type=submit]:hover {
    background-color: #0cad05;
}
```

Fonte: Do autor.

Alterações na página de visualização de notificação, contando com padronizações, cores de fundo, e margin para visualmente ficar mais estável, como visto na *figura 14, 15 e 16.*

Figura 14 - Quinta parte código CSS próprio

```css
.page_visu_notif .card a.btn {
    margin-right: 10px;
}

.page_visu_notif .table {
    width: 100%;
    margin-top: 10px;
}

.page_visu_notif .table th,
.page_visu_notif .table td {
    padding: 8px;
    text-align: left;
}
```

Fonte: Do autor.

Figura 15 - Sexta parte código CSS próprio

```css
.page_visu_notif .table th {
    background-color: #f2f2f2;
}

.page_visu_notif .table-striped tbody tr:nth-of-type(odd) {
    background-color: #f9f9f9;
}

.page_visu_notif .table-striped tbody tr:hover {
    background-color: #e9ecef;
}

.page_visu_notif .table-striped tbody td {
    vertical-align: middle;
}
```

Fonte: Do autor.

Figura 16 - Sétima parte código CSS próprio

```css
.page_visu_notif .notification-details {
    background-color: #f5f5f5;
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    margin-bottom: 20px;
    text-align: left;
}

.page_visu_notif .notification-details p {
    margin-bottom: 10px;
    font-size: 16px;
    line-height: 1.6;
    color: #333;
}

.page_visu_notif .notification-details p strong {
    font-weight: bold;
    color: #555;
}

.FF01{
    font-weight: 600;
    text-decoration: none;
    color: #000;
    font-size: 20px;
}
```

Fonte: Do autor.

Depois de todas essas construções de templates e alterações individuais, o foco quase que integral do sistema foi na sua codificação que se tornou a parte mais

custosa, com uma busca de conhecimentos em várias fontes diferentes e com idiomas diferentes. Segue abaixo a seção sobre o desenvolvimento da codificação.

4.4.2 Desenvolvimento da Codificação

O framework usado neste projeto é o Django na versão 3.1.4, trazendo recursos completos, para criação de templates, de lógica e de banco de dados, é possível criar uma aplicação web completa em todos seus sentidos. Para esta aplicação foi utilizado em primeiro momento o banco de dados integrado, SQLite.

A configuração do ambiente de programação começa com o uso de um software de um ambiente de desenvolvimento, foi utilizado então o software Visual Code Studio para o desenvolvimento. Foi criado uma pasta chamada ‘Denotify’ no próprio computador e esta pasta foi aberta no Visual Code, utilizando da linha de comando no próprio programa, é preciso configurar um ambiente virtual antes de começar o sistema, também a linguagem python deve estar instalada na máquina.

Em primeiro lugar é necessário explicar que o Django funciona na lógica projeto e aplicações, é dentro das aplicações que toda estrutura de programação realmente funciona, utilizando em cada aplicação o padrão MTV.

O MTV é a sigla para model, template e view, que cada aplicação tem em sua estrutura, nos models são os moldes para a criação de entidades que refletirão no banco de dados, no template são aqueles que têm como função primordial a interface visual, e por fim os views, que são toda a programação que em si faz as funcionalidades.

Para configurar um ambiente, precisamos criar primeiro um ambiente virtual através do venv, no console precisa escrever ‘python -m venv venv’ onde criará um ambiente virtual chamado venv, e posterior deve se adentrar nessa virtualização com o comando ‘./venv/Scripts/activate.ps1’. Com o ambiente criado, é preciso instalar o django através do comando ‘pip install django == 3.1.4’, e dentro do framework é preciso agora a criação do projeto por meio do seguinte comando ‘ django-admin startproject denotify . ‘ sendo usado para este projeto o nome denotify, e criando uma aplicação através do comando ‘django-admin startapp appdenoti’. Com o projeto e a aplicação criada é preciso inicialmente eles se comunicarem.

Para fazer o projeto e aplicação se comunicar, se deve acessar o arquivo settings do denotify e aplicar em installed apps o nome appdenoti, posteriormente ir ao arquivo urls do projeto, e por como padrão através de ‘ path('', include('appdenoti.urls'))’ que garante que toda comunicação com o projeto irá também enxergar as as url da aplicação

Tudo configurado, na aplicação precisa criar um arquivo a mais chamado ‘urls.py’ em que ficará todas as rotas entre views e templates, bem como a nomeação dessa rota. Na aplicação além de urls.py, se cria a pasta templates, e se tem os arquivos models.py, views.py. Que serão abaixo explicados em partes.

Modelos é a forma que o framework Django representa os dados que devem ser armazenados no banco de dados, onde se dá o nome do campo, o tipo de valor, e atributos como tamanho, padrão e outras regras. Através dos modelos programados no arquivo models.py dentro do framework, é feito os seguintes processos, de migrations, que faz uma varredura sobre as modificações e criações de modelos, além de criar um documento com uma estrutura de dados para ser exportado para o banco de dados, por fim é utilizado o processo de migrate, exportando de fato.

Para este projeto, foram criados os seguintes modelos: modelusuario, entidade, notificação, parecer, arquivo, liberação. O django tem um padrão de modelo de usuário com campos padrões, o que nesse projeto se tornou inviável pois precisaríamos usar o email como padrão além de outras informações necessárias, para isso foi necessário refazer esse padrão primeiro através do BaseUserManager, como visto na *figura 17 e 18*.

Figura 17 - Primeira parte do código dos Models

```
class UsuarioManager(BaseUserManager):
    use_in_migrations = True

    def _create_user(self, email, password, **extra_fields):
        if not email:
            raise ValueError('O email é obrigatório')
        email = self.normalize_email(email)
        user = self.model(email=email, username=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self,email, password=None, **extra_fields):
        extra_fields.setdefault('is_superuser', False)
        return self._create_user(email, password, **extra_fields)
```

Fonte: Do autor.

Figura 18 - Segunda parte do código dos Models

```
def create_superuser(self, email, password, **extra_fields):
    extra_fields.setdefault('is_superuser', True)
    extra_fields.setdefault('is_staff', True)
    extra_fields.setdefault('is_active', True)

    if extra_fields.get('is_superuser') is not True:
        raise ValueError('Super usuário precisa ter is_superuser=True')
    if extra_fields.get('is_staff') is not True:
        raise ValueError('Super usuário precisa ter is_staff=True')

    return self._create_user(email, password, **extra_fields)
```

Fonte: Do autor.

Logo em seguida da modificação desse padrão, é feito o novo modelo de dados para o usuário, como visto na *figura 19,* definindo os próprios campos necessários para este projeto.

Figura 19 - Terceira parte do código dos Models

```
class ModelUsuario(AbstractUser):
    email = models.EmailField('E-mail', unique=True)
    fone = models.CharField('Telefone', max_length=15)
    is_staff = models.BooleanField('Membro de equipe', default=False)
    matricula = models.CharField('Matricula', max_length=15, blank=True, unique=True)
    liberado = models.BooleanField(default=False)


    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name', 'fone', 'matricula']

    def __str__(self):
        return self.email

    objects = UsuarioManager()
```

Fonte: Do autor.

Esta é a única modificação estrutural feita nos modelos para este projeto, todos os outros são feitos do zero sem qualquer tipo de modificação extra, abaixo segue todos os outros modelos criados.

Na *figura 20,* temos o modelo entidade que define as empresas e autônomos

que foram notificados, com informações que se fazem necessário, como a razão social, cnpj ou cpf, endereço e código profissional, também se fez uma ligação via uma chave estrangeira com um usuário que pode vir a existir, caso o notificado queira ver a notificação ou histórico.

Figura 20 - Quarta parte do código dos Models

```
class Entidade(models.Model):
    razao_social = models.CharField(max_length=100, default="")
    nome_fantasia = models.CharField(max_length=100, blank=True, default="")
    cnpj = models.IntegerField(blank=True, unique=True, null=True, default="")
    cpf = models.IntegerField(blank=True, unique=True, null=True, default="")
    endereco = models.CharField(max_length=100,default="")
    codigo_profissional = models.CharField(max_length=50, blank=True, default="")
    cnae = models.IntegerField(default="", null=True, blank=True,)
    usuario = models.OneToOneField(ModelUsuario, on_delete=models.SET_NULL,  blank=True, null=True)

    def __str__(self):
        return self.razao_social
```

Fonte: Do autor.

O próximo modelo, visto na *figura 21 e 22,* é o de notificação na qual irá criar uma entidade no banco de dados com os dados necessários para uma notificação como um código gerador que é criado de maneira aleatória e automática pelo sistema, sendo a chave primária deste modelo, em seguida tempos notif que representa o número da notificação, data, motivo, observação, regularidade, prazo, o fiscal via chave estrangeira que notificou e a entidade(empresa) que foi notificada.

Figura 21 - Quinta parte do código dos Models

```
class Notificacao(models.Model):
    def gerar_codigo_verificador():
        return random.randint(10000000, 99999999)

    codigo_verificador = models.PositiveIntegerField(primary_key=True, unique=True, default=gerar_codigo_verificador)
    notif = models.PositiveIntegerField(default=None,blank=True, null=True)
    data = models.DateTimeField(blank=True, null=True)

    ESCOLHA_MOTIVO = (
        ('CAD', 'Sem Cadastro'),
        ('ALV', 'Sem Alvará'),
        ('DEB', 'Em Debito'),
        ('SB', 'Sem Baixa'),
    )
```

Fonte: Do autor.

Figura 22 - Continuação da quinta parte do código dos Models

```
class Notificacao(models.Model):
    def gerar_codigo_verificador():
        return random.randint(10000000, 99999999)

    codigo_verificador = models.PositiveIntegerField(primary_key=True, unique=True, default=gerar_codigo_verificador)
    notif = models.PositiveIntegerField(default=None,blank=True, null=True)
    data = models.DateTimeField(blank=True, null=True)

    ESCOLHA_MOTIVO = (
        ('CAD', 'Sem Cadastro'),
        ('ALV', 'Sem Alvará'),
        ('DEB', 'Em Debito'),
        ('SB', 'Sem Baixa'),
    )
```

Fonte: Do autor.

O modelo ‘parecer’ na *figura 23,* tem como objetivo criar uma representação para manter armazenado os pareceres que os fiscais escrevem sobre alguma notificação, com os campos de parecer que são o texto em si, a data, o fiscal e a notificação.

Figura 23 - Sexta parte do código dos Models

```
class Parecer(models.Model):
    parecer = models.TextField(max_length=500)
    data_parecer = models.DateTimeField(default=None, null=True)
    fiscal = models.ForeignKey(ModelUsuario, on_delete=models.PROTECT)
    notificacao = models.ForeignKey(Notificacao, on_delete=models.PROTECT)

    def __str__(self):
        return self.notificacao

    def format_data(self):
        return self.data.strftime("%d/%m/%Y %H:%M")
```

Fonte: Do autor.

O próximo modelo, demonstrado na *figura 24,* é o arquivo que tem como objetivo permitir armazenar arquivos no sistema, sobre uma notificação, como a própria notificação assinada ou outro documento de interesse. Temos o campo de arquivo, nome do arquivo e via chave estrangeira a notificação que ele faz parte.

Figura 24 - Sétima parte do código dos Models

```
class Arquivo(models.Model):
    arquivo = models.FileField(upload_to='uploads/')
    nome_arquivo = models.CharField(max_length=50, unique=True)
    notificacao = models.ForeignKey(Notificacao, on_delete=models.PROTECT)

    def __str__(self):
        return self.nome_arquivo
```

Fonte: Do autor.

Por último temos o modelo de liberação, visto na *figura 25,* que servirá para armazenar os pedidos de liberação de um usuário contendo campo de email, de um usuário já cadastrado. A chave que poderá ser CNPJ ou CPF; e a data de criação que é automática do sistema.

Figura 25 - Oitava parte do código dos Models

```
class Liberacao(models.Model):
    email = models.EmailField('E-mail', null=False,blank=False, default="", primary_key=True)
    chave = models.IntegerField('Chave')
    data_criacao = models.DateTimeField(default=timezone.now, null=True, blank=True)

    def __str__(self):
        return self.email
```

Fonte: Do autor.

Após a construção dos modelos foi exportado os templates criados na etapa de desenvolvimento do ambiente gráfico, para isso primeiro se deve fazer uma alteração da pasta settings, como na *figura 26*, para configurar a rota da pasta static, onde irá todo tipo de conteúdo externo que modifique os templates de alguma maneira, como códigos css, javascript, entre outros.

Figura 26 - Configurações extras das rotas de pasta de mídia e Static

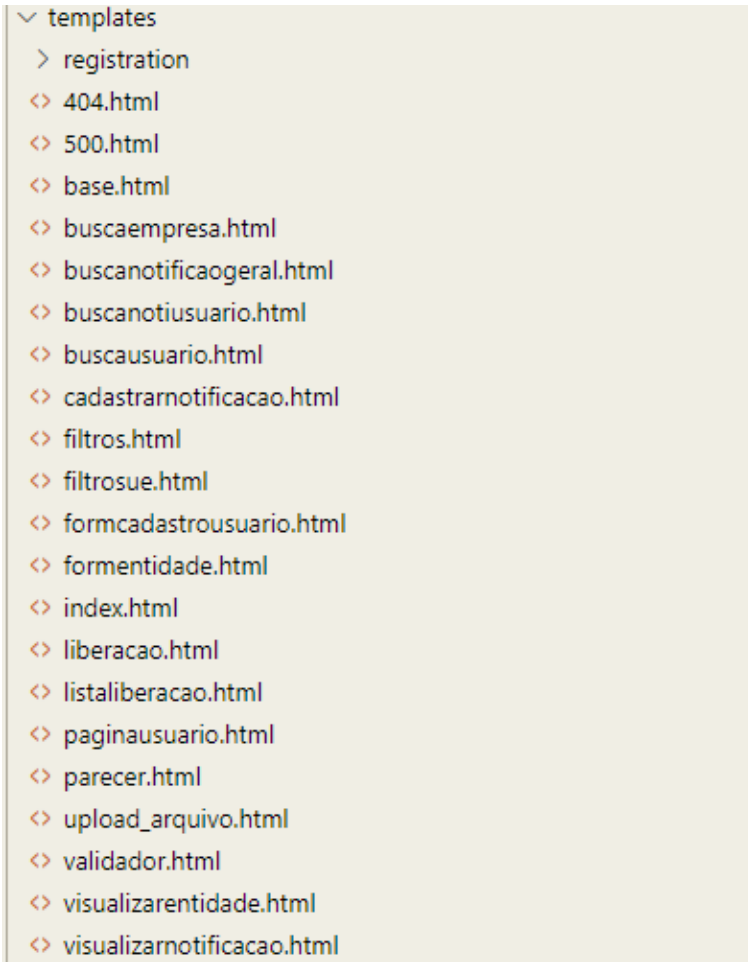
```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static'),
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Fonte: Do autor.

Em cada template é preciso configurar os chamados de códigos externos, para acessar a página static, sendo assim, quando há um chamado de um código css, como exemplo *<link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">* se passa para a seguinte forma *<link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap.min.css' %}">*, esta modificação deve ser feita em todos os templates, porém existe uma maneira no Django que ao se utilizar o mesmo design para todas as páginas, mudando apenas o conteúdo é possível criar um template base, ou seja, um template que pode ser estendido para todos os outros templates, apenas modificando o conteúdo, e mantendo todos os atributos necessários modificados.

Neste projeto foi utilizado um template base.html, em que todos os códigos css e javascript foram configurados para acessar a pasta static, e logo após apenas foi estendida a todos outros templates. No final do projeto, a pasta de templates ficou na forma mostrada na figura 27.

Figura 27- Pasta templates e todas seus elementos



Fonte: Do autor.

As URLs são uma parte essencial do framework, onde se cria a rota de ligação a uma view, é nela também que indicamos qual tipo de ‘nomeação’ será feita para a url do navegador, que ao ser digitada levará para aquela rota em específico, esta parte é feita concomitantemente com a programação das views. Abaixo na *figura 28 e 29,* segue como as urls ficaram no projeto.

Figura 28 - Rotas e Url's do sistema.

```
from django.urls import path, include
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('cadastrarempresa', views.forment, name='cadastrarempresa'),
    path('criaentidade', views.criaentidade, name='criarentidade'),
    path('buscaempresa', views.buscaempresa, name='buscaempresa'),
    path('editarentidade/<int:pk>/', views.editarentidade, name='editarentidade'),
    path('alterarentidade/<int:pk>/', views.alterarentidade, name='alterarentidade'),
    path('deletarentidade/<int:pk>/', views.deletarentidade, name='deletarentidade'),
    path('regularizar/<int:pk>/', views.regularizar, name='regularizar'),
    path('visualizarentidade/<int:pk>/', views.visualizarentidade, name='visualizarentidade'),
    path('criarusuarioentidade', views.criaruser),
    path('formcadastro', views.usuariocad, name='cadastrarent'),
    path("accounts/", include("django.contrib.auth.urls"), name='loginoriginal'),
    path("login", views.loginredirect),
    path("notificar", views.CriarNotificacao, name='notificar'),
    path("salvarnotificacao", views.salvarnotificacao ),
    path('parecer', views.CriarParecer),
    path('salvarparecer', views.salvarparecer),
    path('buscanotificacao', views.buscanotificacao, name='buscanotificacao'),
    path('buscausuario', views.buscausuario, name='buscausuario'),
    path('visualizarnotificacao/<int:pk>/', views.visualizarnotificacao, name='visualizarnotificacao'),
    path('editarnotificacao/<int:pk>/', views.editarnotificacao, name='editarnotificacao'),
    path('alterarnotificacao/<int:pk>/', views.alterarnotificacao, name='alterarnotificacao'),
    path('generate_pdf/<int:codigo_verificador>/', views.generate_pdf, name='generate_pdf'),
    path('upload_arquivo/', views.criaarquivo, name='upload_arquivo'),
    path('download_arquivo/<int:arquivo_id>/', views.download_arquivo, name='download_arquivo'),
```

Fonte: Do autor.

Figura 29 - Segunda parte de rotas e Url's do sistema.

```
    path('validador/', views.validador),
    path('historico',views.historicousuario, name='historico'),
    path('paginausuario/<int:id>/', views.paginausuario, name='paginausuario'),
    path('filtros/', views.filtros, name='filtros'),
    path('filtrousuario/', views.filtrosue, name='filtrosusuario'),
    path('liberacao', views.liberacao, name='liberacao'),
    path('listaliberacao', views.listaliberacao, name='listaliberacao'),
    path('listaliberacao/<str:email>/', views.filtroliberacao, name='filtroliberacao'),
```

Fonte: Do autor.

Uma das partes importantes é a construção de formulários, que define quais campos um formulário irá ter e alguns outros comportamentos que pode vir a ter. Um formulário pode ser criado através da linguagem de marcação HTML de forma tradicional em qualquer template apenas tendo que conectar aos campos do modelo, ou a forma mais fácil é a criação de formulários diretamente do forms, na qual toda a estrutura é facilmente "exportada" para qualquer modelo que seja

chamado, com todas conexões necessárias. Começamos importando todas bibliotecas necessárias para o sistema como vista na *figura 30.*

Figura 30 - Importação de bibliotecas

```
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from django import forms
from django.contrib import messages
from django.forms import ModelForm, DateTimeInput
from .models import Entidade, Notificacao, Parecer, Arquivo, Liberacao
from .models import ModelUsuario
```

Fonte: Do autor.

Logo após começamos a construção dos formulários, o primeiro formulário criado é o do cadastro de usuário, utilizamos como base o modelo criado em models.py 'ModelUsuario', e definimos quais campos daquele modelo será usado em 'fields', e por fim usamos uma função para salvar a instância no banco de dados, pode-se ver o resultado na *figura 31.*

Figura 31 - Formulário de Criação de Model Usuário.

```
class ModelUsuarioCreateForm(UserCreationForm):
    class Meta:
        model = ModelUsuario
        fields = ['username', 'first_name','last_name','fone']
        labels = {'username': 'Username/E-mail'}

    def save(self, commit=True):
        user = super().save(commit=False)
        user.set_password(self.cleaned_data["password1"])
        user.email = self.cleaned_data['username']

        if commit:
            user.save()

        return user
```

Fonte: Do autor.

O formulário de cadastro de usuário vem acompanhado de um formulário de mudança de dados, como o nome, sobrenome, matrícula e telefone. Utilizando do mesmo modelo base, mas usando como parâmetro o UserChangeForm, como visto na *figura 32.*

Figura 32 - Formulário de Alteração de Model Usuário.

```
class ModelUsuarioChangeForm(UserChangeForm):
    class Meta:
        model = ModelUsuario
        fields = ('first_name', 'last_name','matricula', 'fone')
```

Fonte: Do autor.

Para o um campo de busca foi criado um formulário, como na *figura 33,* que o objetivo é trazer uma padronização de campo para todas as partes que precisam de busca no sistema.

Figura 33 - Formulário de setor do campo de busca.

```
class buscafiltro(forms.Form):
    nome = forms.CharField(required=False)
```

Fonte: Do autor.

O formulário para cadastro de empresa ou autônomo demonstrado na *figura 34,* busca no modelo de Entidade, utilizando seus campos padrões, exceto o de 'usuário', pois este último será definido em um campo de liberação via programação, quando o usuário que pede acesso ser liberado, ele será linkado a empresa automaticamente.

Figura 34 - Formulário do Modelo Entidade

```
class EntidadeForm(ModelForm):
    class Meta:
        model = Entidade
        fields = ['razao_social', 'nome_fantasia', 'cnpj', 'cpf', 'endereco', 'codigo_profissional', 'cnae']
        exclude = ['usuario']
```

Fonte: Do autor.

A notificação conta com um formulário, como na *figura 35,* que usa o modelo de notificação como base e seus campos, além disso contém uma atributo widget,

que serve para passar classes(para uso de CSS) e outras informações, que garantem um layout mais padronizado.

Figura 35 - Formulário do Modelo Notificação

```
class NotificacaoForm(forms.ModelForm):
    class Meta:
        model = Notificacao
        fields = [
            'codigo_verificador',
            'notif',
            'data',
            'motivo',
            'observacao',
            'regularidade',
            'prazo',
            'fiscal',
            'entidade'
        ]
        widgets = {
            'data': forms.DateTimeInput(attrs={'class': 'form-control', 'placeholder': 'Data', 'type':'datetime-local'}, format='%Y-%m-%dT%H:%M')
            'codigo_verificador': forms.NumberInput(attrs={'class': 'form-control'}),
            'notif': forms.NumberInput(attrs={'class': 'form-control', 'placeholder': 'Nº Notificação'}),
            'motivo': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Motivo', 'type':''}),
            'observacao': forms.Textarea(attrs={'class': 'form-control', 'placeholder': 'Observação'}),
            'regularidade': forms.CheckboxInput(attrs={'class': 'form-check-input formentidade01'}),
            'prazo': forms.NumberInput(attrs={'class': 'form-control', 'placeholder': 'Prazo em dias'}),
            'fiscal': forms.Select(attrs={'class': 'form-control'}),
            'entidade': forms.Select(attrs={'class': 'form-control'}),
        }

    input_formats = {
        'data': ['%Y-%m-%dT%H:%M']
    }
```

Fonte: Do autor.

Os formulários de parecer e arquivo, respectivamente na *figura 36 e 37,* seguem a mesma lógica que os outros formulários, cada um utilizando seus respectivos modelos como base, atribuindo os campos necessários para estes formulários e passando atributos necessários para via widget.

Figura 36 - Formulário do Modelo Parecer

```
class ParecerForm(forms.ModelForm):
    class Meta:
        model = Parecer
        fields = ['parecer', 'data_parecer', 'fiscal', 'notificacao']
        widgets = {
            'parecer': forms.Textarea(attrs={'class': 'form-control', 'placeholder': 'Digite o parecer'}),
            'data_parecer': forms.DateTimeInput(attrs={'class': 'form-control', 'placeholder': 'Data do Parecer','type': 'datetime-local'}, format='%Y-%m-%dT%H:%M'),
            'fiscal': forms.Select(attrs={'class': 'form-control'}),
            'notificacao': forms.Select(attrs={'class': 'form-control'}),
        }

    input_formats = {
        'data_parecer': ['%Y-%m-%dT%H:%M']
    }
```

Fonte: Do autor.

Figura 37- Formulário do Modelo Arquivo

```
class ArquivoForm(forms.ModelForm):
    class Meta:
        model = Arquivo
        fields = ['arquivo', 'nome_arquivo', 'notificacao']
        widgets = {
            'arquivo': forms.FileInput(attrs={'class': 'form-control', 'placeholder': 'Selecione o arquivo'}),
            'nome_arquivo': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Nome do Arquivo'}),
            'notificacao': forms.Select(attrs={'class': 'form-control'}),
        }
```

Fonte: Do autor.

Toda notificação tem um código verificador, dentro dos requisitos funcionais do sistema se tem uma funcionalidade de validação de documento, para isso foi criado um formulário para a pesquisa deste código, visto na *figura 38.*

Figura 38 - Formulário do Código Verificador

```
class CodigoVerificadorForm(forms.Form):
    codigo_verificador = forms.CharField(max_length=8, label='Código do Documento')
```

Fonte: Do autor.

Por fim o último temos o formulário de liberação, como na *figura 39,* que utiliza também seu modelo como base, e seus campos além de atributos passados pelo widgets. Neste formulário tem uma função diferente, pois o campo chave ele pode receber informação de cnpj ou cpf, esta informação será analisada pelo

tamanho dos dígitos, que atualmente é padrão, nesta função é só pode ser pedido liberação se o cnpj ou cpf existir no banco de dados de alguma entidade, se não houver notificação para a empresa logo não há cadastro com estas informações, se houver este cadastro e já tiver algum usuário responsável, também é impedido de pedir a liberação.

Figura 39 - Formulário do Modelo Liberação

```python
class LiberacaoForm(forms.ModelForm):
    class Meta:
        model = Liberacao
        fields = ['email', 'chave']
        widgets = {
            'email': forms.EmailInput(attrs={'class': 'form-control', 'placeholder': 'Digite o email'}),
            'chave': forms.NumberInput(attrs={'class': 'form-control', 'placeholder': ' CNPJ ou CPF APENAS NUMEROS'}),
        }

    def clean(self):
        cleaned_data = super().clean()
        chave = cleaned_data.get('chave')

        if chave:
            if len(str(chave)) == 11:
                cpf = chave
            elif len(str(chave)) == 14:
                cnpj = chave
            else:
                raise forms.ValidationError("A chave deve ter 11 dígitos (CPF) ou 14 dígitos (CNPJ).")

            if 'cnpj' in locals():
                entidade_existente = Entidade.objects.filter(cnpj=cnpj).exists()
                if not entidade_existente:
                    raise forms.ValidationError("Não existe uma Empresa notificada nesse CNPJ.")
                entidade_usuario = Entidade.objects.filter(cnpj=cnpj, usuario__isnull=False).exists()
                if entidade_usuario:
                    raise forms.ValidationError("Este CNPJ já está associado a um usuário.")

            if 'cpf' in locals():
                entidade_existente = Entidade.objects.filter(cpf=cpf).exists()
                if not entidade_existente:
                    raise forms.ValidationError("Não existe notificação para este CPF.")
                entidade_usuario = Entidade.objects.filter(cpf=cpf, usuario__isnull=False).exists()
                if entidade_usuario:
                    raise forms.ValidationError("Este CPF já está associado a um usuário.")

        return cleaned_data

    def save(self, commit=True):
        # Salva a instância do formulário no banco de dados
        instance = super().save(commit=False)


        if commit:
            instance.save()
        return instance
```

Fonte: Do autor.

Visto as partes dos formulários, a próxima etapa são as views que são a parte que define toda a programação do sistema e é a mais extensa, são elas que as rotas da urls chamam e é nela que definimos os templates que serão usados. Para isso

começamos importando todas bibliotecas que precisamos, além dos modelos e formulários, como na *figura 40.*

Figura 40 - Bibliotecas importadas no arquivo Views

```
from django.shortcuts import render, redirect
from django.views import View
from django.http import JsonResponse, HttpResponse
from django.db.models import Q
from django.conf import settings
from django.utils import timezone
from django.contrib.auth.decorators import login_required, user_passes_test
from django.utils.http import urlencode
from django.contrib import messages
from django.utils.translation import gettext_lazy as _
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
from django.contrib.auth import login, authenticate, logout
from django.shortcuts import get_object_or_404

from datetime import timedelta
import re, os, io

from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from PyPDF2 import PdfReader, PdfWriter

from .models import Entidade, Notificacao, Parecer, Arquivo, ModelUsuario, Liberacao
from .forms import EntidadeForm, NotificacaoForm, ParecerForm, buscafiltro, ArquivoForm, ModelUsuarioCreateForm, LiberacaoForm
```

Fonte: Do autor.

A primeira view criada é a do index, visto na *figura 41*, que representa a página inicial de menu ‘dashboard’. O código tem como objetivo mostrar principalmente os dados dos filtros, e também para definir qual o template a ser usado. É definida duas variáveis, o usuario que recebe um request.user, que no framework retorna o usuário ativo no momento, e a próxima variável é o prazo limite, que recebe a data que uma notificação tem que estar para estar fora do prazo, ou seja, o timezone.now() traz a data atual e subtrai menos 30 dias pois é o padrão a legislação municipal.

Após a definição destas variáveis, é feito um controle de fluxo a partir de um primeiro controle se o usuário está autenticado, se não estiver ele retorna o index.html sem dados nenhum, ou seja, o index se torna uma página para uma pessoa que não tem acesso. Se a pessoa tem acesso ao sistema, é feito outro controle de fluxo, se usuário é staff ou não.

Se o usuário for staff ele retorna quatro tipo de filtros, as notificações irregulares, regulares, total e fora do prazo, com seus respectivos cálculos no código. Se o usuário não for staff ele retorna as notificações que são ligadas a entidade(empresa) do usuário, bem como o prazo, e as abertas.

Figura 41 - Código do view do Index

```
def index(request):

    usuario = request.user
    prazo_limite = timezone.now() - timedelta(days=30)
    if usuario.is_authenticated:
        if usuario.is_staff:
            count_irregulares = Notificacao.objects.filter(regularidade=False).count()
            count_regulares = Notificacao.objects.filter(regularidade=True).count()
            count_total = Notificacao.objects.count()
            count_fora_do_prazo = Notificacao.objects.filter(Q(data__lte=prazo_limite) & Q(regularidade=False)).count()

            context = {
            'irregulares':count_irregulares,
            'regulares':count_regulares,
            'total':count_total,
            'data':count_fora_do_prazo,

            }
            return render(request, 'index.html', context)
        else:
            entidade = Entidade.objects.get(usuario__user=usuario)
            notificacoes = Notificacao.objects.filter(entidade=entidade)
            c_not_usuario_ab = notificacoes.filter(regularidade=False).count()
            c_usuario_limit = notificacoes.filter(Q(data__lte=prazo_limite) & Q(regularidade=False)).count()
            context = {
            'total':notificacoes.count(),
            'abertos':c_not_usuario_ab,
            'data_us':c_usuario_limit,
            }
            return render(request, 'index.html', context)

    else:
        return render(request, 'index.html')
```

Fonte: Do autor.

Depois do index, começamos a desenvolver por parte, a primeira parte é os códigos relacionados à entidade, ou seja, empresas e de autônomos, como na *figura 42*. A primeira parte do código é o form entidade,ou seja, a página de cadastro, a partir de agora vários códigos começarão a ter em conjunto os decoradores @login_required para obrigar o uso de um usuário logado, e o @user_passes_test para atribuir alguma característica necessária ao usuário, neste caso só pode entrar no formulário de cadastro o usuário logado e staff. Será aberto um dicionário vazio, que posteriormente será definido uma chave para esse dicionário, neste caso [‘form’] e receberá uma instância do formulário EntidadeForm(), retornando uma renderização do template formentidade.html e passando a data como contexto.

Figura 42 - Código do view para criação de entidade.

```
@login_required
@user_passes_test(is_staff)
def forment(request):
    data = {}
    data['form'] = EntidadeForm()
    return render(request, 'formentidade.html', data)
```

Fonte: Do autor.

Um exemplo, na *figura 43*, de como pode ser chamado este tipo de contexto em forma de formulário abaixo, quando entramos com um html ‘form’ precisamos definir seu action, a maneira mais fácil é definir uma view, que receberá os dados do formulário e irá fazer todo tipo de uso necessário com ele.

Figura 43 - Exemplo de como chamar em um template um formulário.

```
<form name="form" id="form" action="{% url 'criaentidade' %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" class="btn btn-success" value="Salvar">
</form>
```

Fonte: Do autor.

O view cria entidade, na *figura 44,* tem como objetivo apenas validar e salvar os dados do formulário no banco de dados, recebe o action do formulário e depois redireciona para o dashboard.

Figura 44 - Código view para salvamento de dados da Entidade

```
@login_required
@user_passes_test(is_staff)
def criaentidade(request):
    form= EntidadeForm(request.POST or None)
    if form.is_valid():
        form.save()
        return redirect('index')
```

Fonte: Do autor.

Abaixo segue-se, na *figura 45*, segue-se com duas views diferentes, na view editar entidade, recebe como parâmetro também uma chave primária, abrimos com um dicionário vazio e posteriormente nós adicionamos duas chaves diferentes, a chave ‘db’ tem como objetivo selecionar um dado no banco de dados através da

chave primária correspondente, e a chave ‘form’ cria uma instance do dados conseguido pelo ‘db’, nesse sentido o django entende que o objetivo é alterar os dados, demonstrando os dados no formulário para facilmente ser alterado. Na view altera entidade, ela será utilizado em formulário de edição dos dados, assim salvando no banco de dados.

Figura 45 - Código view para edição dos dados da Entidade

```
@login_required
@user_passes_test(is_staff)
def editarentidade(request, pk):
    data = {}
    data['db'] = Entidade.objects.get(pk=pk)
    data['form'] = EntidadeForm(instance = data['db'])
    return render(request, 'formentidade.html', data)

@login_required
@user_passes_test(is_staff)
def alterarentidade(request, pk):
    data = {}
    data['db'] = Entidade.objects.get(pk=pk)
    form = EntidadeForm(request.POST or None, instance=data['db'])
    if form.is_valid():
        form.save()
        return redirect('buscaempresa')
    else:
        error_message = "FORMULARIO INVALIDO."
        fm = EntidadeForm(instance = data['db'])
        data={
            'error_message':error_message,
            'form':fm,
        }
        return render(request, 'formentidade.html', data)
```

Fonte: Do autor.

As próximas views, na *figura 46,* são deletar entidade e o visualizar entidade, respectivamente, o primeiro recebe uma chave primária via parâmetro e utiliza ele no chaveamento para achar o dado correspondente no banco de dados, e logo depois utilizamos a função delete(), que deleta imediatamente. No segundo também se tem o parâmetro de chave primária que seleciona o objeto correspondente, e também seleciona deste objeto todas notificações emitida para este, sendo assim, envia para o template ‘visualizarentidade.html’.

Figura 46 - Código view de deleção e visualização da entidade.

```
@login_required
@user_passes_test(is_staff)
def deletarentidade(request, pk):
    data = {}
    data['db'] = Entidade.objects.get(pk=pk)
    data['db'].delete()
    return redirect('buscaempresa')

@login_required
@user_passes_test(is_staff)
def visualizarentidade(request, pk):
    data = {}
    data['db'] = Entidade.objects.get(pk=pk)
    data['pn'] = Notificacao.objects.all()
    data['pn'] = data['pn'].filter(entidade__id__icontains= pk)
    return  render(request, 'visualizarentidade.html', data)
```

Fonte: Do autor.

A view abaixo, na *figura 47 e 48*, é uma das quais se tem em quase todos os grupos, é a view de busca, neste caso sendo aplicado a busca de entidades, mas também aplicado em usuários, notificações, liberação, será mostrado apenas esta aplicação de busca para não ficar repetitivo, pois o código é praticamente igual modificando apenas quais tipos de matchs necessários para a pesquisa e qual modelo será selecionado, além do algoritmo de busca em si, é passado também o sistema de paginação, toda esse código é enviado para o template ‘buscaempresa.html’.

O código seleciona na variável ‘all’ qual modelo será pesquisado, ou seja nesse caso selecione todos os objetos da Entidade, e assim usado lá no final do código para gerar a paginação e também para listar todos os objetos na página. Em search_queries se busca criar filtros de pesquisa para facilitar a busca, sendo assim utilizamos por exemplos filtros como ‘empresa’, ‘cnpj’ e ‘cpf’, podendo ser utilizado na pesquisa da seguinte forma, como exemplo, ‘empresa: ufsc’, assim será pesquisado apenas esse objeto na lista.

Figura 47 - Código view para buscar empresa.

```
@login_required
@user_passes_test(is_staff)
def buscaempresa(request):
    dados = {}
    form = buscafiltro(request.GET or None)
    all = Entidade.objects.all()

    if form.is_valid():
        queries = []
        search_input = form.cleaned_data.get('nome')

        if search_input:
            entidade_match = re.search(r'empresa:\s*([^\s]+)', search_input, re.IGNORECASE)
            cnpj_match = re.search(r'cnpj:\s*([^\s]+)', search_input, re.IGNORECASE)
            cpf_match =  re.search(r'cpf:\s*([^\s]+)', search_input, re.IGNORECASE)

            if entidade_match:
                empresa = entidade_match.group(1)
                queries.append(Q(razao_social__icontains=empresa))

            if cnpj_match:
                cnpj = cnpj_match.group(1)
                queries.append(Q(cnpj__icontains=cnpj))

            if cpf_match:
                cpf = cpf_match.group(1)
                queries.append(Q(cpf__icontains=cpf))

            else:
                queries.append(Q(razao_social__icontains=form.cleaned_data['nome']))


        filtrado = next((all.filter(query) for query in queries if all.filter(query).exists()), all)
```

Fonte: Do autor.

Figura 48 - Continuação código view para buscar empresa.

```
    all = filtrado

all =all.order_by('id')
paginator = Paginator(all, 5)
pages = request.GET.get('page')
dados['db'] = paginator.get_page(pages)

if request.is_ajax():
    data = [
        {
            'id': dbs.id,
            'razao_social': dbs.razao_social,
        }
        for dbs in dados['db']
    ]
    return JsonResponse(data, safe=False)

dados['form'] = form
return render(request, 'buscaempresa.html', dados)
```

Fonte: Do autor.

Saímos agora do grupo de entidade e entramos no usuário, visto na *figura 49*, são três views nesse grupo, usuariocad, criaruser e buscausuario. O usuariocad começa com um dicionário vazio e passa via chaveamento o formulário de cadastro, para o template 'formcadastrousuario.html'. A view criaruser recebe depois de submeter o cadastro, validando as informações e assim salvando no banco de dados. Levando em consideração os views de cadastro e validação nas views anteriores, o código deste tipo é igual e se mantém em outros grupos, sendo assim para não se tornar repetitivo não será mostrado, porém as coisas que modificam são o formulário a ser usado no chaveamento 'form' e qual template será usado para receber este contexto.

Figura 49 - Código view para criação de usuário.

```
def usuariocad(request):
    data = {}
    data['form'] = ModelUsuarioCreateForm()
    return render(request, 'formcadastrousuario.html', data)


def criaruser(request):
    form= ModelUsuarioCreateForm(request.POST or None)
    if form.is_valid():
        form.save(commit=True)
        return redirect('index')
    else:
        return render(request, 'formcadastrousuario.html',{'form':form})
```

Fonte: Do autor.

A view visualizar notificação traz três chaveamentos importantes, um que pega a notificação específica, e outros dois com filtros, os filtros trazem os pareceres e os arquivos conectados a uma notificação em específica passando para o template 'visualizarnotificacao.html'. Regularizar traz a função quando um botão de regularização ser pressionado, ele irá selecionar uma notificação específica e transformar sua regularidade em True, isso significa que a notificação foi resolvida e está sem pendências, as duas views estão demonstradas na *figura 50*.

Figura 50 - Código view para visualizar notificações e regularizar.

```
@login_required
def visualizarnotificacao(request, pk):
    data = {}
    data['db'] = Notificacao.objects.get(pk=pk)
    data['pn'] = Parecer.objects.all()
    data['pn'] = data['pn'].filter(notificacao__codigo_verificador__icontains= pk)
    data['ar'] = Arquivo.objects.all()
    data['ar'] = data['ar'].filter(notificacao__codigo_verificador__icontains= pk)
    return  render(request, 'visualizarnotificacao.html', data)

@login_required
@user_passes_test(is_staff)
def regularizar(request, pk):
    notificacao = Notificacao.objects.get(pk=pk)
    notificacao.regularidade = True
    notificacao.save()
    return redirect('index')
```

Fonte: Do autor.

A última view desse grupo, visto na *figura 51*, é um código que também é reutilizado em outros grupos, para o mesmo tipo de ação, modificando apenas o modelo que será escolhido. No primeiro via a chave ‘db’ é utilizado para selecionar uma notificação em específico, e na chave ‘form’ é passado um formulário via instance, que o Django compreende que é para fazer edição, por isso quando o template é carregado com essa view, os dados da notificação escolhida estão abertas para edição. Posterior ao submeter a edição é enviado para o alterar, que faz a validação do formulário e posteriormente salva no banco de dados.

Figura 51 - Código view para edição e alteração da notificação.

```
@login_required
@user_passes_test(is_staff)
def editarnotificacao(request, pk):
    data = {}
    data['db'] = Notificacao.objects.get(pk=pk)
    data['form'] = NotificacaoForm(instance = data['db'])
    return render(request, 'cadastrarnotificacao.html', data)

@login_required
@user_passes_test(is_staff)
def alterarnotificacao(request, pk):
    data = {}
    data['db'] = Notificacao.objects.get(pk=pk)
    form = NotificacaoForm(request.POST or None, instance=data['db'])
    if form.is_valid():
        form.save()
        return redirect('index')
```

Fonte: Do autor.

Em seguida na *figura 52*, é visto o código do parecer, que é um código reutilizável já explicado neste projeto.

Figura 52 - Código view para criação de Parecer e salvamento.

```
@login_required
@user_passes_test(is_staff)
def CriarParecer(request):
    data = {}
    data['f'] = ParecerForm()
    return render(request, 'parecer.html', data)

@login_required
@user_passes_test(is_staff)
def salvarparecer(request):
    form= ParecerForm(request.POST or None)
    if form.is_valid():
        form.save()
        return redirect('index')
```

Fonte: Do autor.

O código na *figura 53,* refere-se a emissão de uma notificação documental, ou seja, um pdf imprimível com a notificação em branco. O código recebe como parâmetro o código gerador, e começa com uma variável ‘notificacao’ recebendo um objeto do modelo ‘Notificacao’ que tenha um código gerador em específico, anteriormente é criado um modelo padrão de documento, e para isso precisamos indicar seu caminho, na variável ‘pdf_template_path’ contém o caminho necessário via o static para o modelo usado, com o uso de bibliotecas se lê o template via PdfReader e também se atribui uma variável a função PdfWriter.

Na variável ‘page’ é selecionado a primeira página e posteriormente criamos um buffer temporário para esse novo pdf selecionado, o código verificador é uma parte importante no processo para que o notificado no futuro tenha acesso a validação e outras funções, sendo assim o modelo não tem um código verificador padrão pois cada notificação tem um código diferente.

Sendo assim é necessário utilizando da função drawString colocamos esse código em uma localização exata, com uma fonte ‘Helvetica’ e de tamanho 12. Posteriormente o buffer é movido para o início para ser lido. Em seguida, o PDFReader processa essa nova página, e essa página temporária é mesclada com um template previamente carregado, representado pela variável page. Com o uso da

função merge_page, a nova página é sobreposta ao template, gerando um documento final com as informações necessárias.

Um PdfWriter é então inicializado para adicionar a página resultante e preparar o PDF para download. A resposta HTTP, que retorna o PDF como anexo, é configurada com o nome "notificacao_{codigo_verificador}.pdf", em que o {codigo_verificador} representa um código único que identifica o documento e permite que o notificado valide a notificação futuramente.

Figura 53 - Código view para geração de PDF.

```
@login_required
@user_passes_test(is_staff)
def generate_pdf(request, codigo_verificador):
    notificacao = Notificacao.objects.get(codigo_verificador=codigo_verificador)

    pdf_template_path  = os.path.join(settings.BASE_DIR, 'noti/static/modelonotif.pdf')

    pdf_template = PdfReader(pdf_template_path)
    pdf_writer = PdfWriter()

    page = pdf_template.pages[0]

    packet = io.BytesIO()
    can = canvas.Canvas(packet, pagesize=letter)

    can.setFont("Helvetica", 12)
    can.drawString(50, 50, f"Código Verificador: {notificacao.codigo_verificador}")
    can.save()

    packet.seek(0)

    new_pdf = PdfReader(packet)
    new_page = new_pdf.pages[0]

    page.merge_page(new_page)
    pdf_writer.add_page(page)

    response = HttpResponse(content_type='application/pdf')
    response['Content-Disposition'] = f'attachment; filename="notificacao_{codigo_verificador}.pdf"'

    pdf_writer.write(response)

    return response
```

Fonte: Do autor.

Na *figura 54*, segue-se o código sobre arquivo, o primeiro salva o arquivo no banco de dados, já demonstrado em outros códigos. E a segunda view, é acerca sobre download deste arquivo, em primeiro momento é selecionado um objeto em específico para fazer download, é ‘avisado’ ao HTTP que tem um arquivo para download e no final a nomeação deste arquivo com o nome definido no sistema.

Figura 54 - Código view para criação e download de arquivos.

```
@login_required
@user_passes_test(is_staff)
def criaarquivo(request):
    if request.method == 'POST':
        form = ArquivoForm(request.POST,request.FILES)
        if form.is_valid():
            form.save()
            return HttpResponse('The file is saved')
    else:
        form =ArquivoForm()
        context = {
            'f':form,
        }
    return render(request, 'upload_arquivo.html', context)

@login_required
def download_arquivo(request, arquivo_id):
    arquivo = Arquivo.objects.get(id=arquivo_id)
    response = HttpResponse(arquivo.arquivo, content_type='application/octet-stream')
    response['Content-Disposition'] = f'attachment; filename="{arquivo.arquivo.name}"'
    return response
```

Fonte: Do autor.

O validador é uma view que serve para a parte de validação de um código gerador, demonstrado na *figura 55*, ele recebe via o método POST o código verificador que o usuário está analisando, esse código será procurado nos objetos das notificações que correspondem ao mesmo código verificador, e analisa essa notificação como se está regular, prazo, data notificada, o fiscal e assim passa essas informações via contexto para o template ‘validador.html’.

Figura 55 - Código view do Validador

```
def validador(request):
    if request.method == 'POST':
        codigo_verificador = request.POST.get('codigo_verificador')
        try:
            notificacao = Notificacao.objects.get(codigo_verificador=codigo_verificador)
            context = {
                'codigo_verificador': codigo_verificador,
                'codigo_valido': True,
                'prazo': notificacao.prazo,
                'data_notif': notificacao.data,
                'data_final': notificacao.data + timedelta(days=30),
                'fiscal': notificacao.fiscal,
            }
        except Notificacao.DoesNotExist:
            context = {
                'codigo_verificador': codigo_verificador,
                'codigo_valido': False,
            }
        return render(request, 'validador.html', context)
    else:
        return render(request, 'validador.html')
```

Fonte: Do autor.

Até agora quase todas as views foram de uso exclusivo dos usuários staff, a partir de agora teremos algumas que são focadas exclusivamente nos usuários não staff. A *figura 56* traz uma view que traz a respeito do histórico do usuário, começa com a criação de um dicionário em branco, em seguida via request.user qual usuário está conectado, imediatamente é selecionado um objeto dentro da Entidade que contenha o mesmo usuario, e logo depois via uma chave 'db' é filtrado as notificações que contenha a entidade selecionada. Passando por fim essa chave em contexto para o template 'buscanotiusuario.html'.

Figura 56- Código view do histórico do usuário.

```
@login_required
@user_passes_test(is_not_staff)
def historicousuario(request):
    data = {}
    usuario = request.user
    entidade = Entidade.objects.get(usuario__user=usuario)
    data['db'] = Notificacao.objects.filter(entidade=entidade)
    return render(request, 'buscanotiusuario.html', data)
```

Fonte: Do autor.

O usuário não staff possui também uma página com certos filtros, neste caso os filtros são notificações em aberto, fora do prazo e total. Esses filtros aparecem na

primeira página, ou seja no index/dashboard, porém ao clicar ele redireciona para uma página de filtros, que dependendo da action passada por url deve mostrar dados diferentes. No código abaixo da *figura 57*, começa com a variável de usuário que recebe um request.user, e prazo limite que faz um cálculo, para saber se na data atual tem alguma notificação fora do prazo. Posteriormente é feito um fluxo de controle, na qual se não tiver autenticado volta para o index, e se tiver é feito uma leitura do action, se trazer na url ‘aberto, é retornado todas notificações em aberto; Se trazer ‘foraprazo’ retorna todas notificações fora do prazo. E por fim se trazer ‘aberto’, retorna todas notificações. Também traz uma paginação no final do código.

Figura 57 - Código view do dos filtros do usuário não-Staff.

```
@login_required
@user_passes_test(is_not_staff)
def filtrosue(request):
    usuario =   request.user
    prazo_limite = timezone.now() - timedelta(days=30)
    if not user.is_autheticated:
        redirect('index')
    entidade = Entidade.objects.get(usuario__user = usuario)
    if not usuario.is_staff:
        action = request.GET.get('action')
        data ={}
        if action == 'aberto':
            notificacoes =  Notificacao.objects.filter(Q(entidade=entidade) & Q(regularidade=False))
        elif action == 'foraprazo':
            notificacoes = Notificacao.objects.filter(Q(data__lte=prazo_limite) & Q(regularidade=False) & Q(entidade=entidade))
        elif action=='total':
            notificacoes = Notificacao.objects.filter(entidade=entidade)
        else:
            return redirect('index')

    paginator = Paginator(notificacoes, 10)
    page_number = request.GET.geeracaoFot('page')
    data['db'] = paginator.get_page(page_number)

    return render(request, 'filtrosue.html', data)
```

Fonte: Do autor.

De mesmo modo o filtro do usuário staff tem um código próprio e uma página própria, seguindo a mesma lógica que a anterior, visto na *figura 58*.

Figura 58 - Código view dos filtros do usuário Staff.

```
@login_required
@user_passes_test(is_staff)
def filtros(request):
    prazo_limite = timezone.now() - timedelta(days=30)

    action = request.GET.get('action')
    data ={}
    if action == 'aberto':
        notificacoes = Notificacao.objects.filter(regularidade=False)
    elif action == 'foraprazo':
        notificacoes = Notificacao.objects.filter(Q(data__lte=prazo_limite) & Q(regularidade=False))
    elif action=='total':
        notificacoes = Notificacao.objects.all()
    elif action=='regular':
        notificacoes = Notificacao.objects.filter(regularidade=True)
    else:
        return redirect('index')

    paginator = Paginator(notificacoes, 10)
    page_number = request.GET.get('page')
    data['db'] = paginator.get_page(page_number)

    return render(request, 'filtros.html', data)
```

Fonte: Do autor.

A página de usuário, vista na *figura 59*, utiliza-se também a função de action, ele recebe um usuário específico através do id, e a action com um padrão view, se este padrão for escolhido ele irá retornar todos dados do usuário e qual entidade ele está ligado, passando por contexto. Se for delete, ele irá setar o usuário na entidade para None e assim deletando o usuário da base de dados.

Figura 59 - Código view para página do usuário.

```
@login_required
@user_passes_test(is_staff)
def paginausuario(request, id):
    usuario = ModelUsuario.objects.get(id=id)
    action = request.GET.get('action', 'view')

    if action == 'view':
        data={}
        data['db'] = ModelUsuario.objects.get(id=id)
        data['et'] = Entidade.objects.get(usuario=usuario)
        return render(request, 'paginausuario.html', data)

    elif action == 'delete':
        try:
            entidade = Entidade.objects.get(usuario=usuario)
            entidade.usuario = None
        except Entidade.DoesNotExist:
            pass
        ModelUsuario.delete()
        return redirect('index')
    else:
        return render(request, 'paginausuario.html', {'usuario_entidade': ModelUsuario})
```

Fonte: Do autor.

Por fim fica o código de Liberação, que nas primeiras duas views deste grupo é uma reutilização do código, como pode ver abaixo na *figura 60*.

Figura 60 - Código view para liberação.

```
def liberacao(request):
    form = LiberacaoForm()
    if request.method == "POST":
        form = LiberacaoForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, 'Pedido de liberação feito')
    return render(request, 'liberacao.html', {'form': form})
```

Fonte: Do autor.

Figura 61 - Continuação dos códigos da figura 61.

```
if request.is_ajax():
    data = [
        {
            'email': dbs.email,
            'chave': dbs.chave,
            'datacriacao':dbs.data_criacao,
        }
        for dbs in dados['db']
    ]
    return JsonResponse(data, safe=False)

dados['form'] = form
```

Fonte: Do autor.

Por fim, a parte mais importante da liberação, é a sua listagem, visto na *figura 62*, na qual vai se receber a liberação que vai receber por parâmetro através do email, e também seta uma variável da chave, e posteriormente é feito uma variável chamada tam, que vai receber o tamanho através da função len da variável chave. É feito um fluxo de controle que se o tamanho for 11, é cpf e se for 14 é cnpj, dependendo do que for, retorna a entidade que tenha um desses dois, e logo após é feito a leitura de qual action foi passado por url, neste caso temos duas, liberar e deleter.

Se for a opção ‘liberar’, é porque os fiscais permitiram o usuário a se ligar com uma empresa, sendo assim o código torna a conta ativa no sistema e seta o campo de usuário da entidade para este usuário, salvando este comportamento no banco de dados. Se for a opção ‘deletar’, a liberação é deletada, e o pedido de liberação deletado.

Figura 62 - Código view para filtro de liberação.

```
@login_required
@user_passes_test(is_staff)
def filtroliberacao(request, email):
    liberacao = get_object_or_404(Liberacao, email=email)
    chave = liberacao.chave
    tam = len(str(chave))

    usuario = get_object_or_404(ModelUsuario, email=email)

    if tam == 11:
        entidade = get_object_or_404(Entidade, cpf=chave)
    elif tam == 14:
        entidade = get_object_or_404(Entidade, cnpj=chave)

    action = request.GET.get('action', 'deletar')

    if action == 'liberar':
        usuario.is_active = True
        usuario.save()
        entidade.usuario = usuario
        entidade.save()
        liberacao.delete()
        return redirect('/listaliberacao')

    elif action == 'deletar':
        liberacao.delete()

        return redirect('/listaliberacao')
```

Fonte: Do autor.

O desenvolvimento desse sistema envolveu a criação e organização de uma considerável quantidade de códigos e de tecnologias externas. Essa base de código, agora completa, culminou na geração desse sistema, cujas principais características e resultados são apresentados na seção a seguir.

,

5. RESULTADOS E DEMONSTRAÇÃO

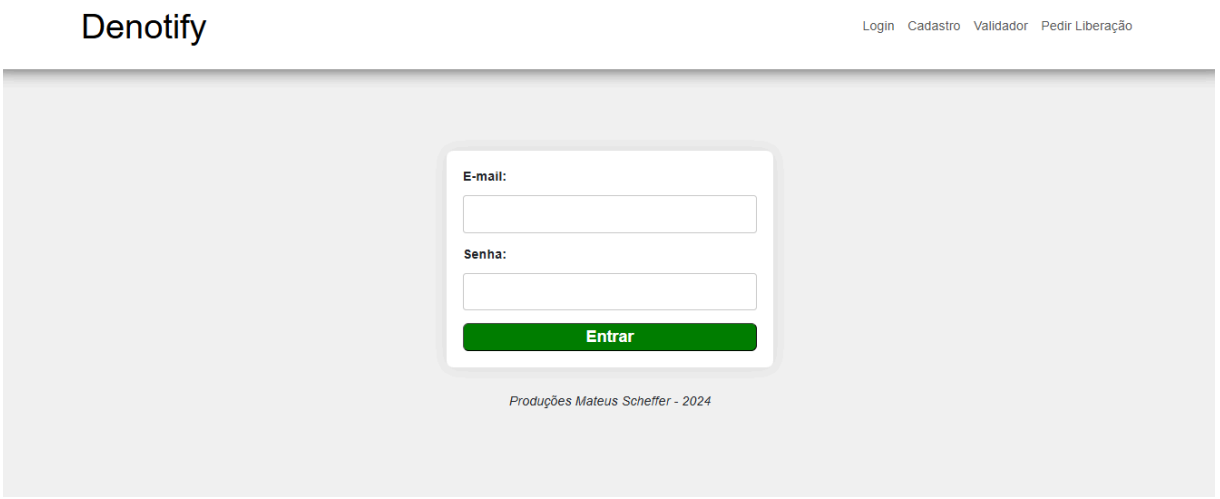
Nesta etapa, realiza-se a demonstração do sistema e a apresentação dos resultados obtidos. Nesse momento, são expostos o funcionamento e as características do sistema após todo o percurso de desenvolvimento realizado até então.

O projeto é um sistema web para organização, emissão e acompanhamento de notificações de um setor de tributos de um determinado departamento de tributos de uma cidade do extremo sul catarinense, onde o autor é alocado, o nome do projeto é '*Denotify*', e se encontra hospedado na plataforma PythonAnywhere a partir do endereço **mateusffe.pythonanywhere.com** e acessível a todos.

5.1 RESULTADOS DO SISTEMA

Nesta seção será demonstrada os resultados do sistema, sendo a primeira tela mostrada na *figura 63*, é a tela de login, a primeira tela que se vê ao entrar no sistema, ela contém o menu com todas funções que um usuário sem cadastro pode ter acesso, o cadastro, o validador, e o pedido de liberação.

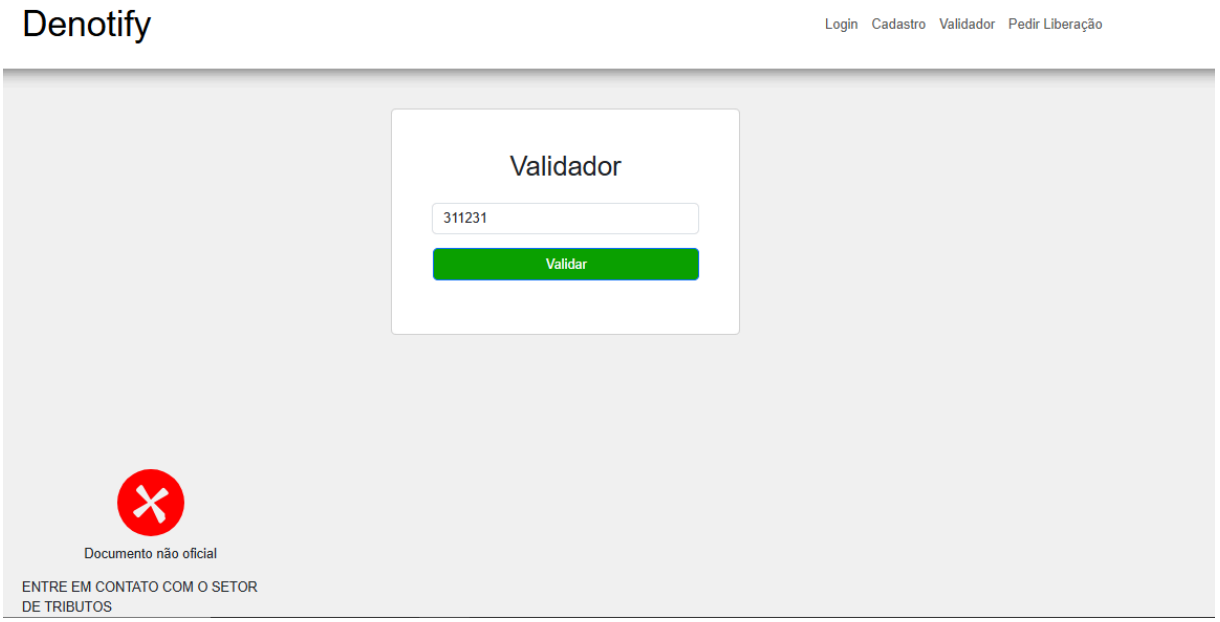
Figura 63 - Tela inicial do sistema.



Fonte: Do autor

O validador tem o mesmo comportamento em todo o site, ao adentrar ele e colocar um código verificar aleatório, ele dará a mensagem que aquele código não reflete a uma notificação oficial do setor, como visto na *figura 64*.

Figura 64 - Tela do validador com Documento não oficial.



Fonte: Do autor.

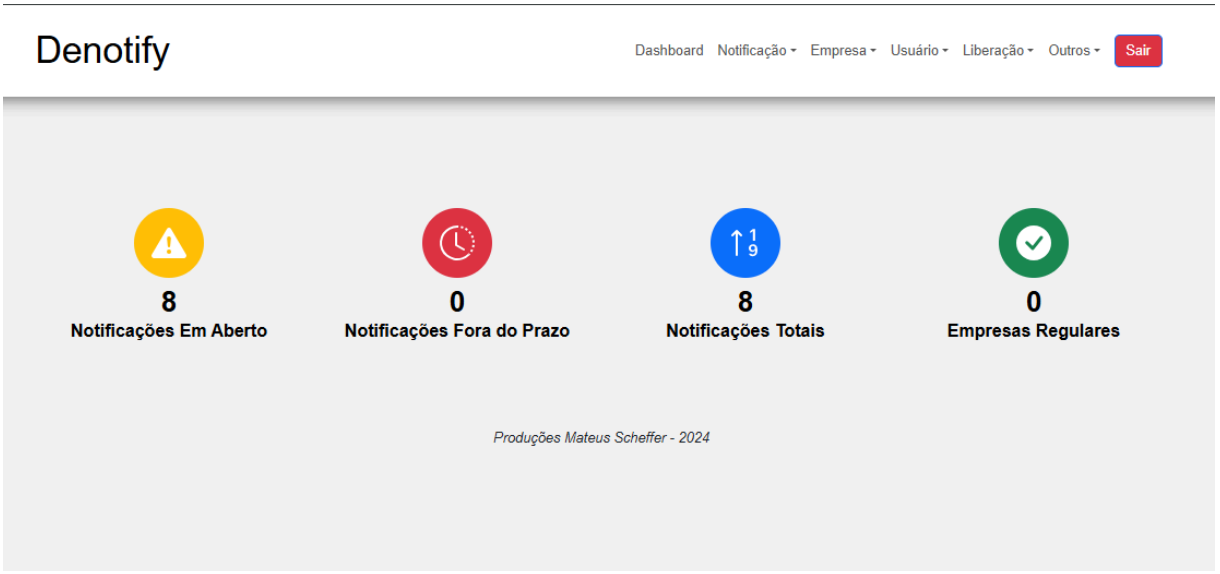
Agora se colocarmos um código de verificador válido, o resultado já se dá como um documento válido, a data que foi notificado, o fiscal que o notificou e o prazo, como visto na *figura 65*.

Figura 65 - Tela do validador com Documento oficial.

Fonte: Do autor.

Adentrando no sistema pelo usuário staff, temos o seguinte dashboard abaixo, contendo os filtros de notificações em aberto, fora do prazo, já regularizadas e o total, tudo isso em tempo real e bem visual para facilitar o acompanhamento. O usuário staff ou fiscal, tem acesso ao menu de notificação, empresa, usuário, e liberação, em cada um tendo acesso ao cadastro, listagem e outras funções, demonstrado na *figura 66.*

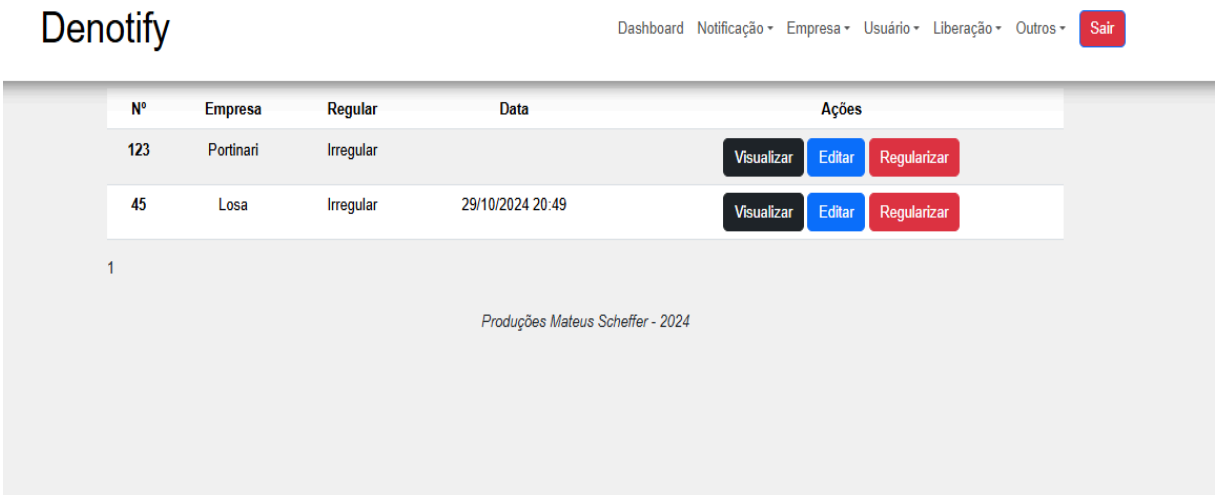
Figura 66 - Tela inicial do dashboard com filtros



Fonte: Do autor.

Ao clicar em um destes filtros, eles levam a uma página com mesmo layout que listam conforme filtro pede, como exemplo, pegarei as notificações em aberto em exemplo na *figura 67*.

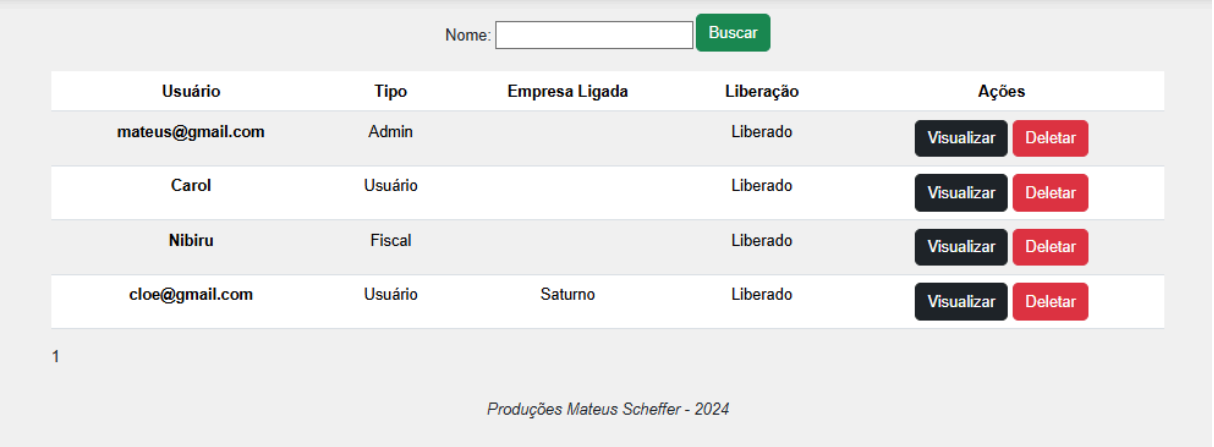
Figura 67 - Tela listagem de notificações.



Fonte: Do autor.

Nas buscas comum, ou seja, fora dos filtros, utiliza-se também o mesmo layout, porém com um campo de busca, como exemplo foi pego a busca de usuários, que além de poder fazer a busca, é separado por campos como, qual o username, qual o tipo de usuário, se ele é liberado para o sistema, visto na *figura 68*.

Figura 68 - Tela listagem de usuários.



Fonte: Do autor.

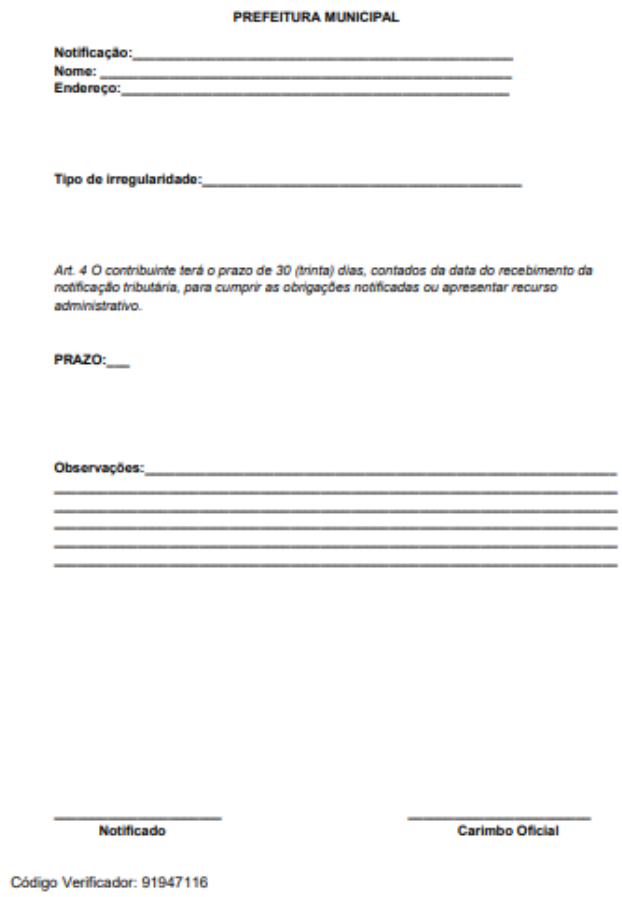
Por último, na *figura 69*, para exemplificar o resultado do usuário não staff, é mostrado o dashboard deste abaixo. Onde também contém filtros, com as notificações ligadas a sua empresa em aberto, fora do prazo e total. Além disso, é possível ver seu histórico.

Figura 69 - Tela dashboard usuário não-Staff.

Fonte: Do autor.

Ao notificar é gerado um documento de notificação com código verificador, que pode ser usado para preencher os dados de uma empresa posteriormente, como podemos ver no documento abaixo, na *figura 70.*

Figura 70 - Documento gerado para notificação.



PREFEITURA MUNICIPAL

Notificação:_____
Nome: _____
Endereço:_____

Tipo de irregularidade:_____

Art. 4 O contribuinte terá o prazo de 30 (trinta) dias, contados da data do recebimento da notificação tributária, para cumprir as obrigações notificadas ou apresentar recurso administrativo.

PRAZO:___

Observações:_____
_____
_____
_____
_____
_____

_____ **Notificado**

_____ **Carimbo Oficial**

Código Verificador: 91947116

Fonte: Do autor.

Ao ser notificado, é possível visualizar a notificação contendo todos os dados da empresa notificada, além dos pareceres e arquivos vinculados a essa notificação, como visto na *figura 71*.

Figura 71 - Tela para visualização de notificação.

Fonte: Do autor.

Encerrando a apresentação dos resultados das principais telas e funcionalidades, a seção a seguir apresenta um caso de uso acompanhado de um roteiro resumido para exemplificar a utilização do sistema.

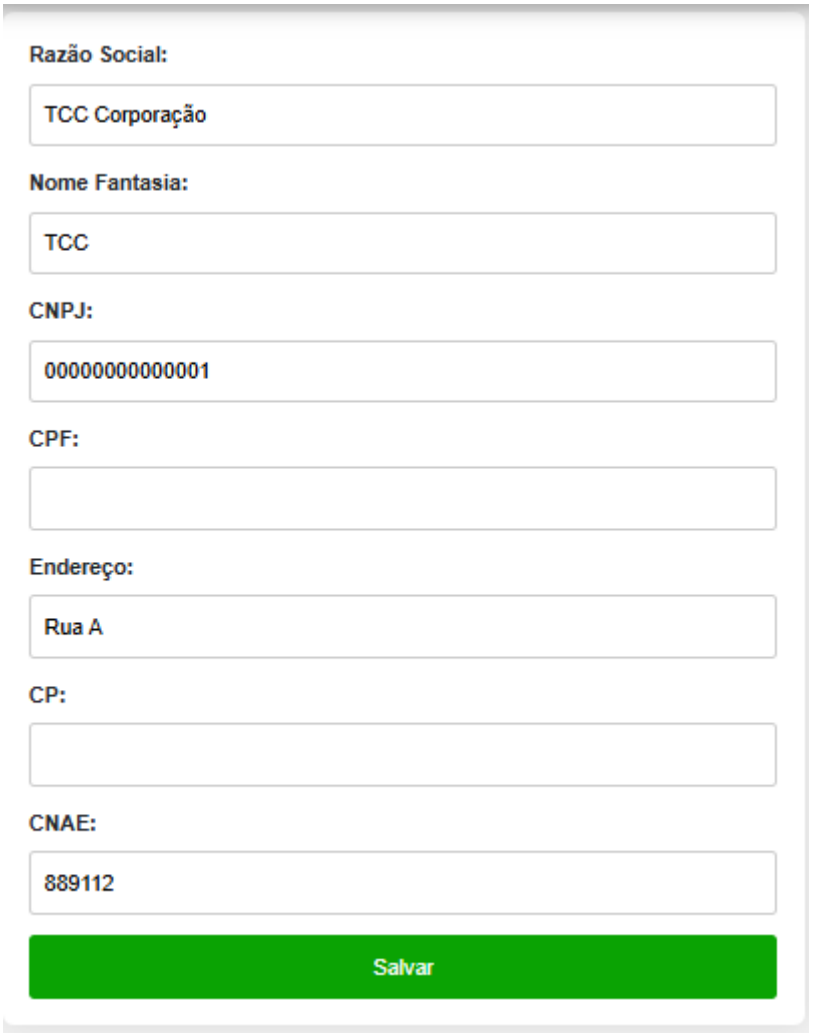
5.2 CASO DE USO

Nesse caso de uso, foram empregadas as funcionalidades de cadastro de usuário, cadastro de entidade e liberação de acesso, deixando de lado as funcionalidades adicionais desenvolvidas no sistema, como cadastro de notificação, verificação, upload de arquivos e pareceres.

Inicialmente, será criado um perfil de usuário, seguido do cadastro de uma entidade vinculada a esse usuário, que, ao final, deverá solicitar a liberação de acesso. Foi escolhido o nome de usuário "tcc@gmail.com", cuja empresa, "TCC Corporação", foi notificada com o CNPJ "30450123000139". Para esse processo, o fiscal realiza o cadastro da empresa notificada. A Figura 73 apresenta um formulário

de cadastro de entidade, no qual são inseridas as informações necessárias para a realização do cadastramento.

Figura 72 - Tela de cadastro de empresa no caso de uso.



Fonte: Do autor.

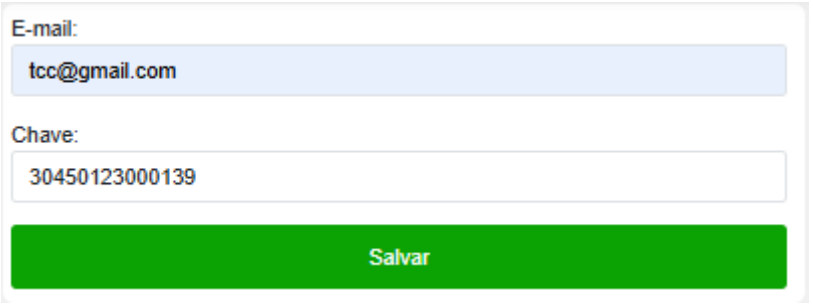
Posteriormente, na figura 73, o contribuinte, como proprietário da empresa, procederá ao cadastramento de suas credenciais de acesso para visualizar as notificações, tanto as que já foram geradas quanto as que estão programadas para o futuro.

Figura 73 - Tela de cadastro de usuário no caso de uso.

Fonte: Do autor.

Finalizando o processo de cadastro, na tela da figura 74, o contribuinte formaliza a solicitação de acesso ao sistema, fornecendo seu endereço de e-mail pessoal e o número de chave, que neste caso pode ser o CNPJ ou CPF, dependendo do tipo de empresa.

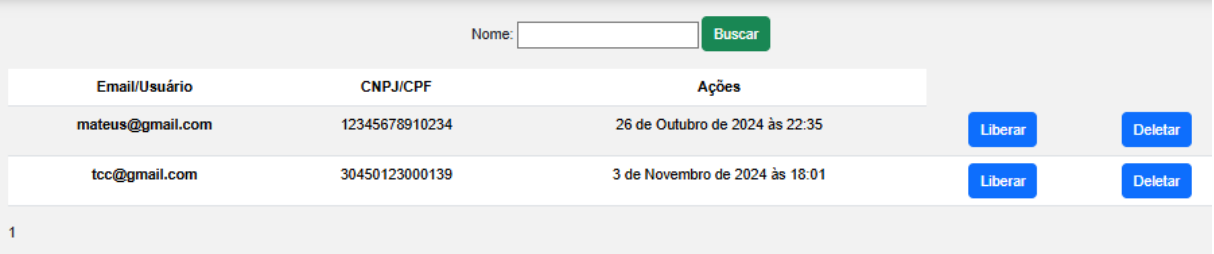
Figura 74 - Tela de pedido de liberação.



Fonte: Do autor.

E para o fiscal, na tabela da figura 75, é demonstrado os pedidos de liberação, na qual é listado esse novo pedido que após uma avaliação de dados de empresa e pedido externo e manual, defere ou indefere o pedido.

Figura 75 - Tela de listagem de liberação no caso de uso.



Fonte: Do autor.

Após deferido, o contribuinte agora tem acesso a empresa, bem como as notificações e ao histórico, tornando o setor público mais transparente com suas notificações e mais eficiente por acompanhar, trazendo um sistema que organiza a fiscalização e facilita o seu trabalho.

6. AVALIAÇÃO

Nesta etapa, o sistema foi avaliado por outra servidora do setor, ocupante do mesmo cargo e com funções similares. Para a avaliação, foi elaborado um questionário na ferramenta Google Forms, dividido em três grupos temáticos: usabilidade, utilidade e uso futuro. As respostas foram estruturadas com base na escala de Likert, variando de 1 a 5, onde 1 representa "não concordo", 5 "concordo", e os valores intermediários correspondem a "não concordo parcialmente", "neutro" e "concordo parcialmente".

Antes da aplicação do questionário, a participante teve acesso livre ao sistema e realizou uma tarefa simples de cadastro de uma notificação. Após familiarizar-se com o funcionamento e realizar as funções solicitadas, foi convidada a responder o questionário.

Conforme se visualiza na tabela 04, o primeiro grupo, referente à usabilidade e design, foram formuladas seis perguntas. As três iniciais focaram no design e nos aspectos visuais do sistema, enquanto as três subsequentes abordaram especificamente a usabilidade.

O segundo grupo, sobre utilidade e uso no trabalho, contou com quatro perguntas voltadas para avaliar se o sistema contribui para o desempenho das atividades laborais e se promove o aumento da produtividade.

Por fim, a terceira seção, relacionada ao uso futuro, apresentou duas perguntas destinadas a verificar se o sistema seria utilizado no dia a dia de trabalho e se sua adoção deveria ser considerada pelo departamento.

Ao final do questionário, foi disponibilizado um campo para que a participante expressasse livremente sua opinião sobre o sistema, incluindo aspectos não contemplados nas perguntas, e atribuísse uma nota de 1 a 10, sendo 10 a avaliação máxima.

Tabela 4 - Perguntas de avaliação do sistema

Grupo	Perguntas
	A organização visual do sistema facilita a navegação e torna o uso intuitivo.
	As cores e o contraste utilizados no sistema são agradáveis e facilitam a

USABILIDADE E DESIGN DO SISTEMA	leitura.
	O design do sistema transmite uma imagem profissional e confiável.
	As instruções do sistema para preencher formulários ou realizar ações eram suficientes?
	Você sentiu que a quantidade de passos para realizar uma tarefa era adequada?
	O sistema respondeu de maneira rápida e eficiente às suas ações?
UTILIDADE E USO NO TRABALHO	As funcionalidades do sistema atendem bem às necessidades do meu trabalho diário.
	O sistema facilitaria o acesso e a organização das informações de que preciso para o meu trabalho.
	As informações fornecidas pelo sistema são precisas e confiáveis para a tomada de decisões.
	O uso do sistema contribuiria para melhorar minha produtividade e desempenho no trabalho.
USO FUTURO	Vejo valor em utilizar este sistema para melhorar meu desempenho no trabalho
	Você acredita que seu departamento deveria buscar adotar esse sistema.
PERGUNTA LIVRE	Opinião sobre o sistema
NOTA LIVRE (0-10)	Nota para o Sistema

Fonte: Do autor.

Após a elaboração do questionário com base na escala Likert, este foi aplicado a uma profissional atuante no mesmo setor de tributos em um município catarinense. A aplicação ocorreu de forma voluntária e com concordância da participante. A seguir, são apresentadas as respostas fornecidas, acompanhadas de análises individuais de cada questão e do grupo de perguntas como um todo.

O primeiro grupo avaliado foi denominado "Usabilidade e Design do Sistema", composto por seis perguntas. Na primeira questão, foi analisado se a organização visual do sistema facilita a navegação e torna o uso intuitivo. A resposta atribuída foi 5, o valor máximo da escala, indicando que a organização visual do sistema foi considerada eficiente, promovendo uma experiência intuitiva.

Na segunda pergunta, verificou-se se as cores e os contrastes utilizados no sistema facilitam a leitura. A avaliação também foi 5, evidenciando que as cores foram percebidas como agradáveis e os contrastes adequados, sem causar impacto visual negativo.

A terceira questão examinou se o design do sistema transmite profissionalismo e confiabilidade. Novamente, a participante atribuiu a nota máxima, 5, indicando que o sistema foi considerado satisfatório a ponto de inspirar confiança no serviço.

Na quarta pergunta, referente às instruções para o preenchimento dos formulários, foi atribuída a nota 4. Esse resultado sugere que, embora as instruções sejam claras e presentes, houve percepção de que melhorias ou incrementos podem ser realizados.

A quinta questão avaliou a quantidade de passos necessários para realizar tarefas no sistema. A nota atribuída foi 4, refletindo que, apesar de os passos serem seguros e organizados, a quantidade foi considerada excessiva, apontando uma oportunidade para otimização.

Por fim, a sexta pergunta do grupo tratou sobre a eficiência e a rapidez nas respostas do sistema às ações do usuário. A avaliação, novamente com nota 4, reforçou a percepção de que o sistema, apesar de funcional e eficiente, apresenta áreas a serem aprimoradas em termos de performance.

Em síntese, o grupo "Usabilidade e Design do Sistema" apresentou avaliações positivas, destacando aspectos como organização visual, confiabilidade e design profissional, mas também evidenciou oportunidades de melhoria em eficiência, clareza das instruções e otimização dos processos.

O próximo grupo analisado é o de “Utilidade e Uso no Trabalho”, com o objetivo de avaliar a relevância e aplicabilidade da ferramenta no cotidiano profissional. Esse grupo é composto por quatro perguntas específicas.

Na primeira pergunta do segundo grupo, temos a necessidade de saber se o sistema e suas funcionalidades atenderam bem as necessidades do trabalho da

respondente, o quesito escolhido foi o 5, assim confirmando que o sistema atenderia.

Na segunda pergunta, temos a necessidade de saber se o sistema ajudaria na organização e acesso das informações que se é preciso no ambiente de trabalho, a respondente, analisou como quesito 4, ou seja, o sistema precisa adicionar alguns tipos de informações, mas é suficiente para utilizar no ambiente de trabalho no momento.

Na terceira pergunta, se avalia se as informações apresentadas pelo sistema são confiáveis, a respondente avaliou como neutro ou não sei, no quesito 3, levando em consideração essa resposta, é possível levantar que a respondente não tinha certeza dos dados pois não tinha o cruzamento com o sistema de utilização comum do setor ou que falta informações e campos mais precisos.

A última pergunta seria sobre se o uso contribuiria com a produtividade, a respondente respondeu no quesito 5, sendo assim afirma que contribuiria para melhorar o desempenho e produtividade.

Encerrando este grupo, mostra-se promissor o sistema para o uso no dia a dia do trabalho, levando a consideração da ressalva feita sobre confiança nos dados demonstrado, sendo assim é percebido que se é preciso de um incremento nos campos de dados e um uso cruzado de informações, como o uso de api's de sistemas como receita federal, simples nacional e afins.

O próximo grupo, representado pelo título 'Uso futuro', tem como objetivo saber o intuito da utilização desse sistema no futuro profissional da respondente ou no setor, com duas perguntas e encerrando o ciclo de questionário no modo Likert.

Na primeira pergunta é avaliado se a respondente vê valor em usar o sistema no seu trabalho para melhorar o seu desempenho, sua avaliação foi o quesito 5, concordando que o sistema pode ser útil ao seu dia a dia.

A última pergunta deste grupo, é sobre se o departamento em questão deveria adotar o sistema, a respondente analisou como 4, concordando que deveria ser adotado o sistema, porém é presumível a partir de suas avaliações anteriores que precisa haver certas modificações.

Foi deixado para o final duas perguntas abertas, para a respondente responder, uma deixa em questão aberta a 'Opinião sobre o sistema' e a outra uma nota de 1 até 10 sobre o sistema.

Na pergunta aberta, a respondente avaliou da seguinte forma:

“Sistema muito bem elaborado, com uma boa compreensão dos requisitos e funcionalidades principais. Seria ideal que os departamentos incluíssem esse sistema, desde que fizessem alguns ajustes.”

E como nota a respondente avaliou como nota 9, ao atribuir esta nota, a respondente demonstra uma avaliação altamente positiva em relação ao sistema, indicando que este possui um grande potencial para otimizar os processos e aumentar a produtividade. A análise do questionário e da resposta aberta corrobora essa percepção, revelando que a entrevistada acredita que o sistema deve ser usado em seu setor, mesmo que precise de alguns ajustes.

Conclui-se na avaliação que o sistema precisa de algumas modificações pontuais, e que o projeto já criado e até mesmo o previsto, é capaz de suprir as necessidades atuais do setor.

7. CONSIDERAÇÕES FINAIS

Este estudo apresentou o desenvolvimento e a implementação do sistema web "Denotify", voltado para a gestão de notificações tributárias em um município do extremo sul catarinense. O sistema demonstrou-se eficaz ao organizar, emitir e acompanhar notificações tributárias, promovendo maior transparência e eficiência no setor de tributos.

A identificação e documentação dos requisitos funcionais e não funcionais, juntamente com a elaboração das regras de negócio, foram etapas fundamentais deste trabalho. Além disso, funcionalidades essenciais, como cadastro, consulta, edição e exclusão de notificações, foram desenvolvidas e implementadas com sucesso.

A avaliação do sistema foi realizada por meio de um questionário aplicado a uma colega de profissão com o mesmo cargo e do mesmo setor. Os resultados revelaram impactos positivos, destacando o aumento na organização e produtividade, bem como melhorias no desempenho das atividades do setor.

Os resultados indicam que o "Denotify" trouxe benefícios significativos para a administração pública municipal, incluindo maior controle sobre prazos e pendências tributárias, além de um acesso mais transparente às informações pelos contribuintes. Traz benefícios, como organização, produtividade e melhora de desempenho, na qual o setor neste presente momento já se teria impactos positivos.

Apesar dos resultados positivos, o estudo identificou limitações, como a necessidade de funcionalidades adicionais que foram postergadas devido às restrições de tempo e recursos. Assim, futuras melhorias incluem a incorporação de novos módulos, como a integração com sistemas de dados disponibilizados pela receita estadual e federal e a aplicação de inteligência artificial para análises preditivas.

Adicionalmente, este trabalho contribui para o debate sobre a democratização do acesso à informação e a transparência no setor público, ao promover a implementação de ferramentas tecnológicas que facilitam a interação entre os cidadãos e os órgãos administrativos. Para trabalhos futuros, sugere-se expandir a aplicação do "Denotify" para outros municípios, considerando adaptações legais e culturais, e explorar o impacto do sistema em médio e longo prazo, com o objetivo de validar sua eficácia em diferentes contextos administrativos.

O "Denotify" representa um avanço significativo na gestão tributária municipal e reforça a importância de investir em TICs para aprimorar a eficiência e a acessibilidade dos serviços públicos, abrindo caminhos para novos estudos e inovações na área.

8. REFERÊNCIAS

AABY, Anthony A.. **Theory Introduction to Programming Languages**. 2004. Disponível em: https://uilis.usk.ac.id/oer/files/original/be28b69d281a44f8b2b7aa8af26e3768.pdf. Acesso em: 14 jun. 2024.

ALURA. **Bootstrap: O que é, Documentação, como e quando usar**. 2023. Disponível em: https://www.alura.com.br/artigos/bootstrap?srsltid=AfmBOooj4d6b6fWychiABSF9go2Q3lBmhm7Jyu3w7ZD1ZSeArpXtf2Pe. Acesso em: 03 nov. 2024.

ALURA. **Django: o que é, para que serve e um Guia desse framework Python**. 2023. Disponível em: https://www.alura.com.br/artigos/django-framework?srsltid=AfmBOopBYtrNKbbuZax3oduUjfqd-MH8dEoJNoNwnhScG0vJaWNBHX78. Acesso em: 03 nov. 2024.

ALVES, Carlos Antonio Dias. **TECNOLOGIAS E NOVOS MODOS DE COMUNICAÇÃO**: a (re) invenção do conhecimento no ciberespaço na percepção dos docentes imigrantes digitais de uma universidade pública. A (RE) INVENÇÃO DO CONHECIMENTO NO CIBERESPAÇO NA PERCEPÇÃO DOS DOCENTES IMIGRANTES DIGITAIS DE UMA UNIVERSIDADE PÚBLICA. 2013. Disponível em: http://www.pgcl.uenf.br/arquivos/carlosantoniodiasalves2013_010220191544.pdf. Acesso em: 14 jun. 2024.

AWS. **O que é uma framework em programação e engenharia?** Disponível em: https://aws.amazon.com/pt/what-is/framework/#:~:text=Em%20engenharia%20e%20programa%C3%A7%C3%A3o%20de,todos%20os%20campos%20da%20engenharia.. Acesso em: 03 nov. 2024.

AWS. **O que é uma aplicação Web?** Disponível em: https://aws.amazon.com/pt/what-is/web-application/#:~:text=Uma%20aplica%C3%A7%C3%A3o%20Web%20%C3%A9%20um,de%20forma%20conveniente%20e%20segura.. Acesso em: 14 jun. 2024.

BOSTON UNIVERSITY. **HISTORY OF HTML**. Disponível em: https://www.bu.edu/lernet/artemis/years/2020/projects/FinalPresentations/HTML/historyofhtml.html. Acesso em: 14 jun. 2024

BRASIL. BRASIL. . **CONSTITUIÇÃO DA REPÚBLICA FEDERATIVA DO BRASIL DE 1988**. Disponível em: https://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm. Acesso em: 14 jun. 2024.

BRITANNICA. **World Wide Web**. Disponível em: https://www.britannica.com/topic/World-Wide-Web#:~:text=The%20development%20of%20the%20World,communication%20between%20servers%20and%20clients.. Acesso em: 14 jun. 2024.

BROCK, Allen Wirfs; EICH, Brendan. **JavaScript: The First 20 Years**. 2020. 189 f. Tese (Doutorado) - Curso de S.I, Sd, S.I, 2020.

CETIC. **TIC Governo Eletrônico - 2023**. Disponível em: https://cetic.br/pt/pesquisa/governo-eletronico/indicadores/. Acesso em: 03 nov. 2024.

COMUNITAS. **O uso da tecnologia na gestão pública**. São Paulo, 2024.

CONGRESSO NACIONAL. Estabelece princípios, garantias, direitos e deveres para o uso da Internet no Brasil.. . Brasil, Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2011-2014/2014/lei/l12965.htm. Acesso em: 03 nov. 2024.

DJANGO. **Django Documentation**. Disponível em: https://docs.djangoproject.com/en/5.0/. Acesso em: 14 jun. 2024.

EESCJR. **Sistemas Web: o que são e quais os seus benefícios**. Disponível em: https://eescjr.com.br/blog/sistemas-web-definicao-beneficios/. Acesso em: 03 nov. 2024.

ENCÍCLOPEDIA JURÍDICA PUCSP. **Notificação**. Disponível em: https://enciclopediajuridica.pucsp.br/verbete/306/edicao-1/notificacao. Acesso em: 03 nov. 2024.

ENCYCLOPEDIA BRITANNICA. **Markup language**. Disponível em: https://www.britannica.com/technology/markup-language. Acesso em: 03 nov. 2024.

ENCYCLOPEDIA BRITANNICA. **Computer programming language**. Disponível em: https://britannica.com/technology/computer-programming-language. Acesso em: 03 nov. 2024.

FEBRABAN. **Quase 90% dos brasileiros usam internet**. Disponível em: https://febrabantech.febraban.org.br/temas/inovacao/quase-90-dos-brasileiros-usam-internet. Acesso em: 14 jun. 2024.

FLANAGAN, David. **Javascript:**: o guia definitivo. [S. L.]: O'Reilly Media,Inc, 2011.

FURGERI, Sérgio. **O papel das linguagens de marcação para a Ciência da Informação**. Campinas: Transinformação, 2006.

GAIKWAD, Suraj Shahu; ADKAR, Prof Pratibha. **Https://www.irejournals.com/formatedpaper/1701173.pdf**. Disponível em: https://www.irejournals.com/formatedpaper/1701173.pdf. Acesso em: 03 nov. 2024.

GOTARDO, Reginaldo Aparecido. **LINGUAGEM DE PROGRAMAÇÃO**. Rio de Janeiro: Seses, 2015.

JAPPUR, Rafael Feyh. **MODELO CONCEITUAL PARA CRIAÇÃO, APLICAÇÃO E AVALIAÇÃO DE JOGOS EDUCATIVOS DIGITAIS**. 2014. 296 f. Tese (Doutorado) -

Curso de Pós- Graduação em Engenharia e Gestão do Conhecimento, Universidade Federal de Santa Catarina, Florianópolis, 2014.

JOHNSON, Ralph E.. Frameworks= (Components+Patterns). **Communications Of The Acm**, S.l, v. 10, n. 40, p. 1-4, out. 1997.

KUPARINEN, Simo. **Improving Web Performance by Optimizing Cascading Style Sheets (CSS): Literature Review and Empirical Findings**. 2023. 68 f. Tese (Doutorado) - Curso de Faculty Of Science, University Of Helsinki, Helsinki, 2023.

LUZ, Ramiro B. da. **Python e Django**. Rio de Janeiro: Rede Nacional de Ensino e Pesquisa, 2017.

MILLER, Stephan. **What Is Python Used For?** Disponível em: https://www.codecademy.com/resources/blog/what-is-python-used-for/. Acesso em: 03 nov. 2024.

MOZER, Merris; LOPER, Adriane Aparecida; SILVA, Danilo Augusto Bambini. **Sistemas Web**. Londrina: Londrina: Editora e Distribuidora Educacional S.A., 2014.

MOZILLA. **Markup**. Disponível em: https://developer.mozilla.org/en-US/docs/Glossary/Markup. Acesso em: 14 jun. 2024.

MOZILLA. **CSS básico**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Getting_started_with_the_web/CSS_basics. Acesso em: 03 nov. 2024.

MOZILLA. **JavaScript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 03 nov. 2024.

MOZILLA. **Django Web Framework (Python)**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django. Acesso em: 14 jun. 2024.

PASSO DE TORRES. Lei nº 167, de 1995. DISPÕE SOBRE O CÓDIGO TRIBUTÁRIO DO MUNICÍPIO DE PASSO DE TORRES E DÁ OUTRAS PROVIDÊNCIAS.. . Passo de Torres, SC, 1995. Disponível em: https://leismunicipais.com.br/codigo-tributario-passo-de-torres-sc. Acesso em: 03 nov. 2024.

PEFFERS, Ken; TUUNANEN, Tuure; ROTHENBERGER, Marcus A.; CHATTERJEE, S.. **A design science research methodology for information systems research**. Disponível em: https://www.researchgate.net/publication/284503626_A_design_science_research_methodology_for_information_systems_research. Acesso em: 03 nov. 2024.

PYTHON. **Beginner's Guide to Python**. Disponível em: https://wiki.python.org/moin/BeginnersGuide. Acesso em: 14 jun. 2024.

Receita Federal. **Notificação de Lançamento**. Disponível em: https://www.gov.br/receitafederal/pt-br/assuntos/meu-imposto-de-renda/malha-fiscal/notificacao. Acesso em: 14 jun. 2024.

RIBEIRO, Maria Ivanilse Calderon; COSTA, Juliana Braz da; BRAVIM, Jhordano Malacarne. **Projeto de Sistemas WEB**. Cuibá: N/A, 2015.

SABBAG, Eduardo. Manual de direito tributário. 4a ed., São Paulo, Saraiva, 2012

STATISTA. **Most used web frameworks among developers worldwide, as of 2024**. Disponível em: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. Acesso em: 03 nov. 2024.

VEROU, Lea. **CSS Secrets**. Sebastopol: O'reilly Media, Inc, 2015.