

UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO POLITÉCNICO DA UFSM
CURSO DE TÉCNICO EM INFORMÁTICA

Luíza Tavares da Silva

**O USO DE FRAMEWORKS NO PROCESSO DE DESENVOLVIMENTO
WEB: SISTEMA PARA CONTROLE DE HORÁRIOS E ATIVIDADES**

Santa Maria, RS
2023

Luíza Tavares da Silva

**O USO DE FRAMEWORKS NO PROCESSO DE DESENVOLVIMENTO WEB: SISTEMA
PARA CONTROLE DE HORÁRIOS E ATIVIDADES**

Trabalho de Conclusão de Curso apresentado no curso Informática do Colégio Politécnico, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Técnico em Informática**.

ORIENTADOR: Prof. Marcos Luís Cassal

Santa Maria, RS
2023

Luíza Tavares da Silva

**O USO DE FRAMEWORKS NO PROCESSO DE DESENVOLVIMENTO WEB: SISTEMA
PARA CONTROLE DE HORÁRIOS E ATIVIDADES**

Trabalho de Conclusão de Curso apresentado no curso Informática do Colégio Politécnico, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Técnico em Informática**.

Aprovado em 19 de janeiro de 2023:

Marcos Luís Cassal, Dr. (UFSM)
(Presidente/Orientador)

Bruno Mozzaquatro, Dr. (UFSM)

Giani Petri, Dr. (UFSM)

Santa Maria, RS
2023

RESUMO

O USO DE FRAMEWORKS NO PROCESSO DE DESENVOLVIMENTO WEB: SISTEMA PARA CONTROLE DE HORÁRIOS E ATIVIDADES

AUTORA: Luíza Tavares da Silva
ORIENTADOR: Marcos Luís Cassal

Este trabalho apresenta um estudo sobre o uso de *Frameworks* no processo de desenvolvimento *web* com a linguagem de programação *Python*. O objetivo deste trabalho é pesquisar as funcionalidades dos *Frameworks Bulma CSS* e *Django* e implementá-las no desenvolvimento de um sistema *web* para controle de horários e atividades, que foi desenvolvido no Colégio Politécnico da Universidade Federal de Santa Maria(UFSM) no curso Técnico em Informática. O presente estudo consiste em uma pesquisa de caráter descritivo. A partir da condução do processo de pesquisa foi possível concluir que o uso de *Frameworks* no processo de desenvolvimento *web* ajuda o desenvolvedor a realizar de forma otimizada algumas tarefas, sendo possível atingir os requisitos necessários para o projeto em desenvolvimento. Concluiu-se que este estudo proporcionou conhecimento da aplicação de *Frameworks* e também do processo de desenvolvimento *web*.

Palavras-chave: Django. Python. Bulma. Aplicações Web.

LISTA DE FIGURAS

Figura 1 – Uso do <i>CDN Link</i> no <i>Head</i> do cabeçalho <i>HTML</i> .	14
Figura 2 – Estrutura de colunas	15
Figura 3 – As colunas na página inicial do sistema.	15
Figura 4 – Estruturas do <i>section</i> .	16
Figura 5 – <i>sections</i> na página inicial do sistema.	17
Figura 6 – Estruturas do <i>container</i> .	17
Figura 7 – Estrutura do <i>footer</i> .	18
Figura 8 – Menu <i>dropdown</i> .	19
Figura 9 – Bloco <i>message</i> .	19
Figura 10 – <i>inputs</i> no formulário de cadastro de usuários.	20
Figura 11 – <i>Textarea</i> dentro do sistema desenvolvido.	20
Figura 12 – apresenta o <i>select</i> dentro do sistema desenvolvido.	20
Figura 13 – apresenta o <i>input</i> do tipo <i>file</i> dentro do sistema desenvolvido.	21
Figura 14 – Botão do sistema desenvolvido.	21
Figura 15 – <i>content list</i> (<i>ul</i> e <i>li</i>).	22
Figura 16 – <i>icons</i> no sistema desenvolvido.	22
Figura 17 – Classe <i><icons></i> .	22
Figura 18 – <i>table</i> na página de relatórios de atuação.	23
Figura 19 – Classe <i><table></i> .	23
Figura 20 – Fluxo de requisição.	25
Figura 21 – Utilização do arquivo <i>views.py</i> .	26
Figura 22 – Pasta <i>Templates</i> e a separação do código.	26
Figura 23 – Arquivo <i>models.py</i> .	27
Figura 24 – Estrutura de diretórios padrão.	28
Figura 25 – Estrutura da aplicação <i>home</i> .	29
Figura 26 – Estrutura de diretórios da aplicação <i>atividade</i> .	30
Figura 27 – Classe Pessoa.	30
Figura 28 – Arquivo <i>admin.py</i> .	31
Figura 29 – Informações inseridas na página administrativa do <i>Django</i> pelo <i>list_display</i> .	31
Figura 30 – Apresenta o arquivo <i>apps.py</i> .	32
Figura 31 – Apresenta o arquivo <i>apps.py</i> .	32
Figura 32 – Apresenta o arquivo <i>views.py</i> .	33
Figura 33 – Apresenta o arquivo <i>urls.py</i> .	33
Figura 34 – Apresenta o arquivo <i>forms.py</i> .	34
Figura 35 – Arquivo <i>models.py</i> da classe Atividade.	34
Figura 36 – Requisitos Funcionais do Sistema Desenvolvido.	43
Figura 37 – apresenta o diagrama de classes.	45
Figura 38 – Arquivo <i>models.py</i> da aplicação pessoa.	46
Figura 39 – Página de login criada com <i>Django Auth</i> .	47
Figura 40 – Apresenta <i>action</i> e o <i>method</i> de login.	48
Figura 41 – Código utilizado para validar o login do usuário.	48
Figura 42 – Arquivo <i>views.py</i> da aplicação <i>login</i> .	49
Figura 43 – Página de acesso ao sistema <i>web</i> com a mensagem de erro.	49
Figura 44 – <i>for message</i> inserido no login de pessoas.	50

Figura 45 – Página inicial.	51
Figura 46 – Menu de cadastros.	52
Figura 47 – Menu de relatórios.	52
Figura 48 – Menu de criar atividade e confirmar frequência.	52
Figura 49 – Página de cadastro de login de pessoas.	53
Figura 50 – Mensagem de que o usuário foi cadastrado com sucesso no sistema. ...	54
Figura 51 – Código utilizado para inserir as mensagens dentro do <i>for messages</i>	55
Figura 52 – Página de cadastro de pessoas.	55
Figura 53 – Página de cadastro de atividades.	56
Figura 54 – Página de relatório de pessoas.	57
Figura 55 – Página de relatório de atividades.	57
Figura 56 – Página de criação de atividades.	58
Figura 57 – Relatório das atividades criadas.	59
Figura 58 – Página de confirmação de frequência.	59
Figura 59 – Página de relatório de confirmação de frequência.	60
Figura 60 – Arquivo <i>utils.py</i> com a <i>class HtmlPdf()</i>	61
Figura 61 – <i>function geraPdf(imprimir, form)</i>	62
Figura 62 – <i>def get(self, *args, **kwargs)</i> no arquivo <i>views.py</i>	62
Figura 63 – <i>Template RelatoriosAtividadesPDF.html</i>	63
Figura 64 – Criação do <i>input</i> imprimir.	64
Figura 65 – botão imprimir.	64
Figura 66 – Botão buscar.	64
Figura 67 – Página de relatórios de atividades.	65
Figura 68 – <i>PDF</i> gerado das atividades cadastradas.	65
Figura 69 – A classe <i>AtividadeAddView</i>	66
Figura 70 – <i>path</i> no arquivo <i>urls.py</i>	66
Figura 71 – <i>Tempalte AtividaForm.html</i>	67
Figura 72 – Formulário preenchido na página de cadastro de pessoas.	68
Figura 73 – Nome que foi preenchido no formulário de cadastro de pessoas.	68
Figura 74 – <i>class AtividadeUpdateView(UpdateView)</i>	69
Figura 75 – <i>path</i> no arquivo <i>urls.py</i>	69
Figura 76 – Ícone para realizar alteração.	70
Figura 77 – Alteração de Informações.	70
Figura 78 – <i>class AtividadeDeleteView(DeleteView)</i>	71
Figura 79 – <i>path</i> no arquivo <i>urls.py</i>	71
Figura 80 – Ícone para realizar exclusão.	72
Figura 81 – Exclusão de um dado.	72
Figura 82 – Base de dados da aplicação.	74
Figura 83 – Definição do <i>search_fields</i> na aplicação <i>atividade</i>	75
Figura 84 – Arquivo <i>views.py</i> com a definição da <i>QuerySet</i> e o filtro.	76
Figura 85 – Arquivo <i>admin.py</i>	76
Figura 86 – Campo de busca da página administrativa do <i>Django</i>	77
Figura 87 – Arquivo <i>urls.py</i> da aplicação <i>home</i> com a definição da rota de <i>login</i>	78
Figura 88 – Arquivo <i>views.py</i> da aplicação <i>pessoa</i> com as opções de perfil.	78
Figura 89 – O <i>template RelatoriosPessoas</i> da aplicação <i>pessoa</i>	79
Figura 90 – apresenta o <i>template RelatoriosPessoas</i> da aplicação <i>pessoa</i> , mostrando como deve ser definida apresentar o nome completo dentro da interface	80
Figura 91 – Cadastro de pessoas	80

Figura 92 – Cadastro de <i>login</i> de pessoas	81
---	----

LISTA DE ABREVIATURAS E SIGLAS

NPM - Node Package Manager

MIT - Massachusetts Institute of Technology ou Instituto de Tecnologia de Massachusetts

UML – Unified Modeling Language ou Linguagem de Modelagem Unificada

IDE - Integrated Development Environment

UI - User Interface

CSS - Cascading Style Sheets ou Folhas de Estilo em Cascata

CWI - Centrum Wiskunde e Informatica

UFSM - Universidade Federal de Santa Maria

SUMÁRIO

1	INTRODUÇÃO	8
1.1	CONTEXTUALIZAÇÃO	8
1.2	OBJETIVOS	9
1.2.1	Objetivo Geral	9
1.2.2	Objetivos Específicos	9
1.3	METODOLOGIA	10
2	FRAMEWORKS	12
2.1	FRAMEWORKS DE FRONT-END	12
2.2	FRAMEWORKS DE BACK-END	13
2.3	FRAMEWORK BULMA CSS	14
2.3.1	Sistema Grid	15
2.3.2	Layout	16
2.3.3	Components	18
2.3.3.1	Componentes de Navegação	18
2.3.3.2	Componentes de Mídia	19
2.3.4	Forms	19
2.3.5	Elements	21
2.4	FRAMEWORK DJANGO	24
2.4.1	Arquitetura MTV do Django	24
2.4.2	Comando pip	27
2.4.3	Aplicações no Django	28
2.4.4	Migrations	35
2.4.4.1	O Comando Makemigrations	35
2.4.4.2	O Comando Migrate	35
2.4.4.3	Banco de Dados Django	36
2.5	FRAMEWORK DJANGO NA LINGUAGEM DE PROGRAMAÇÃO PYTHON	36
2.5.1	A Linguagem de Programação Python	36
2.5.2	O Uso da Linguagem de Programação Python em Aplicações Web	38
2.6	FERRAMENTAS UTILIZADAS NO DESENVOLVIMENTO WEB	38
2.6.1	Editor de Código Fonte	39
2.6.1.1	Visual Studio Code	40
2.6.2	IDE	40
2.6.2.1	Pycharm	41
3	APLICAÇÃO DOS FRAMEWORKS NO SISTEMA	42
3.1	CONTEXTO DO SISTEMA	42
3.2	MODELAGEM DO SISTEMA	43
3.2.1	Diagrama de Classes	44
3.3	DESENVOLVIMENTO	46
3.3.1	Migração do SQLite para PostgreSQL	72
3.3.2	Problemas no Desenvolvimento	75
4	CONCLUSÃO	82
	REFERÊNCIAS BIBLIOGRÁFICAS	83

1 INTRODUÇÃO

O presente trabalho, descreve um estudo sobre o uso de *Frameworks* no processo de desenvolvimento *web*, mais precisamente apresenta as funcionalidades dos *Frameworks*, sendo estes considerados uma estrutura de trabalho que atuam com algumas funções pré-estabelecidas que visam na otimização dos resultados na execução de tarefas (CAMARGO; MASIERO, 2005).

Os *Frameworks* utilizados no desenvolvimento deste estudo foram o *Bulma CSS* e o *Django*. Além disso, o presente estudo irá exemplificar o uso destes *Frameworks* com a implementação de um sistema para controle de horários e atividades, desenvolvido com a linguagem de programação *Python*.

O *Framework Bulma CSS* fornece componentes de *front-end* que permitem o desenvolvedor *web* criar interfaces mais modernas, responsivas e modulares. O *Bulma CSS* utiliza o mínimo de código *HTML*, facilitando sua escrita e leitura, além disso, para a utilização deste *Framework* não é necessário nenhum conhecimento sobre *CSS* (BULMA, 2022).

O *Framework Django* é utilizado para desenvolver aplicações *web* com a linguagem de programação *Python*, tornando o processo de desenvolvimento otimizado (DJANGO, 2022b).

Sendo assim, o principal objetivo deste trabalho é aplicar o estudo dos *Frameworks* utilizando a linguagem de programação *Python* para desenvolver um sistema *web* para o controle de horários e atividades, que foi desenvolvido no Curso Técnico em Informática do Colégio Politécnico da Universidade Federal de Santa Maria(UFSM).

A metodologia utilizada, foi uma pesquisa de caráter descritivo, que segundo Studybay (2022) tem como objetivo descrever uma população, processos ou outros assuntos, sem a interferência do pesquisador, sem estipular a análise de documentos para gerar relação ou interpretação, somente a descrição.

1.1 CONTEXTUALIZAÇÃO

O *Framework Bulma CSS* fornece componentes de *front-end* prontos para serem utilizados, sendo assim, pode-se facilmente criar interfaces mais modernas, responsivas e modulares, sendo possível implementar na sua interface apenas o que for necessário utilizar. O *Bulma CSS* utiliza o mínimo de código *HTML* possível para facilitar a sua leitura e escrita, portanto, para a utilização deste *Framework* não é necessário nenhum conhecimento de *CSS*. A versão utilizada no desenvolvimento deste estudo foi a V.0.9.4 (BULMA, 2022).

O *Django* foi desenvolvido para otimizar tarefas do desenvolvimento *web*, além disso, as funcionalidades do *Django* fazem desde a conexão com o banco de dados, até gerar automaticamente uma interface administrativa. Segundo Ramos (2018), a documentação do *Django* descreve ele como um *Framework* MTV, que significa: *Model*, *Template* e *View*, cada um possui uma função. O *View* é uma forma de processar os dados de uma *URL* específica, pois ela descreve qual informação foi apresentada. Os *Templates* servem para separar o conteúdo de apresentação, descrevendo como uma informação é apresentada. E o *Model*, é responsável pela estruturação do modelo de dados que será gerenciado pela aplicação. O *Django* foi utilizado com a linguagem de programação *Python* no processo de desenvolvimento da aplicação que se desenvolveu neste estudo.

A linguagem de programação *Python*, é uma linguagem de propósito geral e de alto nível, facilitando a escrita de um código limpo, simples e legível, que pode ser aplicada a muitas classes de diferentes de problemas, podendo ser utilizado tanto em aplicações mais simples quanto em aplicações mais complexas (PYTHON, 2022a). Segundo Python (2022a) o uso de agrupamento de instruções e a inclusão de tipos de dados de alto nível no Python, foram criadas devido a experiência de *Guido Van Rossum* na implementação de uma linguagem interpretada no grupo ABC onde ele também aprendeu sobre design de linguagem. Sendo assim, essa é a origem de muitos recursos do Python, embora todos os detalhes criados sejam totalmente diferentes. A versão do *Python* utilizada no desenvolvimento deste estudo foi a 3.10.7.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral aplicar as funcionalidades dos *Frameworks Bulma CSS* e *Django* com a linguagem de programação *Python* durante o processo de desenvolvimento de um sistema web.

1.2.2 Objetivos Específicos

- Estudar os *Frameworks Bulma CSS* e *Django* utilizado com a linguagem de programação *Python*;
- Aplicar o estudo do *Framework* de *front-end Bulma CSS* nas interfaces do sistema web desenvolvido;

- Aplicar o estudo do *Framework* de *back-end Django* utilizando a linguagem de programação *Python* para criação de uma aplicação *web* para controle de horários e atividades;

1.3 METODOLOGIA

O presente estudo consiste em uma pesquisa de caráter descritivo, que segundo Studybay (2022) tem como objetivo descrever uma população, processos ou outros assuntos, sem a interferência do pesquisador, sem estipular a análise de documentos para gerar relação ou a interpretação, tem somente como objetivo a descrição. Neste estudo, o objetivo foi descrever as funcionalidades e a aplicabilidade dos *Framework Bulma CSS* e *Django*. O trabalho também apresentará os resultados obtidos com a aplicação dos *Frameworks* no desenvolvimento de uma aplicação *web*.

Um *Framework* é composto por um conjunto de classes e pode ser definido como um sistema semi-completo e reutilizável que pode ser instanciado por um desenvolvedor de aplicações, estes são utilizados para solucionar problemas bem específicos e que atuam com funções pré-estabelecidas que ajudam na otimização de resultados (CAMARGO; MASIERO, 2005).

O *Bulma CSS*, é um *Framework* de *front-end* e foi utilizado para a criação das interfaces do sistema *web* desenvolvido neste estudo, criando interfaces modernas, responsivas e modulares.

O *Framework Django*, é um *Framework* de *back-end* e utiliza a linguagem de programação *Python* para criar aplicações *web*, neste estudo foi utilizado para a criação de um sistema *web* para controle de horários e atividades.

Segundo Mendonça (2014), o levantamento de requisitos é importante no desenvolvimento de um sistema, pois é onde se iniciam todas as atividades do desenvolvimento de *software*. Os requisitos de um sistema podem ser definidos e divididos em funcionais e não funcionais. Neste estudo os requisitos foram definidos como requisitos funcionais, que descreve as principais funcionalidades do sistema. No sistema *web*, que serviu como referência para o desenvolvimento deste trabalho, as principais funcionalidades identificadas foram: criar atividades, gerar relatórios, confirmar frequência, cadastrar pessoas, cadastrar o login de acesso das pessoas, cadastrar atividades, cadastrar as categorias das atividades, cadastrar a atuação dos usuários nas atividades e cadastrar os níveis de atuação de cada usuário nas atividades. Na criação de atividades também será possível trocar o usuário responsável pela atividade além de filtrar os usuários.

A modelagem de *software*, segundo Guedes (2009) tem o objetivo de auxiliar os engenheiros de *software* a definirem características do sistema, como os requisitos, a es-

estrutura lógica e as necessidades físicas em relação ao equipamento no qual deverá ser implementado.

Conforme Guedes (2018), o diagrama de classes é um dos mais importantes e utilizados da *UML* ou *Linguagem de Modelagem Unificada*. Além disso, segundo Guedes (2009), o diagrama de classes tem como objetivo definir a estrutura das classes utilizadas, além de estabelecer como as classes se relacionam e trocam informação entre si. O diagrama de classes foi utilizado neste estudo para definir as classes e o relacionamento entre elas.

O sistema para controle de horários e atividades desenvolvido neste estudo, possui diferenciação de níveis de usuários, que são: "Administrador", "Técnico" e "Usuário", dependendo do nível de usuários, é restringido o uso de algumas funções do sistema. O nível "Administrador" pode realizar todas as funções disponíveis no sistema. O nível "Técnico", possui funções limitadas dentro do sistema, não sendo possível, por exemplo, criar novos usuários ou editá-los mas pode imprimir dados de outras pessoas cadastradas e realizar buscas. E o nível "Usuário", também possui funções limitadas dentro do sistema, não sendo possível, por exemplo, criar novos usuários ou editar e imprimir dados de outras pessoas cadastradas, mas pode realizar buscas nas páginas de consulta. Na página inicial, o sistema possui três menus principais, que são: menu de cadastros, relatórios e de criar atividades e confirmar frequência.

O sistema é composto por três menus principais, o primeiro é o menu de cadastros, em seguida o menu de relatórios e por fim o menu de criar atividade e confirmar frequência. No menu de cadastro contém as opções de cadastro de login de pessoas, cadastrar pessoas, cadastro de atividades, cadastrar categorias, cadastrar atuações e cadastro de nível de atuação. No menu de relatório contém as opções de relatório de pessoas, relatório de frequência, relatório de categorias, relatório de atuações, relatório de níveis de atuação, relatórios de atividades cadastradas. E no menu de criar atividades e confirmar frequência, que o próprio nome do menu define suas opções.

Para desenvolver o sistema foram utilizados os *Frameworks Bulma* e *Django*. O sistema gerenciador de banco de dados (*SGBD*) escolhido para a aplicação foi o *PostgreSQL*.

2 FRAMEWORKS

Um *Framework* é composto por um conjunto de classes e pode ser definido como um sistema semi-completo e reutilizável que pode ser instanciado por um desenvolvedor de aplicações. São códigos prontos que são utilizados para solucionar problemas bem específicos, portanto, é considerada uma estrutura de trabalho que atua com funções pré-estabelecidas que serve para otimizar resultados. Sendo assim, o *Framework*, é um padrão que pode ser implementado a sistemas para intensificar a codificação de algumas partes dos projetos (CAMARGO; MASIERO, 2005).

Apesar de os conceitos serem semelhantes aos de uma biblioteca, ou seja, com códigos prontos para serem aplicados, os *Frameworks* podem ser compreendidos como um agrupamento de bibliotecas, que possuem uma estrutura ainda maior e mais robusta permitindo assim configurar partes maiores dos códigos (TRYBE, 2020).

Os *Framework* são utilizados para tarefas que podem ser consideradas repetitivas tornando-as em algo mais simples, como por exemplo, formulários de login, onde as ações que envolvem a construção destes formulários, raramente mudam dependendo do contexto da aplicação. Além disso, podem ser implementados em outras aplicações, facilitando o trabalho dos programadores (TRYBE, 2020).

2.1 FRAMEWORKS DE FRONT-END

Os *Frameworks* de *front-end* são a parte que os usuários conseguem ver e interagir em um sistema (DEVMEDIA, 2009). Alguns exemplos de *Frameworks* utilizados em desenvolvimento *web* no *front-end* são: *Bootstrap*, *Materialize*, *Vue* e o *Bulma CSS*.

Com o *Bootstrap*, pode-se projetar e personalizar rapidamente sites responsivos que são voltados para dispositivos móveis. Além disso, o *Bootstrap* é uma ferramenta de código aberto, com arquivos *SASS*, este possui um sistema de *grid layout* responsivo e também possui outros componentes pré construídos, além de *plugins JavaScript*. A versão mais atual do *Bootstrap* é a v5.2.2 (BOOTSTRAP, 2022).

O *Materialize* é um *Framework* que foi criado e projetado pelo *Google*, é uma ferramenta de design que combina designs clássicos com inovação e tecnologia. O *Materialize* possui duas formas de ser utilizado, o *Materialize*, que é a versão padrão que vem junto com os arquivos *CSS* e *Javascript*, esta opção requer pouca ou nenhuma configuração adicional. Outra forma é o *SASS*, esta versão contém os arquivos *SCSS* de origem. Ao escolher esta versão se tem mais controle sobre os componentes que foram escolhidos, e além disso para utilizá-lo será necessário o uso de um compilador *SASS* (MATERIALIZE, 2022).

O *Vue* possui uma estrutura acessível, de alto desempenho e versátil para a construção de interfaces de usuários da web. É acessível, pois se baseia em *HTML*, *CSS* e *JavaScript* padrão com *API* intuitiva e documentação mundial. Além disso, possui um sistema de renderização verdadeiramente reativo e otimizado para compiladores e que raramente requerem otimização manual. É versátil, possui um ecossistema rico e adotável de forma incremental que escala entre uma biblioteca e uma estrutura completa (VUE.JS, 2022).

E o *Bulma CSS* que é uma estrutura gratuita e de código aberto que fornece componentes *front-end* prontos para uso e que podem ser utilizados para criar interfaces modernas, responsivas e modulares (BULMA, 2022).

2.2 FRAMEWORKS DE BACK-END

Os *Framework* de *back-end* são a parte que os usuários não conseguem ver, porém é a camada principal do sistema. O *back-end* é responsável em processar os dados e executar as ações que o sistema se propõe a realizar (DEV MEDIA, 2009). Alguns exemplos de *Frameworks* utilizados em desenvolvimento *web* no *back-end* são: *Django*, *Laravel*, *Ruby* e o *Flask*.

O *Django* é um *Framework* de código aberto baseado na linguagem de programação *Python* (DJANGO, 2022a). O *Django* é utilizado para desenvolver aplicações *web* de forma fácil e rápida. Além disso, as funcionalidades do *Django* fazem desde a conexão com o banco de dados, até gerar automaticamente uma interface administrativa (DJANGO, 2022e).

O *Laravel* é uma estrutura da *Web PHP* de código aberto. Este oferece um sistema de empacotamento modular equipado com um gerenciador de dependências dedicado. O *Laravel* fornece a seus usuários várias maneiras de acessar bancos de dados relacionais, juntamente com manutenção de aplicativos e utilitários de implementação. Além disso, possui uma licença *MIT* e um código-fonte hospedado no GitHub (LARAVEL, 2022).

O *Ruby*, também conhecido como *Rails*, é uma estrutura de aplicativo *Web*, que possui uma licença *MIT*. Além disso, possui arquitetura *MVC* que é um das arquiteturas de aplicativos da *Web* mais amplamente utilizadas globalmente, esta arquitetura separa os códigos por suas funções, ou seja, camada de dados, camada de apresentação e manutenção de uma camada de recursos. O *Ruby* depende de uma biblioteca conhecida como *Active Record* ou registro ativo, que permite que os desenvolvedores realizem consulta de interação de banco de dados (RUBY, 2022).

E o *Flask* é uma microestrutura da *Web* baseada em *Python* que não requer bibliotecas ou qualquer tipo de ferramentas específicas. Este *Framework* não possui a validação de formulário. O *Flask* fornece suporte para extensões que podem ser adicionadas de uma

forma que irá parecer que foram nativamente implementados no *Flask* (FLASK, 2022).

2.3 FRAMEWORK BULMA CSS

O *Bulma* é um *Framework CSS* de código aberto e gratuito (BULMA, 2022). Foi criado por *Jeremy Thomas*, publicado no ano de 2016 e distribuído sob licença do *MIT* (ISOLUTION, 2022).

O *Bulma* fornece componentes de *front-end* prontos para serem utilizados, sendo assim, pode-se facilmente criar interfaces mais modernas, responsivas e modulares, sendo possível implementar na interface apenas o que for necessário utilizar, além de ser uma interface de fácil usabilidade para os usuários. O *Bulma CSS* utiliza o mínimo de código *HTML* para facilitar a sua leitura e escrita. Para a utilização deste *Framework* não é necessário nenhum conhecimento de *CSS* (BULMA, 2022).

A versão v0.9.4 do *Bulma CSS* foi a versão utilizada no processo de desenvolvimento da aplicação que se desenvolveu com este estudo.

Existem três formas de realizar a instalação do *Bulma CSS* que podem ser vistas na URL <https://bulma.io/>. A forma escolhida para realizar a instalação para este estudo foi por *CDN Link*, sendo uma das formas mais recomendadas a se utilizar. O *CDN Link* deve ser adicionado ao *Head* do Cabeçalho do documento *HTML*. Este processo pode ser visto na Figura 1.

Figura 1 – Uso do *CDN Link* no *Head* do cabeçalho *HTML*.

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css">
  <script defer src="https://use.fontawesome.com/releases/v5.14.0/js/all.js"></script>
  <title>Página Inicial</title>
</head>
```

Fonte: A autora(2022)

O *Bulma* é uma biblioteca *CSS* que possui um conjunto de elementos e componentes destinados para o desenvolvimento de páginas *web* (*website*). O desenvolvimento de um *website* utilizando o *Bulma CSS* pode ser composto por elementos e componentes (BULMA, 2022).

Alguns elementos e componentes do *Framework Bulma CSS* não foram utilizados ao longo do desenvolvimento deste estudo, estes podem ser visto na URL <https://bulma.io/>. Serão mostrados e comentados nesta seção apenas os elementos e componentes que

foram utilizados para compor as interfaces desenvolvidas neste estudo.

2.3.1 Sistema Grid

Columns é uma maneira simples de construir colunas responsivas, com elas, pode-se definir tamanhos diferentes para cada coluna, definindo um tamanho único para cada uma delas individualmente e também lidar com as diferentes posições das mesmas. Dessa forma, pode-se projetar diferentes tipos de layouts.

As colunas no *Bulma CSS* são definidas com um elemento `<div>` utilizando a classe *columns*. Então, dentro deste elemento *columns* principal, é necessário definir colunas com a classe *column* (BULMA, 2021). A estrutura de colunas desenvolvida neste estudo pode ser vista na Figura 2.

Figura 2 – Estrutura de colunas

```
<div class="columns is-flex is-justify-content-center is-align-items-center">
  <div class="column">
    <a class="has-text-black is-uppercase" href="{% url 'index' %}">
      <p>Laboratório de Pesquisa da UFSM</p>
    </a>
  </div>
  <div class="column">...
```

Fonte: A autora(2022)

O *columns* foi utilizado neste estudo pra posicionar lado a lado o título da página e os itens do menu, como pode ser visto na Figura 3, "Laboratório de Pesquisa da UFSM" esta dentro de uma classe `<column>` e "Início" e o "Nome de usuário" estão em outra classe `<column>`, sendo assim têm-se um tamanho diferente definido para cada *column* utilizado, e as duas colunas estão dentro de uma classe `<columns>`.

Figura 3 – As colunas na página inicial do sistema.



Fonte: A autora(2022)

2.3.2 Layout

Existem oito tipos diferentes de *Layouts* para serem utilizados, os *Layouts* servem para definir a estrutura de uma página *web*, e para isso utiliza-se uma classe *CSS* (BULMA, 2022). Os *Layouts* utilizados para o desenvolvimento deste estudo foram: *section*, *container* e *footer*.

A *section* é na realidade um *container* porém mais simples, este é utilizado para dividir a página em uma ou várias seções (BULMA, 2021). Além disso, as *sections* no *Bulma CSS* podem ser definidas por uma *<class>* dentro de uma *<section>*. Uma das estruturas do *<section>* criadas para este estudo pode ser vista na Figura 4.

Figura 4 – Estruturas do *section*.

```

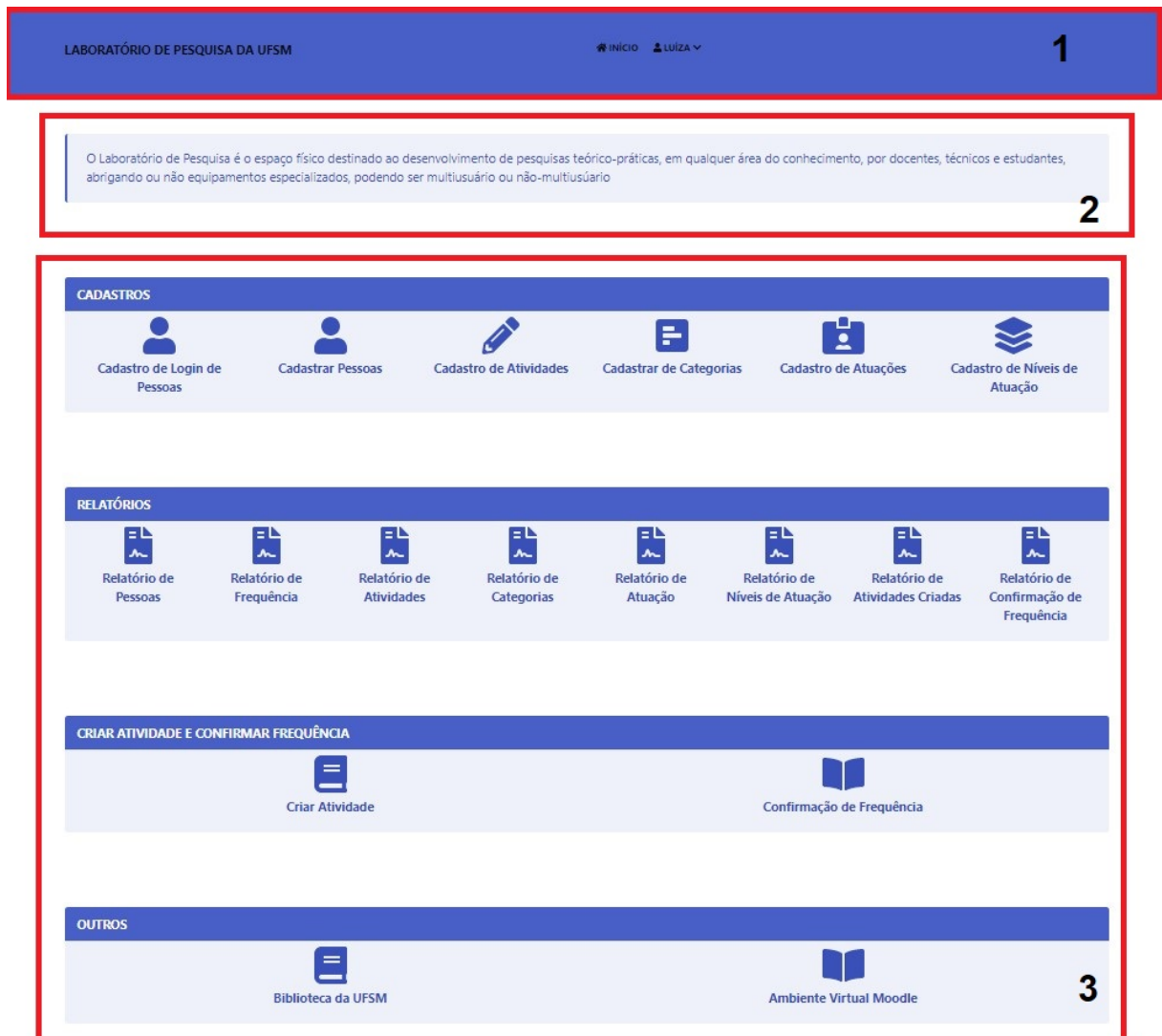
40 <section class="section">
41   <div class="container">
42     <div class="columns is-flex is-justify-content-center is-align-items-center ">
43       <div class="column">...
44     </div>
45   </div>
46   <div class="column">...
47 </div>
73 </div>
74 </div>
75 </section>

```

Fonte: A autora(2022)

O *container section* foi utilizado no desenvolvimento deste estudo para delimitar o corpo do sistema em três seções diferentes. As três seções podem ser vistas na Figura 5.

Figura 5 – *sections* na página inicial do sistema.



Fonte: A autora(2022)

O *Container* é uma classe da *Bulma CSS* que possui regras *CSS* com margens laterais e está alinhado com a responsividade através dos *breakpoints* definidos pelo *Bulma*. Este elemento é utilizado dentro de uma classe `<div>` (BULMA, 2021). A estrutura do *container* criada para este estudo pode ser vista na Figura 6.

Figura 6 – Estruturas do *container*.

```

40 <section class="section">
41   <div class="container">
42 >     <div class="columns is-flex is-justify-content-center is-align-items-center">...
74   </div>
75 </section>

```

Fonte: A autora(2022)

A classe `<container>` foi utilizada no desenvolvimento deste estudo para centralizar tudo que foi inserido dentro da *section*.

O *Footer*, é um rodapé responsivo, nele podem ser incluídas listas, cabeçalhos, colunas, ícones e botões (BULMA, 2021). O footer é definido dentro de um `<footer>`. A estrutura do *footer* criada para este estudo pode ser vista na Figura 7.

Figura 7 – Estrutura do *footer*.

```

197 <footer class="section has-background-link">
198   <div class="container">
199     <div class="column has-text-black has-text-weight-bold is-flex is-justify-content-center">
200       Copyright &copy; - Laboratório de Pesquisa da UFSM
201     </div>
202   </div>
203 </footer>

```

Fonte: A autora(2022)

O *footer* foi utilizado no desenvolvimento deste estudo para definir o rodapé do sistema, sendo ele responsivo. E para centralizar os itens dentro dele foi utilizada uma classe `<container>`.

2.3.3 Components

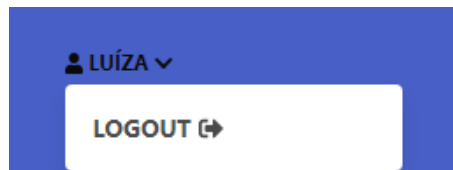
Os componentes são mais avançados e podem ser utilizados de várias formas. Além disso, existem dois tipos de componentes, os componentes de navegação e de mídia (BULMA, 2021).

2.3.3.1 Componentes de Navegação

Existem cinco tipos diferentes de componentes de navegação que podem ser utilizados, o componente de navegação utilizados para o desenvolvimento deste estudo foi o *Dropdown*.

O *Dropdown* é um componente que na realidade serve como um *container* para um botão suspenso que conterà dentro dele um menu suspenso (BULMA, 2021). Este processo pode ser visto na Figura 8, onde foi criando um *dropdown* com a opção de *logout*, para que o usuário possa clicar neste botão suspenso e sair da página em que esta logado.

Figura 8 – Menu *dropdown*.



Fonte: A autora(2022)

2.3.3.2 Componentes de Mídia

Existem cinco tipos diferentes de componentes de mídia que podem ser utilizados. O componente de mídia utilizados para o desenvolvimento deste estudo foi o *Message*.

Message é um bloco de mensagens colorido, o *message* é o local onde se exibe mensagem, o *message-header* pode conter um título e algum outro elemento e o *message-body* que é para o corpo mais longo da mensagem (BULMA, 2021). O *message-body* pode ser visto na Figura 9, que mostra uma mensagem informando o que é um laboratório de pesquisa.

Figura 9 – Bloco *message*.

O Laboratório de Pesquisa é o espaço físico destinado ao desenvolvimento de pesquisas teórico-práticas, em qualquer área do conhecimento, por docentes, técnicos e estudantes, abrigando ou não equipamentos especializados, podendo ser multiusuário ou não-multiusuário

Fonte: A autora(2022)

2.3.4 Forms

Forms são utilizados em formulários para se obter maior controle e clareza sobre as informações inseridas no mesmo (BULMA, 2021). Existem seis tipos diferentes de atributos que podem ser utilizados dentro do *forms*, para o desenvolvimento deste estudo foram utilizados os atributos: *Inputs*, *Textarea*, *Select* e o *File*.

Inputs são as entradas de texto, os tipos de entrada são *text*, *password*, *email* e *number* (BULMA, 2021).

Figura 10 – *inputs* no formulário de cadastro de usuários.

Nome:
João Siva

Email:
nome@gmail.com

Telefone:
(55) 99999-9999

Matrícula:
202200000009

Fonte: A autora(2022)

Textarea é um espaço onde o usuário poderá escrever um texto mais longo contendo as suas considerações (BULMA, 2021).

Figura 11 – *Textarea* dentro do sistema desenvolvido.

Texto:

Fonte: A autora(2022)

O *Select* é uma lista de seleção que dá mais estilo e flexibilidade (BULMA, 2021).

Figura 12 – apresenta o *select* dentro do sistema desenvolvido.

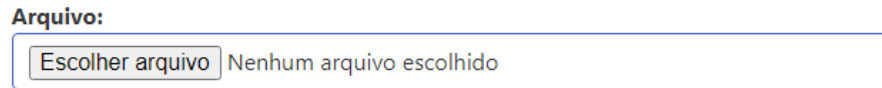
Situação:

- Em Andamento
- Concluído
- Encerrado
- Cancelado
- Em Andamento

Fonte: A autora(2022)

File permite o *upload* de arquivos personalizados sem a utilização do *Javascript* (BULMA, 2021).

Figura 13 – apresenta o *input* do tipo *file* dentro do sistema desenvolvido.



Fonte: A autora(2022)

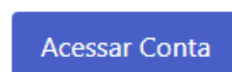
Os *forms* foram utilizados no desenvolvimento deste estudo para criar os formulários de cadastro de pessoas, cadastro de logins de pessoas, cadastro de atividades, cadastro de categorias, cadastro de atuações e cadastro de níveis de atuação.

2.3.5 Elements

Os *elements* são fundamentais e necessitam de apenas uma classe *CSS* para serem utilizados (BULMA, 2021). Existem doze tipos diferentes de *elements* que podem ser utilizados. Os *elements* utilizados para o desenvolvimento deste estudo são: *Buttons*, *Content*, *Icon* e o *Table*.

O *Button* é um botão clássico que possui diferentes cores, tamanhos e estados (BULMA, 2021).

Figura 14 – Botão do sistema desenvolvido.



Fonte: A autora(2022)

Os *buttons* tem dentro do sistema desenvolvido várias funções que vão desde acessar a página inicial até cadastrar e apagar pessoas e outras coisas que estão cadastradas no sistema.

Content pode ser usado a qualquer momento para definir parágrafos, lista, títulos, citações e tabelas (BULMA, 2021).

Figura 15 – *content list(ul e li)*.

```
<div class="column">
  <ul style="font-size: 12px;">
    <li class="is-inline-block mr-4 is-uppercase">
      <div class="dropdown is-hoverable"...>
    </li>
  </ul>
</div>
```

Fonte: A autora(2022)

O *content list(ul e li)* foi utilizado na criação do *dropdown* onde na sua criação é necessário criar um lista não ordenada para alinhar os itens que ficarão suspensos dentro do menu suspenso do *dropdown*.

O *Icon* é um elemento que serve como um *container*, e é utilizado para controlar o espaço que os ícones ocuparão, ou seja, evitará que a página pule o ícone no carregamento. Os *icons* vem das bibliotecas *Font Awesome 5*, *Font Awesome 4*, *Material Design Icons*, *Ionicons*, entre outros nas quais o Bulma é compatível (BULMA, 2021).

Figura 16 – *icons* no sistema desenvolvido.



Fonte: A autora(2022)

A classe *<icons>* foi utilizada no desenvolvimento deste estudo para controlar o espaço que os ícones ocupariam dentro das tabelas de relatórios. Este processo pode ser visto na Figura 17.

Figura 17 – Classe *<icons>*.

```
<a href="{% url 'atividade.editar' RelatoriosAtividades.pk %}">
  <span class="icon">
    <i class="fas fa-edit"></i>
  </span>
</a>
```

Fonte: A autora(2022)

O *Table* é usado para criar tabelas, pode-se criá-las apenas utilizando a classe `<table>` (BULMA, 2021). As tabelas foram utilizadas no desenvolvimento deste sistema para exibir, nas páginas *HTML*, os dados das tabelas. Este processo pode ser visto na Figura 18.

Figura 18 – *table* na página de relatórios de atuação.

Atuação	Descrição	Ações
Coordenação	Descrição da atuação um	✎ 🗑
Colaborador	Descrição da atuação dois	✎ 🗑
Ouvinte	Descrição da atuação três	✎ 🗑

Fonte: A autora(2022)

Figura 19 – Classe `<table>`.

```
<table class="table is-table is-bordered is-striped is-narrow is-hoverable is-fullwidth">
  <thead>
    <tr>
      <th class="has-text-centered" style="border: 1px solid blue">Titulo da Atividade</th>
      <th class="has-text-centered" style="border: 1px solid blue">Descrição</th>
      <th class="has-text-centered" style="border: 1px solid blue">Data de Início</th>
      <th class="has-text-centered" style="border: 1px solid blue">Data de Encerramento</th>
      <th class="has-text-centered" style="border: 1px solid blue">Prioridade</th>
      <th class="has-text-centered" style="border: 1px solid blue">Categoria</th>
      {% if request.user.is_authenticated %}
        {% if request.user.groups.all.0.name == 'Administração' %}
          <th class="has-text-centered" style="width: 100px; border: 1px solid blue;">Ações</th>
          {% endif %}
        {% endif %}
    </tr>
  </thead>
  <tbody...>
</table>
```

Fonte: A autora(2022)

Após criadas todas as interfaces com o *Framework* de *front-end* *Bulma CSS*, foi dado início ao desenvolvimento da aplicação no *Framework* de *back-end* *Django*. Este processo pode ser visto na próxima seção.

2.4 FRAMEWORK DJANGO

O *Django* é um *Framework Web* gratuito, de código aberto e de alto nível, e é escrito na linguagem de programação *Python* (DJANGO, 2022a). O *Django* foi desenvolvido para otimizar tarefas do desenvolvimento *web*, além disso, as funcionalidades do *Django* fazem desde a conexão com o banco de dados, até gerar automaticamente uma interface administrativa (DJANGO, 2022b).

A área administrativa é integrada ao projeto, sendo assim, é utilizada para fazer o gerenciamento do sistema, portanto, permite controlar os dados e até mesmo definir um sistema de gerenciamento de conteúdo (DJANGO, 2022e).

O *Django* foi lançado no verão do ano de 2005 e foi desenvolvido por *Adrian Holovaty* e *Simon Willison* como um sistema para o *World Online*, que foi o responsável por construir grandes aplicações *web* em prazos curtos, eles tinham geralmente apenas algumas horas para lançar uma nova aplicação pra o público (DJANGO, 2022e).

O nome *Django* é uma homenagem ao guitarrista *Django Reinhardt*. A pronúncia do *Django* é JANG -oh, onde o “D” é mudo (DJANGO, 2022e).

A versão do *Django* 4.1, foi lançada em 3 de agosto de 2022, sendo assim, esta foi a versão utilizada no processo de desenvolvimento da aplicação que se desenvolveu com este estudo.

2.4.1 Arquitetura MTV do Django

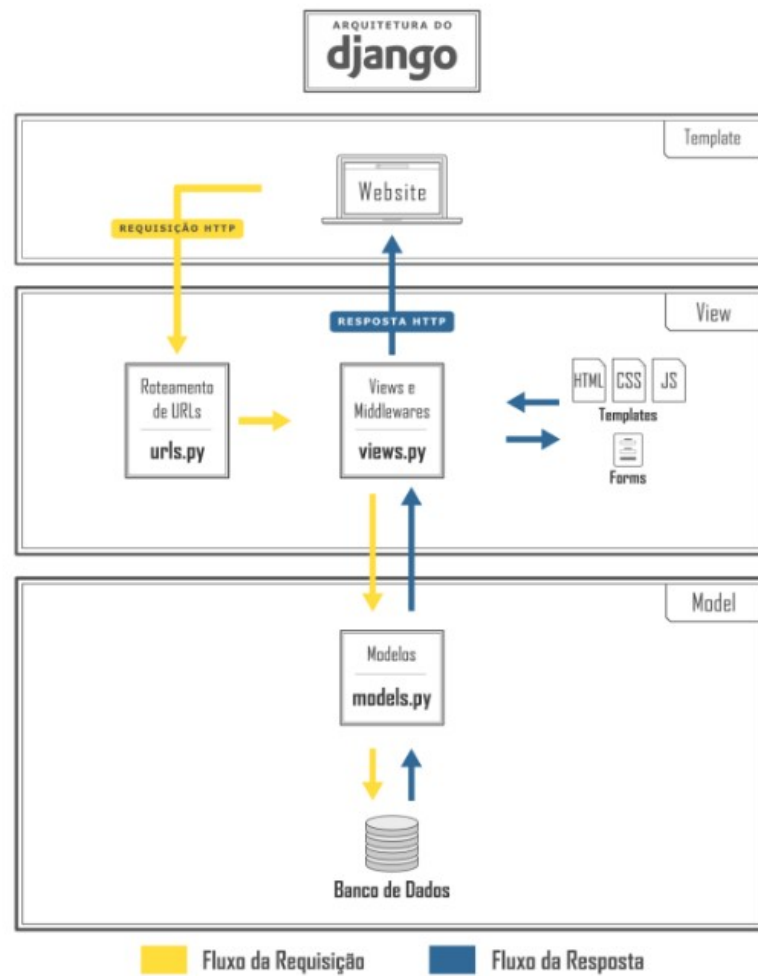
De acordo com Ramos (2018), a documentação do *Django* descreve ele como um *Framework Model, Template e View (MTV)*.

Há uma interdependência entre eles para que uma ação solicitada pelo usuário seja colocada em prática, ou seja, toda essa arquitetura é interligada e conversa entre si (DEVOPS, 2020).

Cada um tem uma função, o *View* é uma forma de processar os dados de uma URL específica, pois ela descreve qual informação foi apresentada. Os *Templates* servem para separar o conteúdo de apresentação, ou seja, descreve como uma informação é apresentada. E o *Model*, é responsável pela estruturação do modelo de dados que será gerenciado pela aplicação (RAMOS, 2018).

Segundo Ramos (2018), uma requisição feita pelo usuário sai do *browser* do usuário, passa para o servidor onde o *Django* está sendo executado e retorna ao *browser* do usuário. Este processo pode ser visto no fluxo de requisição da Figura 20.

Figura 20 – Fluxo de requisição.



Fonte: (RAMOS, 2018)

As informações inseridas nas *Views* ficam em um arquivo dentro da aplicação chamado *views.py*, onde se encontra o código que processa as requisições, formula respostas e as envia de volta ao usuário (RAMOS, 2018).

Figura 21 – Utilização do arquivo *views.py*.

```
from django.urls import reverse_lazy
from django.views.generic import ListView
from django.views.generic.edit import CreateView, UpdateView,
DeleteView
from home.utils import HtmlToPdf

from .forms import AtividadeModelForm
from .models import Atividade

class AtividadeView(ListView):
    model = Atividade
    template_name = 'RelatoriosAtividades.html'

    def get_queryset(self, *args, **kwargs):
        buscar = self.request.GET.get('buscar')
        qs = super(AtividadeView, self).get_queryset(*args, **kwargs
)

        if buscar:
            qs = qs.filter(titulo__icontains=buscar)

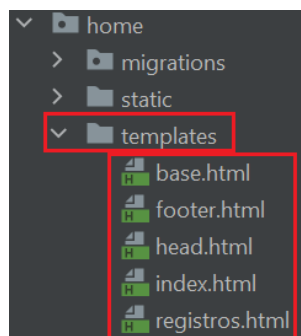
        return qs
```

Fonte: A autora(2022)

Alguns fatores importantes devem ser considerados na hora da implementação dos *templates*, o primeiro fator, é que deve ser feita a separação lógica da visualização do código e o segundo fator é que deve-se evitar a repetição de código que pode ser solucionado com o processo de herança de outros *templates* (LABORATORIO, 2021).

Os arquivos inseridos no *Template* ficam em uma pasta dentro da aplicação chamada de *templates*. Para cada aplicação que foi criada uma pasta *templates*, e dentro dela ficam todos os arquivos *HTML*.

Figura 22 – Pasta *Templates* e a separação do código.



Fonte: A autora(2022)

As informações inseridas no *Model* ficam em um arquivo dentro da aplicação chamado *models.py*, onde apresenta o código que representa a estruturação do modelo de dados. Este processo pode ser visto na Figura 23.

Figura 23 – Arquivo *models.py*.

```
from django.db import models

class Atuacao(models.Model):
    ATUACAO_OPCOES = (
        ('C1', 'Coordenador'),
        ('C2', 'Colaborador'),
        ('0', 'Ouvinte')
    )

    descricao = models.CharField('Descricao', max_length=50,
    help_text='Informe_a_descricao')
    atuacao = models.CharField('Atuacao_do_Usuario', max_length
    =15, help_text='Atuacao_do_Usuario_na_Atividade', choices=
    ATUACAO_OPCOES, default='C1')

    class Meta:
        verbose_name = 'Atuacao'
        verbose_name_plural = 'Atuacoes'

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

Fonte: A autora(2022)

2.4.2 Comando pip

O *pip* (*Python Package Index*) ou Índice de Pacotes *Python*, é uma ferramenta de gerenciamento de pacotes, e é um facilitador para a realização de instalações, remoção e atualizações de pacotes externos em projetos (PIP, 2022).

Para realizar a instalação do *Django*, que foi utilizado para desenvolver este estudo, foi necessário utilizar a ferramenta *pip*. Para realizar a instalação utilizou-se o *pip* a partir da linha de comando *pip install django* (DJANGO, 2022e).

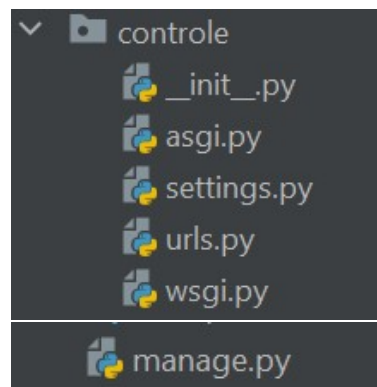
No projeto desenvolvido neste estudo, foi utilizado o comando *pip freeze*, este lista todas as versões de pacotes instalados e suas dependências, após listar todas essas versões, utilizou-se o comando *pip freeze > requirements.tx* para exportar a lista de pacotes instalados para o arquivo *requirements.txt*, este arquivo irá fixar as versões de pacotes e suas dependências protegendo as aplicações que forem criadas de *bugs* ou incompatibilidades em versões recém-lançadas (BACK, 2020).

2.4.3 Aplicações no Django

Para criar uma aplicação, primeiro é necessário criar um projeto, o comando utilizado para criar um projeto foi o *django-admin startproject controle* . .

Ao criar o projeto, cria-se de forma automática a estrutura de diretórios correta para dar início ao desenvolvimento das aplicações. Esta estrutura de diretórios pode ser vista na Figura 24.

Figura 24 – Estrutura de diretórios padrão.

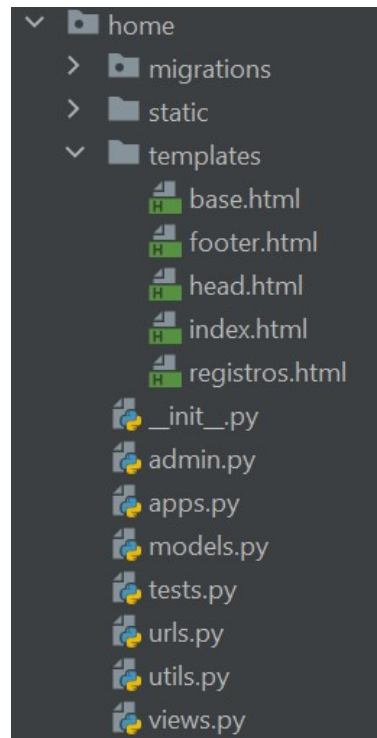


O arquivo *settings.py* é importante, pois nele contém as configurações do nosso projeto, como as configurações de banco de dados, aplicativos instalados, configurações de arquivos estáticos entre outros (RAMOS, 2018).

No projeto desenvolvido neste estudo foram criadas onze aplicações. As aplicações foram criadas com o comando *django-admin startapp*. Segundo Ramos (2018), em um projeto podem ter várias aplicações e uma aplicação pode estar presente em diversos projetos.

A aplicação *home* que foi criada com o comando *django-admin startapp home*, é considerada a aplicação principal, nesta aplicação ficam os diretórios que são acessados pelas outras aplicações, tudo que for de uso genérico para a aplicação, ficará nesta aplicação. As pasta *templates* e *static* ficam nesta pasta. Na pasta *templates* da aplicação principal, é feita a divisão do código HTML em *base*, *head*, *footer*, *index* e *registros*, e a pasta *static* também fica na aplicação principal, esta serve para guardar os arquivos estáticos, como por exemplo os arquivos *CSS*, *Javascript* e imagens. A estrutura de diretórios da aplicação principal pode ser vista na Figura 25.

Figura 25 – Estrutura da aplicação *home*.

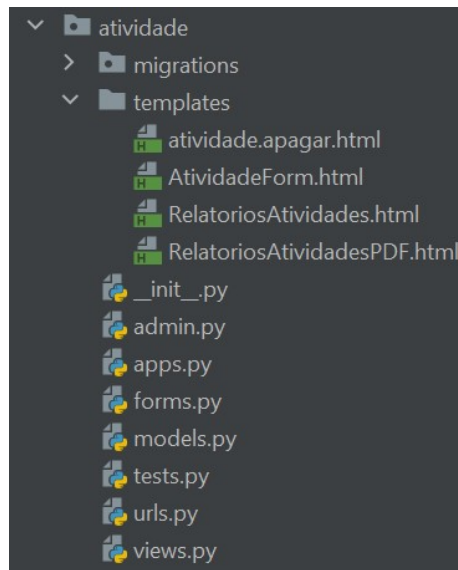


Fonte: A autora(2022)

As aplicações não foram criadas todas de uma vez, elas foram criadas uma de cada vez e implementadas com código. Para a criação de cada uma das aplicações foi utilizado o comando *django-admin startapp* e o nome da aplicação, os nomes das aplicações criadas são *atividade*, *pessoa*, *atuacao*, *nivel*, *categoria*, *criaatividade*, *frequenciaatividade*, *cadastro* e *login*.

Com a criação de uma aplicação, o *Django* cria uma estrutura de diretórios dentro da aplicação, os principais arquivos dessa estrutura são *admin.py*, *apps.py*, *models.py*, *views.py*, *urls.py* e *forms.py*. Na Figura 26 pode ser vista a estrutura de diretórios da aplicação *atividade*, que é uma das aplicações criadas no desenvolvimento deste estudo.

Figura 26 – Estrutura de diretórios da aplicação *atividade*.



Fonte: A autora(2022)

O *ModelAdmin*, serve para alterar como um *model* é exibido na interface de administração do *Django*. É definida uma classe *ModelAdmin* que descreve o *layout* e registra-o no *model* (MOZILLA, 2022a).

Primeiramente é criada uma classe com a lista de campos de banco de dados definida pelo *model* (DJANGO, 2022c). Estes campos são especificados por atributos de uma classe. A Figura 27 mostra a classe *Pessoa* e seus atributos.

Figura 27 – Classe *Pessoa*.

```
class Pessoa(models.Model):
    PERFIL_OPCOES = (
        ('A', 'Administrador'),
        ('U', 'Usuário'),
        ('T', 'Técnico')
    )
    nome = models.CharField('Nome', max_length=35, help_text='Informe o nome completo')
    email = models.EmailField('Email', max_length=45, help_text='Informe o email', unique=True)
    telefone = models.CharField('Telefone', max_length=14, help_text='Informe o telefone')
    matricula = models.CharField('Matricula', max_length=20, help_text='Informe a matricula', unique=True)
    perfil = models.CharField('Perfil', max_length=15, help_text='Perfil do Usuário', choices=PERFIL_OPCOES, default='U')

    class Meta:
        verbose_name = 'Pessoa'
        verbose_name_plural = 'Pessoas'

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

Fonte: A autora(2022)

Após criar a classe, as informações são armazenadas no arquivo *admin.py* na aplicação da classe criada (DJANGO, 2022d). A Figura 28 mostra o arquivo *admin.py* da

classe pessoa.

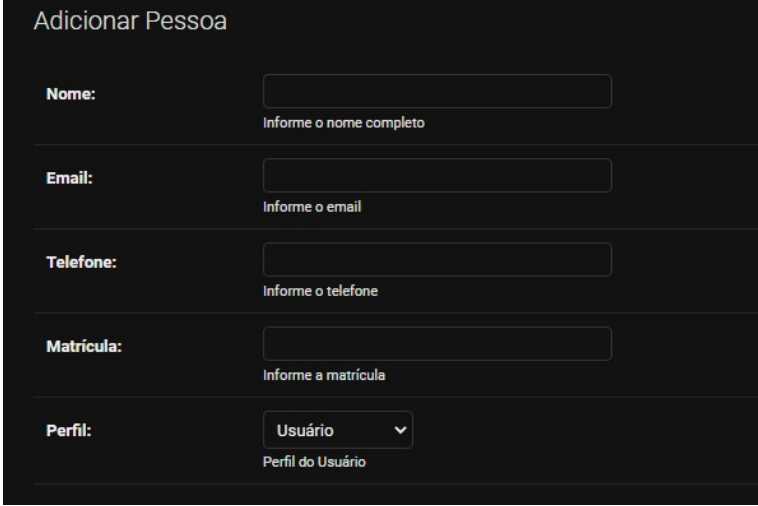
Figura 28 – Arquivo *admin.py*.

```
@admin.register(Pessoa)
class PessoaAdmin(admin.ModelAdmin):
    list_display = ('nome', 'email', 'telefone', 'matricula', 'perfil')
    search_fields = ('nome', 'matricula', 'perfil')
```

Fonte: A autora(2022)

O *list_display* é utilizado para adicionar os inputs na página administrativa do *Django* (MOZILLA, 2022a). A Figura 29, mostra como as informações inseridas no *list_display* na página administrativa do *Django*.

Figura 29 – Informações inseridas na página administrativa do *Django* pelo *list_display*.



Adicionar Pessoa

Nome:
Informe o nome completo

Email:
Informe o email

Telefone:
Informe o telefone

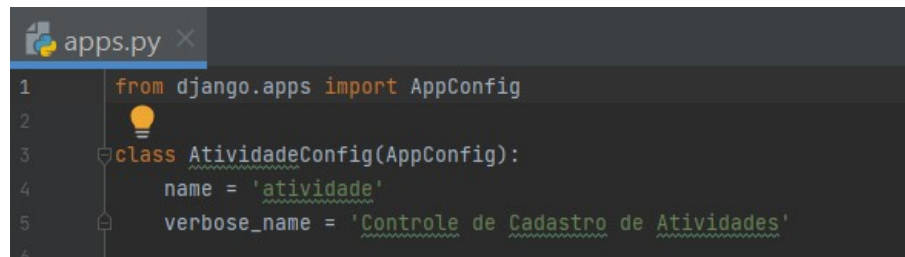
Matricula:
Informe a matrícula

Perfil:
Perfil do Usuário

Fonte: A autora(2022)

O arquivo *apps.py* é responsável pela configuração da aplicação do projeto *Django* (TREINAWEB, 2020b). A Figura 30 mostra o arquivo *apps.py* da aplicação *atividade* que é uma das aplicações criada no desenvolvimento deste estudo. O atributo *verbose_name*, presente na classe *AtividadeConfig*, armazena o texto que será exibido na interface administrativa do *Django*, permitindo uma personalização do módulo administrativo. E a Figura 31 mostra como o texto é exibido na página administrativa do *Django*.

Figura 30 – Apresenta o arquivo *apps.py*.



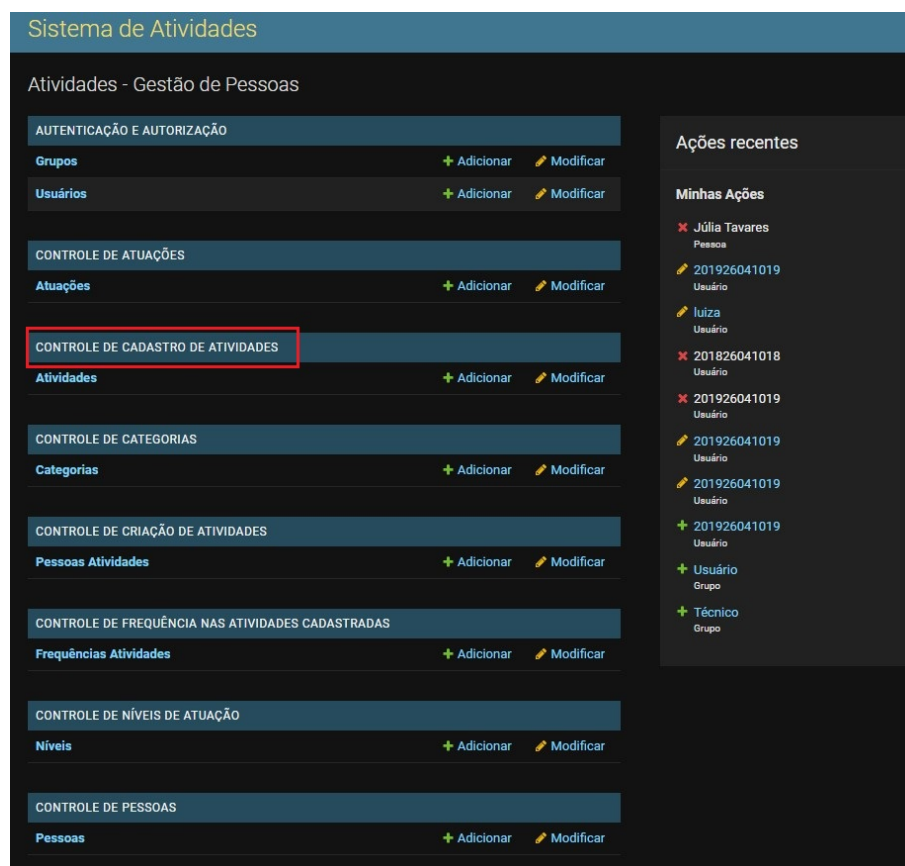
```

1  from django.apps import AppConfig
2
3  class AtividadeConfig(AppConfig):
4      name = 'atividade'
5      verbose_name = 'Controle de Cadastro de Atividades'

```

Fonte: A autora(2022)

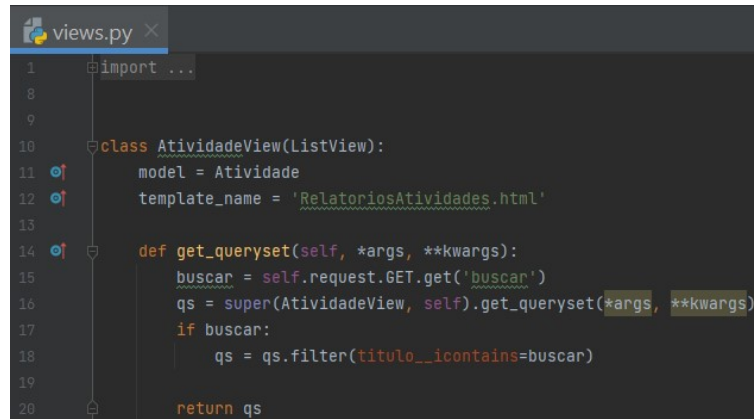
Figura 31 – Apresenta o arquivo *apps.py*.



Fonte: A autora(2022)

O arquivo *views.py* é responsável por processar os dados de uma URL específica, pois ela descreve qual informação foi apresentada (RAMOS, 2018). A Figura 32 mostra o arquivo *views.py* da aplicação *atividade* que é uma das aplicações criadas no desenvolvimento deste estudo.

Figura 32 – Apresenta o arquivo *views.py*.



```

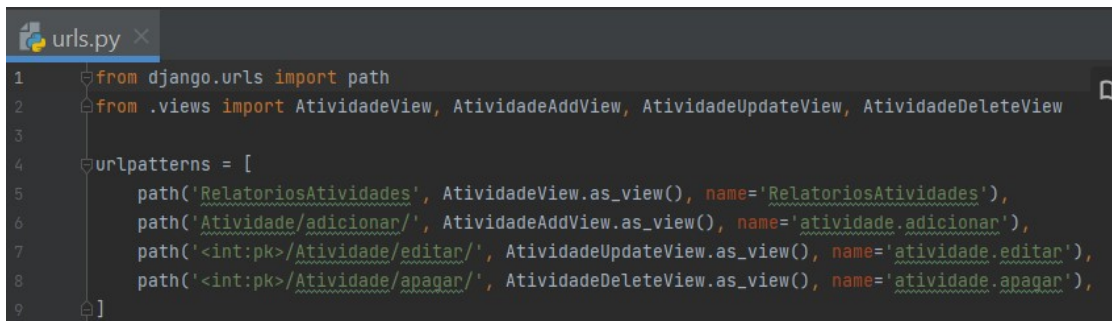
1  import ...
8
9
10 class AtividadeView(ListView):
11     model = Atividade
12     template_name = 'RelatoriosAtividades.html'
13
14     def get_queryset(self, *args, **kwargs):
15         buscar = self.request.GET.get('buscar')
16         qs = super(AtividadeView, self).get_queryset(*args, **kwargs)
17         if buscar:
18             qs = qs.filter(titulo__icontains=buscar)
19
20     return qs

```

Fonte: A autora(2022)

O arquivo *urls.py* é responsável por definir a rota para as *urls* das diversas aplicações que podem estar relacionadas em um projeto. A Figura 33 mostra o arquivo *apps.py* da aplicação *atividade* que é uma das aplicações criadas no desenvolvimento deste estudo.

Figura 33 – Apresenta o arquivo *urls.py*.



```

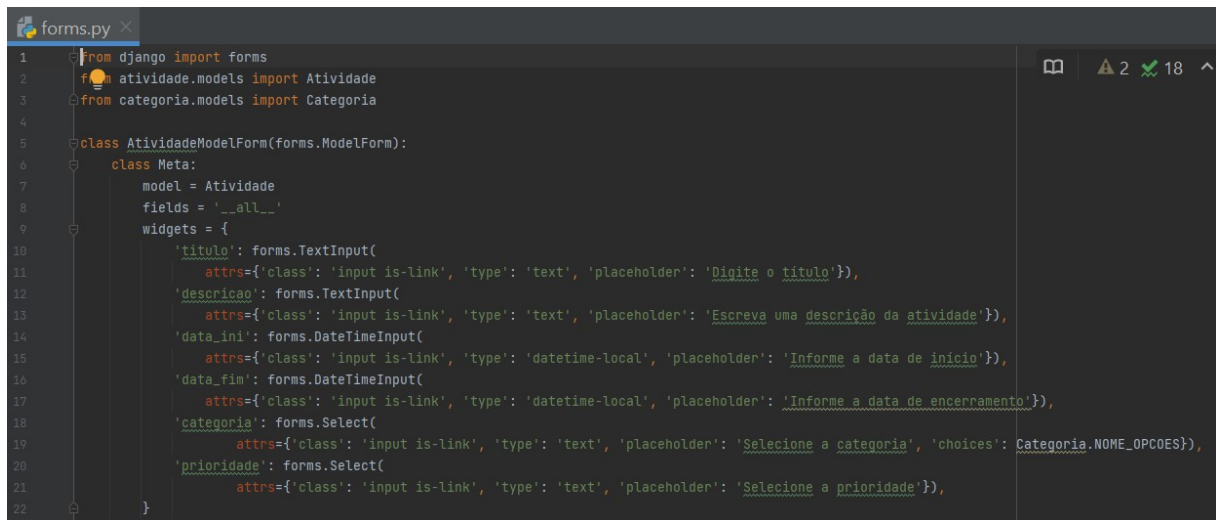
1  from django.urls import path
2  from .views import AtividadeView, AtividadeAddView, AtividadeUpdateView, AtividadeDeleteView
3
4  urlpatterns = [
5      path('RelatoriosAtividades', AtividadeView.as_view(), name='RelatoriosAtividades'),
6      path('Atividade/adicionar/', AtividadeAddView.as_view(), name='atividade.adicionar'),
7      path('<int:pk>/Atividade/editar/', AtividadeUpdateView.as_view(), name='atividade.editar'),
8      path('<int:pk>/Atividade/apagar/', AtividadeDeleteView.as_view(), name='atividade.apagar'),
9  ]

```

Fonte: A autora(2022)

A classe *form* no *Django*, segundo Mozilla (2022b), especifica os campos no formulário entre outros elementos. Os dados do formulário são armazenados no arquivo *forms.py* de cada aplicação que for criada. A Figura 34 mostra o arquivo *forms.py* da aplicação *atividade* que é uma das aplicações criadas no desenvolvimento deste estudo.

Figura 34 – Apresenta o arquivo *forms.py*.



```

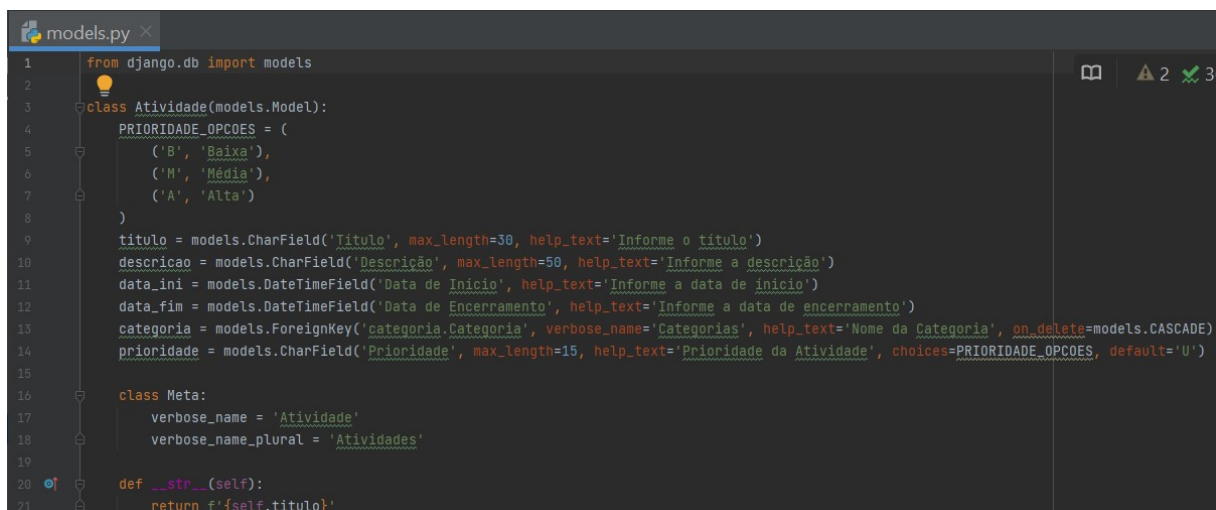
1  from django import forms
2  from atividade.models import Atividade
3  from categoria.models import Categoria
4
5  class AtividadeModelForm(forms.ModelForm):
6      class Meta:
7          model = Atividade
8          fields = '__all__'
9          widgets = {
10             'titulo': forms.TextInput(
11                 attrs={'class': 'input is-link', 'type': 'text', 'placeholder': 'Digite o título'}),
12             'descricao': forms.TextInput(
13                 attrs={'class': 'input is-link', 'type': 'text', 'placeholder': 'Escreva uma descrição da atividade'}),
14             'data_ini': forms.DateTimeInput(
15                 attrs={'class': 'input is-link', 'type': 'datetime-local', 'placeholder': 'Informe a data de início'}),
16             'data_fim': forms.DateTimeInput(
17                 attrs={'class': 'input is-link', 'type': 'datetime-local', 'placeholder': 'Informe a data de encerramento'}),
18             'categoria': forms.Select(
19                 attrs={'class': 'input is-link', 'type': 'text', 'placeholder': 'Selecione a categoria', 'choices': Categoria.NOME OPCOES}),
20             'prioridade': forms.Select(
21                 attrs={'class': 'input is-link', 'type': 'text', 'placeholder': 'Selecione a prioridade'}),
22         }

```

Fonte: A autora(2022)

No *models.py*, é feita a definição das classes que representam os objetos que serão incorporados na aplicação (TREINAWEB, 2020b). A Figura 35 mostra o arquivo *models.py* da aplicação *atividade* que é uma das aplicações criadas no desenvolvimento deste estudo. Cada atributo da classe é definido com suas propriedades, definidas de acordo com o tipo de informação que ele armazenará.

Figura 35 – Arquivo *models.py* da classe Atividade.



```

1  from django.db import models
2
3  class Atividade(models.Model):
4      PRIORIDADE OPCOES = (
5          ('B', 'Baixa'),
6          ('M', 'Média'),
7          ('A', 'Alta')
8      )
9      titulo = models.CharField('Título', max_length=30, help_text='Informe o título')
10     descricao = models.CharField('Descrição', max_length=50, help_text='Informe a descrição')
11     data_ini = models.DateTimeField('Data de Início', help_text='Informe a data de início')
12     data_fim = models.DateTimeField('Data de Encerramento', help_text='Informe a data de encerramento')
13     categoria = models.ForeignKey('categoria.Categoria', verbose_name='Categorias', help_text='Nome da Categoria', on_delete=models.CASCADE)
14     prioridade = models.CharField('Prioridade', max_length=15, help_text='Prioridade da Atividade', choices=PRIORIDADE OPCOES, default='U')
15
16     class Meta:
17         verbose_name = 'Atividade'
18         verbose_name_plural = 'Atividades'
19
20     def __str__(self):
21         return f'{self.titulo}'

```

Fonte: A autora(2022)

Após escrever os dados da classe no arquivo *models.py*, deve-se executar primeiro o comando *python manage.py makemigrations* e logo em seguida o comando *python ma-*

nage.py migrate.

2.4.4 Migrations

Segundo Django (2022e),

Migrações é a maneira do *Django* de propagar as mudanças que você faz em seus modelos (adicionando um campo, deletando um modelo, etc.) em seu esquema de banco de dados. Eles foram projetados para serem quase todos automáticos, mas você precisará saber quando fazer migrações, quando executá-las e os problemas comuns que você pode encontrar.

2.4.4.1 O Comando Makemigrations

Este comando é responsável por criar novas migrações com base nas alterações feitas em seus modelos, as alterações feitas no arquivo *models.py* (DJANGO, 2022e). O comando utilizado foi o *python manage.py makemigrations*.

O comando *makemigrations* deve ser executado toda vez que for feita alguma alteração no arquivo *models.py* (DJANGO, 2022e).

Após executar o comando *python manage.py makemigrations*, cria-se de forma automática um diretório chamado *makemigrations* dentro da aplicação. E dentro deste diretório contém um arquivo chamado *0001_initial.py*, neste arquivo contém a *migration* que cria o modelo no banco de dados.

A cada nova migração executada para o mesmo modelo, novos arquivos são criados com a numeração sequencial no nome do arquivo, permitindo o registro do histórico de migrações.

2.4.4.2 O Comando Migrate

Este comando é responsável por aplicar e desaplicar migrações. Então após executar o comando *makemigrations* deve-se executar o comando *migrate* (DJANGO, 2022e). O comando utilizado foi o *python manage.py migrate*.

Segundo Ramos (2018), quando executado o comando *python manage.py makemigrations*, o *Django* cria o banco de dados e as *migrations*, porém não as executa, isto significa que o *Django* não aplica as alterações no banco de dados. Para que o *Django* aplique estas alterações, é necessário realizar três coisas. Primeiro é necessário que a

configuração da interface com o banco de dados esteja descrita no *settings.py*. Segundo, é necessário que os modelos e as *migrations* estejam definidas para este projeto e terceiro, é necessário a execução do comando *python manage.py migrate*.

Ao executar o comando *python manage.py migrate*, o *Django* irá criar diversas tabelas no banco de dados, referente ao modelo definido dentro do *models.py* (RAMOS, 2018).

2.4.4.3 Banco de Dados Django

Por padrão o *Django* utiliza o banco de dados *SQLite*, sendo assim, o *SQLite* já está incluso no *Django* e não precisa ser instalado (DJANGO, 2022f).

Segundo Python (2022c), o *SQLite* é uma biblioteca C que fornece um banco de dados leve baseado em disco que não requer um processo de servidor separado e permite acessar o banco de dados utilizando uma variante não padrão da linguagem de consulta *SQL*, alguns aplicativos podem utilizar o *SQLite* para armazenar dados internos. Além disso, é possível desenvolver um sistema utilizando o *SQLite* e em seguida migrar o código para um banco de dados maior, como o *postgreSQL*, este foi o processo que ocorreu na aplicação desenvolvida neste estudo.

2.5 FRAMEWORK DJANGO NA LINGUAGEM DE PROGRAMAÇÃO PYTHON

Segundo Django (2022e), o *Framework Django* foi criado por Adrian Holovaty e Simon Willison, quando em 2003 eles decidiram deixar de lado a linguagem de programação *PHP* para utilizarem o *Python*, a partir disso, conforme criavam sites intensivos e ricamente interativos, eles extraíram uma estrutura genérica do desenvolvimento *Web* o que lhes permitia criar aplicativos cada vez mais rápido. Portanto, no verão de 2005, a *World Online*, que era responsável por construir aplicações *Web* intensivas nos prazos do jornalismo decidiu abrir o software resultante, sendo ele o *Framework Django*.

2.5.1 A Linguagem de Programação Python

Segundo Rossum e Drake (2003),

Python é uma linguagem de programação poderosa e fácil de aprender. Possui estruturas de dados de alto nível muito eficientes e uma abordagem simples, mas eficaz para programação orientada a objetos. A sintaxe elegante e a tipagem dinâmica do *Python*, juntamente com sua natureza interpretada, o tornam uma linguagem ideal para *scripts* e desenvolvimento rápido de aplicativos em muitas áreas na maioria das plataformas.

O *Python* possui uma tipagem dinâmica e forte. Além disso, é uma linguagem orientada a objetos, multiplataforma e que possui multiparadigmas, portanto, é uma linguagem de programação funcional e imperativa (PYTHON, 2022a).

O *Python* foi desenvolvido no início da década de 1990, e foi criado pelo matemático *Guido Van Rossum* no Centro de Matemática e Tecnologia da Informação *CWI - Centrum Wiskunde e Informatica* na Holanda. Embora o *Python* possua a contribuição de outros autores para sua criação, *Guido Van Rossum* continua sendo o principal autor (PYTHON, 2022b).

No ano de 1991 Guido [...] faz o lançamento da primeira versão da linguagem como 0.9.0 utilizando-se para sua distribuição uma licença derivada da licença MIT (Massachusetts Institute of Technology). Hoje a linguagem é mantida por uma organização sem fins lucrativos chamada *Python Software Foundation* (MANZANO, 2018).

O nome da linguagem de programação *Python* foi inspirado no *Monty Python's Flying Circus*, um famoso grupo de comédia britânico da década de 1970 (PYTHON, 2022a). No entanto, segundo Manzano (2018) o nome *Python* passou a ter uma relação inevitável com o nome do animal após o primeiro lançamento sobre a linguagem *Python* da editora *O'Reilly* que tem o hábito de utilizar imagens de animais na capa de seus livros e sendo assim, utilizou a imagem da serpente da espécie *Python*.

Python é uma linguagem de programação código aberto, gratuito, interpretada, interativa e orientada a objetos, desenvolvida pela primeira vez em 1990 por *Guido Van Rossum*. No final de 1998, ele havia crescido para uma base de usuários estimada em 300.000, e estava começando a atrair grande atenção na indústria. (HAMMOND; ROBINSON, 2000).

Segundo Python (2022a) o uso de agrupamento de instruções e a inclusão de tipos de dados de alto nível no Python, foram criadas devido a experiência de *Guido Van Rossum* na implementação de uma linguagem interpretada no grupo ABC onde ele também aprendeu sobre design de linguagem. Sendo assim, essa é a origem de muitos recursos do Python, embora todos os detalhes criados sejam totalmente diferentes.

Atualmente, o *Python* já possui uma quantidade considerável de bibliotecas neces-

sárias para o seu funcionamento, sendo assim, possui bons recursos em sua biblioteca padrão e também via módulos e *Frameworks* desenvolvidos pela comunidade e é também uma linguagem bem organizada (PYTHON, 2022a).

Segundo Python (2022a), esta linguagem de programação vem com uma vasta biblioteca padrão que abrange áreas como processamento de strings, protocolos de internet, engenharia de *software* e interfaces de sistema operacional. Esta disponível também uma grande variedade de extensões de terceiros.

A versão do *Python* 3.10.7, foi lançada em 6 de setembro de 2022, sendo assim, esta foi a versão utilizada no processo de desenvolvimento da aplicação que se desenvolveu com este estudo.

2.5.2 O Uso da Linguagem de Programação Python em Aplicações Web

O *Python* também é utilizado em diversas aplicações. As aplicações mais famosas feitas em *Python*, que são *Youtube*, que inicialmente foi criado em *PHP*, mas pela necessidade constante de mudança optaram por usar a linguagem de programação *Python* (BYLEARN, 2020). O *Facebook*, que tem o *Python* como a terceira linguagem mais utilizada entre os seus desenvolvedores, neste caso uma de suas funções é fazer o gerenciamento de infraestrutura (BYLEARN, 2020). E a *Netflix*, que utiliza o a linguagem de programação *Python* como ferramenta de segurança e também em modelos de aprendizagem de máquina para o seu sistema de recomendações (BYLEARN, 2020).

No desenvolvimento deste estudo, utiliza-se o *Framework Django*, que é um *Framework* voltado para a linguagem de programação *Python*. Sendo assim, a linguagem de programação *Python* foi amplamente utilizada no processo de desenvolvimento da aplicação que se desenvolveu neste estudo.

2.6 FERRAMENTAS UTILIZADAS NO DESENVOLVIMENTO WEB

O *Visual Studio Code*, foi utilizado neste estudo para desenvolver a parte de *front-end* utilizando o *Framework Bulma CSS*. O *Visual Studio Code* possui algumas funções,

como por exemplo, a edição de código onde possui *IntelliSense* que ajuda no preenchimento automático de códigos (CODE, 2022). Além disso, possui diversas extensões, como por exemplo a *dracula* que proporciona melhor visualização do código ao desenvolvedor.

O *Pycharm*, é uma *IDE* criada para a linguagem de programação *Python*. O *Pycharm* facilita muito o desenvolvimento *web*, pois possui recursos como o editor de código inteligente, que tem como uma de suas funções o *IntelliSense* com reconhecimento de linguagem, a detecção de erros e as correções dinâmicas de código, a navegação inteligente pelo código e também possui refatorações rápidas e seguras (JETBRAINS, 2022b).

2.6.1 Editor de Código Fonte

O editor de código-fonte é um *software* feito para ajudar desenvolvedores na hora de programar. Estes editores de código possuem algumas funções, uma delas é que eles funcionam como editores de texto, além disso contém funcionalidades adicionais para gerenciar e editar o código-fonte (HOSTGATOR, 2021).

Os editores de código são exclusivos da área de programação. Alguns editores de códigos oferecem suporte a mais de uma linguagem de programação (HOSTGATOR, 2021).

Os editores de código-fonte não compilam código diferente das *IDEs*, os editores cuidam da sintaxe do código, ou seja, eles avisam imediatamente sobre algum erro, alguns até possuem recuo automático, deixando o código mais limpo e de fácil visualização, alguns também possuem preenchimento automático, essa função que permite que o editor seja capaz de completar alguns trechos de código de forma automática. Além disso, alguns editores de código-fonte nos oferecem a possibilidade de fazer download de *plugins* para facilitar nosso trabalho, como por exemplo o *Visual Studio Code* que foi o editor de código fonte utilizado neste estudo, permite fazer download de compiladores para uma determinada linguagem de programação (HOSTGATOR, 2021).

Alguns exemplos de editores de código-fonte são o *Sublime Text*, o *Visual Studio Code* e o *Notepad++* (HOSTGATOR, 2021).

2.6.1.1 *Visual Studio Code*

O *Visual Studio Code* é um editor de código-fonte, e foi desenvolvido pela *Microsoft*. O *Visual Studio Code* possui algumas funções, como por exemplo a edição de código com suporte a várias linguagens de programação, terminal de comando integrados e controle de versão. Além disso, possui diversas extensões, com elas pode-se adicionar diversas funcionalidades ao *Visual Studio Code* de forma rápida e simples (TREINAWEB, 2021).

A versão do *Visual Studio Code* 1.64, foi a versão utilizada no processo de desenvolvimento da aplicação que se desenvolveu com este estudo.

2.6.2 IDE

IDE sigla em inglês para *Integrated Development Environment*, que significa, Ambiente de Desenvolvimento Integrado, é uma ferramenta de desenvolvimento para editar códigos, acessar um terminal para realizar a execução de *scripts*, compilar e debugar utilizando apenas um único ambiente (ALURA, 2022).

Uma *IDE* possui diversas funções, e as mais comuns são as de editor de código, onde esta função é utilizada para escrever comandos de uma determinada linguagem de programação, como por exemplo o *Pycharm* que foi utilizado no desenvolvimento da aplicação que se desenvolveu neste estudo, e que é utilizado para escrever códigos da linguagem de programação *Python*. Além disso, as *IDEs* também possuem a função de preenchimento inteligente que permite completar alguns trechos de código de forma automática. Outras funções que as *IDEs* possuem é a de compilador e interpretador, que é a função que transformará todo o código de determinada linguagem de programação em linguagem de máquina, e a função de debugar que é utilizada para encontrar e corrigir erros de código (TREINAWEB, 2020).

Alguns exemplos de *IDEs* são *Intellij*, *Eclipse*, *Atom* e *Netbeans* (ALURA, 2022).

2.6.2.1 *Pycharm*

O *Pycharm* é uma *IDE* que fornece uma ampla gama de ferramentas essenciais para desenvolvedores *Python*. O *Pycharm* possui três edições disponíveis atualmente, que são as Edições *Community*, *Edu* e *Professional*, as edições *Community* e *Edu* são versões gratuitas e de código aberto, já a edição *Professional* é uma edição paga (JETBRAINS, 2022a).

A edição *community* que foi a edição utilizada no desenvolvimento deste estudo, possui assistência de código, refatorações, depuração visual e integração de controle de versões. A edição *Edu* é usada por quem está aprendendo ou quer aprender linguagens de programação e outras tecnologias que estão relacionadas com ferramentas educacionais integradas como esta. E a edição *Professional*, foi feita para desenvolvimento profissional em *Python*, *Web* e ciência de dados. A edição *Professional*, assim como a edição *Community* também possui a assistência de código, refatoração visual e integração de controle de versão, e além disso possui configurações remotas, implementações, suporte para estruturas web populares, como o *Django* e o *Flask*, e ainda possui suporte a banco de dados, entre outras ferramentas (JETBRAINS, 2022a).

A versão 2021.3.1 do *Pycharm*, lançada em 28 de dezembro de 2021, e foi a versão utilizada no processo de desenvolvimento da aplicação que se desenvolveu com este estudo.

3 APLICAÇÃO DOS FRAMEWORKS NO SISTEMA

Com o objetivo de exemplificar a aplicação do *Framework* de *front-end Bulma CSS* e o *Framework* de *back-end Django* com as informações encontradas neste trabalho, foi realizado um estudo para o desenvolvimento de um sistema *web* para o controle de horários e atividades na Universidade Federal de Santa Maria (UFSM)

3.1 CONTEXTO DO SISTEMA

Este sistema *web* tem como objetivo principal aplicar os conhecimentos que foram adquiridos durante o período em que se desenvolveu o estudo dos *Frameworks Bulma CSS* e *Django*.

O sistema foi criado para controlar horários, criar atividades e gerir pessoas para um laboratório de pesquisas da Universidade Federal de Santa Maria(UFSM). Além disso, o sistema deve permitir que os alunos e professores controlem suas frequências conforme a definição das atividades.

Segundo Mendonça (2014), o levantamento de requisitos tem um papel importante na construção de um sistema, pois é onde se iniciam todas as atividades do desenvolvimento do software [...].

Além disso, segundo DevMedia (2022), pode-se dividir os requisitos em funcionais, este tem como objetivo descrever as principais funcionalidades que o sistema irá realizar, e os não funcionais, que apresentam uma descrição geral de outros requisitos do sistema que podem até limitar o desenvolvimento do sistema, e também os requisitos de escopo não contemplado cujo seu objetivo é mostrar as funcionalidades não escolhidas do sistema a ser desenvolvido.

Após analisar os requisitos existentes, foi possível elaborar um documento chamado de documento de requisitos, este documento delimitou o escopo do conjunto de funcionalidades que o sistema deve prover. A Figura 36 apresenta os requisitos funcionais do sistema.

Figura 36 – Requisitos Funcionais do Sistema Desenvolvido.

Sistema de gestão de pessoas - Laboratório de Pesquisas da UFSM	
Documento de Requisitos	
Requisitos Funcionais	
1.	Criar Atividades
a.	Trocar Usuário Responsável pela Atividade
b.	Filtrar Usuários
2.	Gerar Relatórios
3.	Confirmar Frequência
4.	Cadastrar Pessoas
5.	Cadastrar Login de Pessoas
6.	Cadastrar Atividades
7.	Cadastrar Categorias de Atividades
8.	Cadastrar Atuação dos Usuários nas Atividades
9.	Cadastrar Níveis de Atuação

Fonte: A autora(2022)

No sistema desenvolvido neste estudo, é possível criar atividades, na página de atividades criadas é possível realizar a troca de usuários responsáveis pela atividade e também realizar buscas por um usuário específico. Além disso, nas páginas de relatórios é possível gerar um PDF com as informações contidas nos relatórios. O usuário também pode confirmar a frequência em nas atividades criadas.

O sistema possui diferentes níveis de perfis, sendo assim algumas das funções do sistema são exclusivas para os administradores, as funções destinadas aos administradores são: cadastrar pessoas, cadastrar o login de pessoas, cadastrar atividades, cadastrar categorias de atividades, cadastrar atuações dos usuários nas atividades e cadastrar os níveis de atuação dos usuários as atividades. E nas páginas de relatórios, somente o administrador pode gerar *PDFs*.

3.2 MODELAGEM DO SISTEMA

Segundo Guedes (2009),

[...] Um modelo de *software* captura uma visão de um sistema físico, e é uma abstração do sistema com um certo propósito, como descrever aspectos estruturais ou comportamentais do *software*. Esse propósito determina o que deve ser incluído no modelo e o que é considerado irrelevante. Assim um modelo descreve completamente aqueles aspectos do sistema físico que são relevantes ao propósito do modelo, no nível apropriado de detalhe.

Além disso, segundo Booch (2006), existem quatro objetivos principais para a criação dos modelos de *software*, sendo o primeiro deles ajudar e permitir visualizar o sistema da forma que ele é e da forma que ele ficará, ademais, os modelos permitem especificar a estrutura ou o comportamento de um sistema, sendo assim, proporcionam um guia para a construção do sistema e portanto, pode-se concluir que os modelos são uma forma de manter documentada todas as decisões que foram tomadas durante sua construção.

Conforme Guedes (2009), *UML* ou Linguagem de Modelagem Unificada tornou-se atualmente a linguagem-padrão de modelagem sendo adotada internacionalmente pela indústria de engenharia de *software*. Entretanto, não é uma linguagem de programação, mas sim uma linguagem de modelagem cujo objetivo é auxiliar os engenheiros de *software* a definirem as características do sistema, tais como seus requisitos, comportamento, estrutura lógica, a dinâmica de processos e até mesmo as suas necessidades físicas em relação ao equipamento sobre qual o sistema deverá ser implementado.

3.2.1 Diagrama de Classes

Segundo Guedes (2018),

O diagrama de classes é um dos mais importantes e utilizados da UML. Seu principal enfoque está em permitir a visualização das classes que irão compor o sistema com seus respectivos atributos e métodos, bem como em demonstrar com as classes do diagrama se relacionam, complementam e transmitem informações entre si. Esse diagrama apresenta uma visão estática de como as classes estão organizadas, preocupando-se em como definir a estrutura lógica delas. O diagrama de classes serve ainda como apoio para a construção da maioria dos outros diagramas da linguagem UML.

Além disso, segundo Guedes (2018), o diagrama de classes é composto de classes e associações existente entre elas, ou seja, relacionamento entre classes. As classes contém atributos que armazenam os dados dos objetos da classe, portanto as classes costumam ter relacionamentos entre si, chamados de associações, permitindo que elas

Figura 38 – Arquivo *models.py* da aplicação pessoa.

```
class Pessoa(models.Model):
    PERFIL_OPCOES = (
        ('A', 'Administrador'),
        ('U', 'Usuário'),
        ('T', 'Técnico')
    )
    nome = models.CharField('Nome', max_length=35, help_text='Informe o nome completo')
    email = models.EmailField('Email', max_length=45, help_text='Informe o email', unique=True)
    telefone = models.CharField('Telefone', max_length=14, help_text='Informe o telefone')
    matricula = models.CharField('Matricula', max_length=20, help_text='Informe a matricula', unique=True)
    perfil = models.CharField('Perfil', max_length=15, help_text='Perfil do Usuário', choices=PERFIL_OPCOES, default='U')

    class Meta:
        verbose_name = 'Pessoa'
        verbose_name_plural = 'Pessoas'

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

Fonte: A autora(2022)

Assim que criada a classe dentro de aplicação pessoa, foi dada continuidade ao processo de desenvolvimento da aplicação.

3.3 DESENVOLVIMENTO

Nesta seção serão apresentadas as principais interfaces do sistema.

O sistema em questão possui diferenciação de níveis de usuários que dependendo do nível, restringe o uso de algumas de suas funções.

Existem três níveis de perfis. O perfil “Usuário”, que possui funções limitadas dentro do sistema, não sendo possível, por exemplo, criar novos usuários ou editar e imprimir dados de outras pessoas cadastradas, mas pode realizar buscas dentro dos relatórios. O perfil “Técnico”, também possui funções limitadas dentro do sistema, não sendo possível, por exemplo, criar novos usuários ou editá-los mas pode imprimir dados de outras pessoas cadastradas e realizar buscas. Já o perfil de nível “administrador” tem acesso a todas as funcionalidades do sistema.

Para realização da parte de autenticação do sistema desenvolvido neste estudo, foi utilizada a *API Django Auth*. Uma *API* ou *Application Programming Interface* é definida como um conjunto de padrões que permite construir aplicativos com maior facilidade além do que o uso de *Frameworks* como o *Django* já permite. Uma *API* não permite ao usuário

saber que ela está sendo utilizado no desenvolvimento do sistema pois ela é "invisível" ao usuário comum (TREINAWEB, 2020a).

Além disso, a comunicação entre o sistema em desenvolvimento e a *API* é feita através de vários códigos que definem um comportamento específico para o sistema que esta sendo desenvolvido (TREINAWEB, 2020a).

O uso de *APIs* no geral ajudam o desenvolvedor *web* a ter mais produtividade, pois com elas é mais fácil realizar tarefas que eram mais demoradas (TREINAWEB, 2020a). Neste sistema foi utilizada a *API Django Auth* para criação de toda a parte de autenticação, desde os códigos até as interfaces de login e cadastro de usuários tornando sua criação mais produtiva.

Figura 39 – Página de login criada com *Django Auth*.

A imagem mostra uma interface web de login com um fundo azul sólido. No topo, o título "ACESSE SUA CONTA" está centralizado em letras brancas. Abaixo dele, há um formulário branco com o título "Identificação de Usuário". O formulário contém dois campos de entrada: o primeiro, rotulado "Identificação de Usuário", possui um ícone de pessoa e o valor "202224081019"; o segundo, rotulado "Senha", possui um ícone de cadeado. Abaixo dos campos, há um botão azul com o texto "Acessar Conta" em branco.

Fonte: A autora(2022)

Para validar o dados inseridos nos campos de *login*, utiliza-se no `<form>` de *login* o *action* e o *method*. O *action* vai enviar os dados inseridos no formulário para a *view* de *login*, ou seja, para a página de *login*. E o *method=POST*, solicita que o servidor *web* aceite os dados que foram anexados no formulário. O *action* e o *method* podem ser vistos da Figura 40.

Figura 40 – Apresenta *action* e o *method* de login.

```
<form action="{% url 'login' %}" method="POST" class="box">
  {% csrf_token %}

  {% for msg in messages %}
    <div class="notification is-warning"...>
  {% endfor %}

  <div class="field"...>

  <div class="field"...>

  <div class="field has-text-centered"...>
</form>
```

Fonte: A autora(2022)

Assim que definidos o *action* e o *method*, deve-se definir no arquivo *views.py* que se o *request* de *method* for igual a *GET* ele irá retornar a página de login de usuários, caso isso não aconteça é porque os dados de *login* foram preenchidos, sendo assim ele irá utilizar estes dados para realizar o *login* do usuário no sistema. Este processo, pode ser visto na Figura 41.

Figura 41 – Código utilizado para validar o login do usuário.

```
def login(request):
    if request.method == "GET":
        return render(request, 'login.html')
    else:
        username = request.POST.get('username')
        senha1 = request.POST.get('senha')

        user = authenticate(username=username, password=senha1)
```

Fonte: A autora(2022)

Assim que o usuário clicar em "Acessar Conta", será feita uma busca nos usuários que estão cadastrados, se tudo estiver correto o usuário é redirecionado para a página inicial do sistema. Caso alguma informação inserida esteja incorreta, será mostrado ao usuário a mensagem "Matrícula ou Senha Inválidos". O arquivo *views.py* com o código utilizado para exibir a mensagem de "Matrícula ou senha inválidos" pode ser visto na Figura

42.

Figura 42 – Arquivo *views.py* da aplicação *login*.

```
if user:
    login_django(request, user)
    if request.user.is_authenticated:
        return render(request, 'index.html')
    else:
        messages.success(request, 'Matrícula ou Senha inválidos')
        return render(request, 'login.html')
```

Fonte: A autora(2022)

A Figura 43 mostra como a mensagem de "Matrícula ou Senha Inválidos" aparece para o usuário.

Figura 43 – Página de acesso ao sistema *web* com a mensagem de erro.



A imagem mostra a interface de login de um sistema web. O título principal é "ACESSE SUA CONTA" em letras brancas sobre um fundo azul. Abaixo, há uma caixa branca com uma mensagem de erro em um banner amarelo: "Olá! Matrícula ou Senha inválidos". Seguem os campos de "Identificação de Usuário" (com um ícone de pessoa e o valor "202224081019") e "Senha" (com um ícone de cadeado). Um botão azul "Acessar Conta" está na base da caixa.

Fonte: A autora(2022)

Para que fosse possível mostrar as mensagens citadas acima no sistema foi necessário inserir no `<form>` de *login* um `for messages`. Este foi o responsável por mostrar a mensagem ao usuário de que alguma das informações de *login* inseridas está incorreta. O `for messages` pode ser visto na Figura 44.

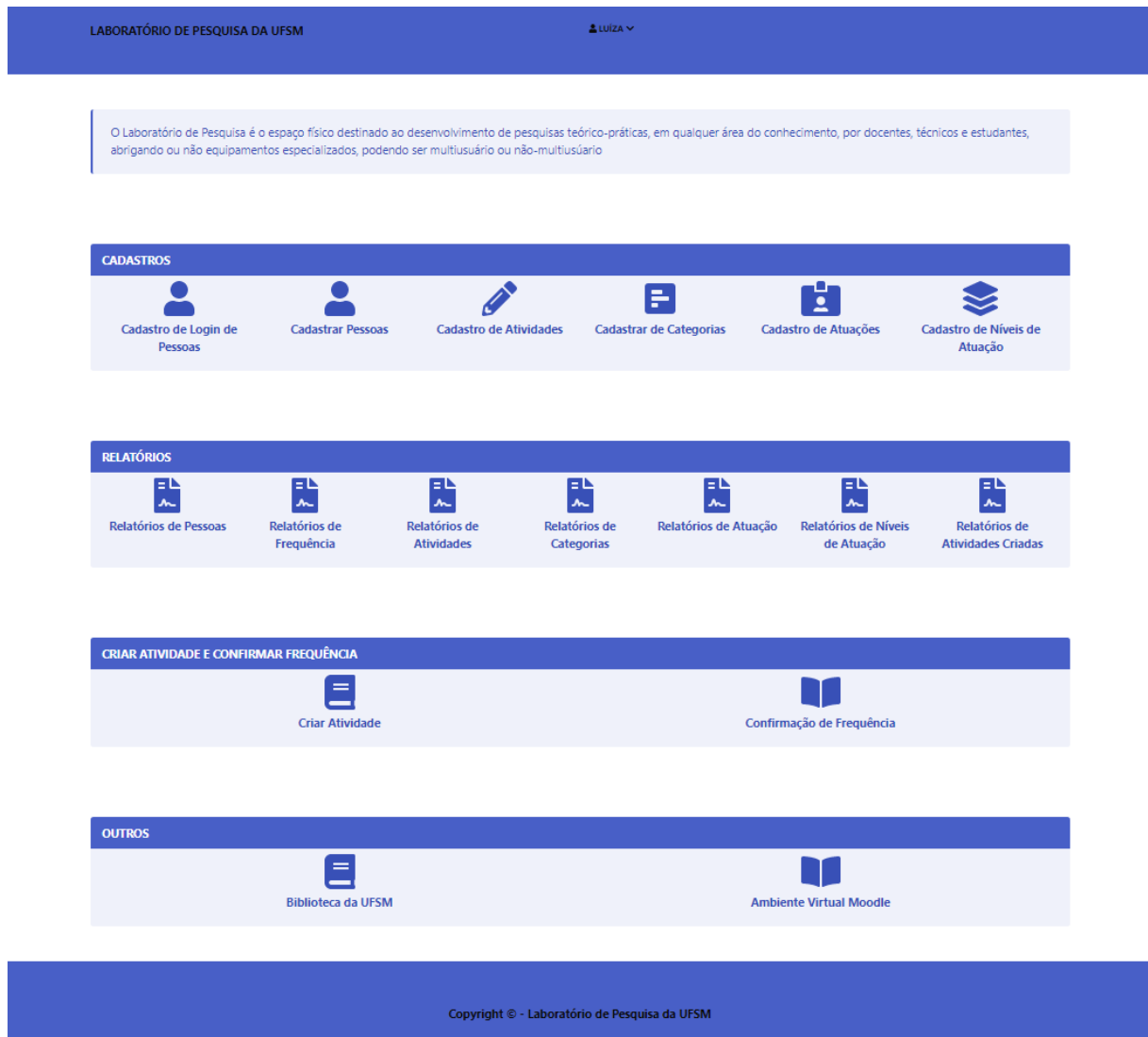
Figura 44 – *for message* inserido no login de pessoas.

```
{% for msg in messages %}
  <div class="notification is-warning">
    <strong>Olá!</strong>
    {{ msg }}
  </div>
{% endfor %}
```

Fonte: A autora(2022)

Assim que o *login* for feito o usuário logado terá acesso a página inicial do sistema, na página inicial contém um menu com todas as funcionalidades que o usuário pode exercer dentro do sistema. Foi com o perfil de administrador logado que foram feitas as imagens apresentadas nesta seção. A página inicial do sistema desenvolvido neste estudo pode ser vista na Figura 45.

Figura 45 – Página inicial.

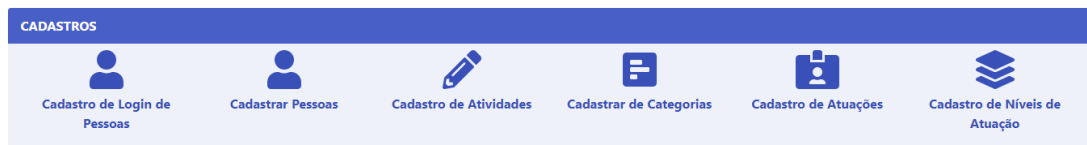


Fonte: A autora(2022)

A página inicial é composta por três menus principais, o primeiro é o menu de cadastros, em seguida o menu de relatórios e por fim o menu de criar atividade e confirmar frequência.

O menu de cadastros possui as opções de cadastro de login de pessoas, cadastro de pessoas, cadastro de atividades, cadastro de categorias, cadastro de atuações e cadastro de níveis de atuação. O menu de cadastros do sistema desenvolvido neste estudo pode ser visto na Figura 46.

Figura 46 – Menu de cadastros.



Fonte: A autora(2022)

O menu de relatórios possui as opções de relatório de pessoas, relatório de atividades, relatório de categorias, relatório de atuação, relatório de níveis de atuação, relatório de atividades criadas e relatório de confirmação de frequência. O menu de relatórios do sistema desenvolvido neste estudo pode ser visto na Figura 47.

Figura 47 – Menu de relatórios.



Fonte: A autora(2022)

E por fim o menu de criar atividade e confirmar frequência, onde possui duas opções na qual estão descritas em seu nome. O menu de criar atividade e confirmar frequência do sistema desenvolvido neste estudo pode ser visto na Figura 48.

Figura 48 – Menu de criar atividade e confirmar frequência.



Fonte: A autora(2022)

As principais páginas de cadastro são: cadastro de login de pessoas, cadastro de pessoas e cadastro de atividades.

A página de cadastro de *login* de pessoas possui os campos de nome, *e-mail*, matrícula, senha e confirmação de senha. O cadastro de *login* de pessoas só pode ser realizado pelo perfil "administrador". A página de cadastro de *login* de pessoas pode ser vista na Figura 49.

Figura 49 – Página de cadastro de login de pessoas.

CADASTRAR LOGIN DO USUÁRIO

Nome Completo

E-mail

Informações de Login

A matrícula cadastrada aqui será usada no campo de identificação do usuário na hora de acessar a sua conta.

Matrícula

Senha

Confirmar Senha

Cadastrar Usuário

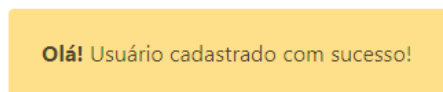
Fonte: A autora(2022)

O processo de validação dos dados inseridos nos campos de cadastro de *login* de pessoas, foi o mesmo processo utilizado para validar os dados inseridos no formulário de

login de usuários.

Após preencher todos os dados da pessoa e clicar em "cadastrar o usuário", o administrador logado será redirecionado para a página inicial do sistema e será mostrado na tela uma notificação de que o usuário foi cadastrado com sucesso. E caso o usuário já esteja cadastrado, irá mostrar dentro da área de cadastro de *login* de usuário uma mensagem informando que a matrícula digitada já está cadastrada no sistema.

Figura 50 – Mensagem de que o usuário foi cadastrado com sucesso no sistema.



Fonte: A autora(2022)

Para que fosse possível mostrar a mensagem citada acima no sistema foi necessário inserir no *<form>* de cadastro um *for messages*, também foi necessário inserir um *for messages* no *template registros.html* para que a mensagem fosse mostrada na página inicial. Este foi o responsável por mostrar a mensagem de usuário já cadastrado para o "administrador" que é quem realiza os cadastros.

Assim que inserido o *for messages* no *<form>* de cadastro e na página inicial, deve-se definir no arquivo *views.py* as mensagens que serão mostradas ao "administrador" caso o usuário já esteja cadastrado no sistema. Se for cadastrar o usuário com senhas iguais o "Administrador" será informado que as "Senhas são diferentes", esta mensagem será mostrada na página inicial do sistema. Se o usuário já estiver cadastrado o "Administrador" será informado com a mensagem "Matrícula já cadastrada!", e se o e-mail já estiver cadastrado o "Administrador" será informado com a mensagem "E-mail já cadastrado!". Este processo pode ser visto na Figura 51.

Figura 51 – Código utilizado para inserir as mensagens dentro do *for messages*.

```

if senha1==senha2:
    if User.objects.filter(username=username).exists():
        messages.success(request, 'Matricula já cadastrada!')
        return render(request, 'CadastroLogin.html')
    elif User.objects.filter(email=email).exists():
        messages.success(request, 'Email já cadastrado!')
        return render(request, 'CadastroLogin.html')
    else:
        user = User.objects.create_user(first_name=first_name, email=email, username=username, password=senha1)
        user.save()
        messages.success(request, 'Usuário cadastrado com sucesso!')
else:
    messages.success(request, 'As senhas são diferentes, para cadastrar informe a mesma senha no campo de "SENHA E CONFIRMAÇÃO DE SENHA")
return render(request, 'index.html')

```

Fonte: A autora(2022)

A página de cadastro de pessoas possui os campos de nome, *e-mail*, telefone, matrícula e perfil. O cadastro de pessoas só pode ser realizado pelo perfil "administrador", após preencher todos os dados da pessoa e clicar em salvar, o administrador logado será redirecionado para a página de relatórios de pessoas onde o nome da pessoa cadastrada será mostrado no final da tabela e se clicar em cancelar, o administrador logado será redirecionado novamente para a página inicial do sistema. A página de cadastro de pessoas pode ser vista na Figura 52.

Figura 52 – Página de cadastro de pessoas.

CADASTRO DE PESSOAS

Nome:

Email:

Telefone:

Matrícula:

Perfil:

Fonte: A autora(2022)

A página de cadastro de atividades possui os campos de título, descrição, data de início, data de encerramento, categoria e prioridade. O cadastro de atividades só pode ser realizado pelo perfil "administrador", após preencher todos os dados da atividade que deseja cadastrar e clicar em salvar, o administrador logado será redirecionado para a página de relatórios de pessoas onde o nome da pessoa cadastrada será mostrado no final da tabela e se clicar em cancelar, o administrador logado será redirecionado novamente para a página inicial do sistema. A página de cadastro de atividades do sistema desenvolvido neste estudo pode ser visto na Figura 53.

Figura 53 – Página de cadastro de atividades.

CADASTRO DE ATIVIDADES

Título:

Descrição:

Data De Início:
 

Data De Encerramento:
 

Categorias:

Prioridade:

Fonte: A autora(2022)

A página de relatório de pessoas mostra todos os dados que foram inseridas na hora do cadastro. Além disso, o sistema dá a opção de editar caso algum dado inserido esteja incorreto e excluir uma pessoa do relatório caso não seja mais necessário o seu cadastro. Os relatórios podem ser acessados pelo perfil "administrador, usuário e técnico", e além do administrador, o perfil "usuário e técnico" podem realizar buscas mas somente o perfil "técnico" pode imprimir os dados das tabelas. A página de relatório de pessoas do sistema desenvolvido neste estudo pode ser vista na Figura 54.

Fonte: A autora(2022)

Figura 54 – Página de relatório de pessoas.

Nome da Pessoa:

[Buscar](#) [Imprimir](#) [CADASTRAR OUTRO USUÁRIO+](#)

PESSOAS CADASTRADAS

Nome	E-mail	Telefone	Matrícula	Perfil	Ações
Julia Maria	julia@exemplo.com	(55) 5555-5555	202126041011	Usuários	✎ 🗑

A página de relatório de atividades mostra todos os dados da atividade que foram inseridas na hora do cadastro. Além disso, o sistema dá a opção de editar caso algum dado inserido esteja incorreto e excluir uma atividade do relatório caso não seja mais necessário o seu cadastro. Os relatórios podem ser acessados pelo perfil "administrador, usuário e técnico", e além do administrador, o perfil "usuário e técnico" podem realizar buscas mas somente o perfil "técnico" pode imprimir os dados das tabelas. O perfil "usuário" só pode criar atividades com os nomes de atividades cadastradas pelo perfil "administrador" eles não podem cadastrar uma atividade, para isso é necessário aguardar que o administrador cadastre um nome de atividade por exemplo. A página de relatório de atividades do sistema desenvolvido neste estudo pode ser vista na Figura 55.

Figura 55 – Página de relatório de atividades.

Nome da Atividade:

[Buscar](#) [Imprimir](#) [CADASTRAR OUTRA ATIVIDADE+](#)

ATIVIDADES CADASTRADAS

Título da Atividade	Descrição	Data de Início	Data de Encerramento	Prioridade	Categoria	Ações
Atividade Um	Descrição Um	23 de Dezembro de 2022 às 16:00	23 de Dezembro de 2022 às 19:00	Média	Laboratório	✎ 🗑

Fonte: A autora(2022)

A página de criação de atividades possui os campos de pessoas, atividades, categorias, atuações, níveis de atuação, data de início e encerramento, pode-se também inserir um texto mais longo, um arquivo *txt*, *pdf* ou outros e definir uma situação da a atividade. Os campos de pessoas, atividades, categorias, atuações, níveis de atuação e situação são do tipo *select*, o formulário já trás todos dados cadastrados com seu respectivo dado, sendo assim, é mais fácil a criação da atividade. O campo de situação possui quatro opções, em andamento, concluído, encerrado e cancelado. A criação de atividades pode ser realizada pelo perfil "administrador, Usuário e Técnico", após preencher todos os dados da atividade

que deseja criar e clicar em salvar, o perfil logado será redirecionado para a página de relatórios de atividades criadas onde o nome da atividade criada será mostrado no final da tabela e se clicar em cancelar, o perfil logado será redirecionado novamente para a página inicial do sistema. A página de criação de atividades do sistema desenvolvido neste estudo pode ser vista na Figura 56 e o relatório da atividade criada pode ser visto na Figura 57.

Figura 56 – Página de criação de atividades.

CRIAR ATIVIDADE

Pessoas:


Atividades:

Categorias:

Atuações:

Nível De Atuação:

Data De Início:
 

Data De Encerramento:
 

Texto:

Arquivo:
 Nenhum arquivo escolhido



Situação:

Fonte: A autora(2022)

Figura 57 – Relatório das atividades criadas.

Nome: Atividade:

ATIVIDADES CRIADAS

Usuário Responsável pela Atividade Criada	Atividade	Categoria	Atuação	Nível de Atuação	Data de Início	Data de Encerramento	Texto	Arquivo	Situação	Ações
Julia Maria	Atividade Um	Organização	Coordenador	Pós-graduação	23 de Dezembro de 2022 às 00:00	23 de Dezembro de 2022 às 00:00	Texto exemplo	Arquivo	Concluído	 

Fonte: A autora(2022)


A página de confirmação de frequência pode ser acessada por todos os perfis "administrador, usuário e técnico". Os campos a serem preenchidos são pessoas, atividades e data. Os campos de pessoas, atividades são do tipo *select*, ou seja, o formulário já trás todos dados cadastrados com seu respectivo dado, sendo assim, é mais fácil a confirmação de frequência. Após preencher todos os dados de confirmação de frequência basta clicar em salvar e o perfil logado será redirecionado para a página de relatório de confirmação de frequência onde a confirmação de frequência será mostrada ao final da tabela e se clicar em cancelar, o perfil logado será redirecionado novamente para a página inicial do sistema. A página de confirmação de frequência do sistema desenvolvida neste estudo pode ser vista na Figura 58.

Figura 58 – Página de confirmação de frequência.

CONFIRMAÇÃO DE FREQUÊNCIA

Pessoas:

Atividades:

Data:
 

Fonte: A autora(2022)

A página de relatório de confirmação de frequência que mostra todos os dados inserido no formulário de confirmação de frequência. Além disso, o sistema dá a opção

de editar caso algum dado inserido esteja incorreto e excluir uma frequência do relatório caso não seja mais necessário o seu cadastro. Os relatórios podem ser acessados pelo perfil "administrador, usuário e técnico", além do administrador, o perfil "usuário e técnico" podem realizar buscas mas somente o perfil "técnico" pode imprimir os dados das tabelas.

Figura 59 – Página de relatório de confirmação de frequência.

Usuário	Atividade	Data	Ações
Julia Maria	Atividade Um	23 de Dezembro de 2022	

Fonte: A autora(2022)

Para realizar a geração de relatórios em *PDF* neste estudo, foi necessário a utilização do *WeasyPrint* dentro da aplicação, este é um pacote que utiliza um conteúdo que está em *HTML* e transforma-o em *PDF*.

Primeiro, é necessário realizar a instalação do *WeasyPrint* utilizando o comando `python -m pip install WeasyPrint`. Com a execução deste comando, o *pip* irá buscar todos os arquivos necessários do *WeasyPrint* e realizar a instalação dentro da aplicação. O *WeasyPrint* foi construído com a biblioteca *GTK+*, sendo assim, é preciso realizar o *download* desta biblioteca e instalá-la no computador. Após realizar a instalação do *WeasyPrint* e da biblioteca *GTK+* é preciso atualizar o arquivo *requirements.txt* utilizando o comando `pip freeze > requirements.txt`.

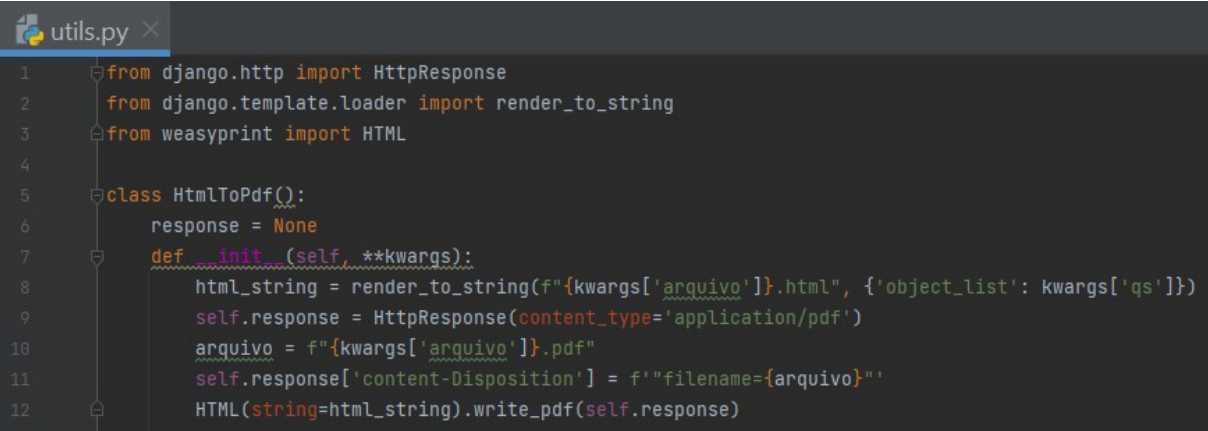
Após realizar todas as etapas de instalação, foi criado dentro da aplicação *home* deste sistema um arquivo chamado de *utils.py*, que é responsável pelo armazenamento de uma nova classe. Após criado o arquivo é necessário realizar algumas importações de alguns arquivos e classes que serão utilizadas na definição da classe que será criada. A primeira importação é a do *HttpResponde* utilizando o `from django.http import HttpResponde`, em seguida é realizada a importação do *render_to_string* utilizando o `from django.template.loader import render_to_string` e por fim a importação do *HTML* utilizando o `from weasyprint import HTML`.

O nome da classe criada é `class HtmlToPdf()`, esta classe irá utilizar um arquivo que esta no formato de *HTML* e transforma-la em um arquivo *PDF*. Sendo assim é necessário realizar algumas definições dentro do método construtor. Foi declarada uma variável identificada como *html_string* que chama o *render_to_string*, esta função faz a renderização

de um *HTML* para *String*, é necessário observar que o *kwargs* esta buscando o arquivo que é passado como parâmetro e este arquivo irá ser um *HTML* e também é utilizado um *object_list* que na realidade é o retorno de um *QuerySet*. No *self.response* esta mostrando o tipo de arquivo que será gerado, sendo assim, será gerado um arquivo do tipo *PDF*.

Além disso, é declarada uma variável *arquivo*, é necessário observar aqui que é utilizado o mesmo parâmetro utilizado no *render_to_string*, a única diferença é que aqui ao invés de ser um arquivo *HTML* é passada com uma extensão de arquivo *PDF*.

Figura 60 – Arquivo *utils.py* com a *class HtmlPdf()*.



```

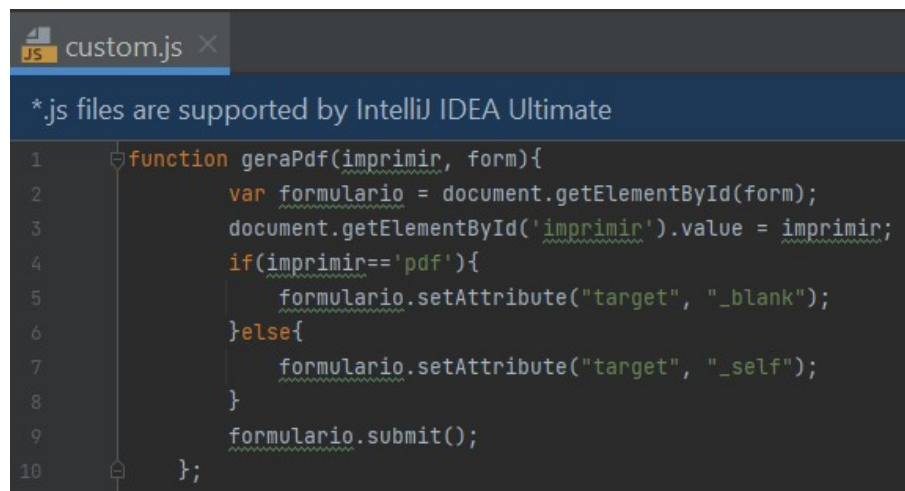
1  from django.http import HttpResponse
2  from django.template.loader import render_to_string
3  from weasyprint import HTML
4
5  class HtmlToPdf():
6      response = None
7      def __init__(self, **kwargs):
8          html_string = render_to_string(f'{kwargs["arquivo"]}.html', {'object_list': kwargs['qs']})
9          self.response = HttpResponse(content_type='application/pdf')
10         arquivo = f'{kwargs["arquivo"]}.pdf'
11         self.response['content-Disposition'] = f'filename={arquivo}'
12         HTML(string=html_string).write_pdf(self.response)

```

Fonte: A autora(2022)

Após criar a classe, é preciso criar uma função *Javascript* para o botão *imprimir*, este botão será responsável pela geração de *PDFs*. Na aplicação *home*, que é a principal aplicação do sistema desenvolvido neste estudo, possui uma pasta chamada *static* e nesta pasta contém outra pasta chamada *js*, nesta pasta contém um arquivo chamado *custom.js*. No arquivo *custom.js*, a *function geraPDF* que passa dois parâmetros, o *imprimir* e um *form*, sendo assim, este formulário irá mudar de acordo com *HTML* que faz a chamada da função, então ele busca pelo *id* do elemento *HTML*, o formulário. Além disso, existe um variável chamada *imprimir*, portanto a função testa se a variável *imprimir* é == *PDF*, se for igual, a função cria uma nova janela no *browser* e nesta janela será exibido o *PDF* que foi gerado. Caso contrario, se o conteúdo da variável *imprimir* não for == *PDF*, ele gerará o formulário na mesma janela do **browser** e após isso, é feito o *submit* do formulário. A função criada no arquivo *custom.js* pode ser vista na Figura 61.

Figura 61 – *function geraPdf(imprimir, form).*




```

1  function geraPdf(imprimir, form){
2      var formulario = document.getElementById(form);
3      document.getElementById('imprimir').value = imprimir;
4      if(imprimir=='pdf'){
5          formulario.setAttribute("target", "_blank");
6      }else{
7          formulario.setAttribute("target", "_self");
8      }
9      formulario.submit();
10 };
```

Fonte: A autora(2022)

Posteriormente, no arquivo *views.py* de cada uma das aplicações criadas, é necessário importar a classe que foi implementada do arquivo *utils.py*. Na classe contida no arquivo *views.py* é preciso reescrever o método *get* utilizando um *override*. Na *def get(self, *args, **kwargs)* é testado se no método *get* da requisição que foi feita a página, a variável *imprimir* é "==" 'PDF'. Se for igual, é feita a chamada da classe *HtmlToPdf* criando uma instancia, passando o nome do arquivo e também é passada a *QuerySet* pois caso dentro da aplicação tenha sido realizada alguma consulta de uma determinada atividade por exemplo, o método passará essa consulta na variável *qs* que representa o *QuerySet* e retornará o *Html_pdf.response*. Caso contrario se não for *PDF* retornará apenas a própria *view*. Este processo pode ser visto na Figura 62.

Figura 62 – *def get(self, *args, **kwargs)* no arquivo *views.py*.



```

#Gera_PDF
def get(self, *args, **kwargs):
    if self.request.GET.get('imprimir') == 'pdf':
        html_pdf = HtmlToPdf(arquivo='RelatoriosAtividadesPDF', qs=self.get_queryset())
        return html_pdf.response
    else:
        return super(AtividadeView, self).get(*args, **kwargs)
```

Fonte: A autora(2022)

Após isso, é necessário criar um novo *template*. Ao criar o novo *template*, foi definido o nome de *RelatoriosAtividadesPDF.html*. Este *template* representa a estrutura do documento *HTML* que será transformado em *PDF*. Neste *PDF*, é testado se tem valores dentro do *QuerySet*, se existem objetos dentro da lista, este irá gerar uma tabela, percorrerá o *object_list*, caso contrario exibirá a mensagem de "Não existem atividades cadastradas". Este *template* pode ser visto na Figura 63.

Figura 63 – *Template RelatoriosAtividadesPDF.html*.

```

1  <!DOCTYPE html>
2  {% load static %}
3  <html lang="pt-br">
4      {% block head %}
5          {% include 'head.html' %}
6      {% endblock %}
7  <body>
8      <div class="container has-text-centered">
9          <h1 class="title is-size-1">Sistema de Atividades</h1>
10     </div>
11     <section class="section">
12         {% if object_list %}
13             <h2 class="subtitle is-size-5 is-uppercase has-text-weight-bold has-text-black">Atividades Cadastradas</h2>
14             <div class="table">
15                 <table class="table is-table is-bordered is-striped is-narrow is-hoverable is-fullwidth">
16                     <thead>
17                         <tr...>
25                     </thead>
26                     <tbody>
27                         {% for RelatoriosAtividades in object_list %}
28                         <tr>
29                             <td>{{ RelatoriosAtividades.titulo }}</td>
30                             <td>{{ RelatoriosAtividades.descricao }}</td>
31                             <td>{{ RelatoriosAtividades.data_ini }}</td>
32                             <td>{{ RelatoriosAtividades.data_fim }}</td>
33                             <td>{{ RelatoriosAtividades.get_prioridade_display }}</td>
34                             <td>{{ RelatoriosAtividades.categoria.get_nome_display }}</td>
35                         </tr>
36                         {% endfor %}
37                     </tbody>
38                 </table>
39             </div>
40         {% else %}
41             <h2>Não Existem Atividades Cadastradas!</h2>
42         {% endif %}
43     </section>
44 </body>
45 </html>

```

Fonte: A autora(2022)

E é necessário também atualizar o *template RelatoriosAtividades.html*, neste *template* é preciso adicionar o botão imprimir. Onde será passado um valor no *input* do tipo *hidden*, este valor fica dentro do formulário mas ele não aparece, o nome do *input* é *imprimir* e o *id* também possui o nome de *imprimir* e o *value* irá mudar no momento em que for

feita a chamada da função. Este processo pode ser visto na Figura 64.

Figura 64 – Criação do *input* imprimir.

```
<input type="hidden" name="imprimir" id="imprimir" value="">
```

Fonte: A autora(2022)

Ao criar o botão é necessário alterar o *onclick* do botão *buscar* que passa a chamar a função *geraPDF* criada no *Javascript*. Neste momento como é somente realizar a busca, não é passado como parâmetro o *PDF*, pois a busca irá retornar registros que podem ou não serem imprimidos e o nome do formulário que esta sendo passado é o mesmo que foi definido no *<form>* que é *formListaAtividades*. E no botão *imprimir* é passado o parâmetro *PDF*, este parâmetro é a *String* testada dentro da função para ver se irá ou não gerar um *pdf*. Além disso, é testado se o *object_list* tem valor, pois se não existir valores dentro dele, não haverá o que imprimir. O *onclick* do botão buscar e imprimir pode ser visto nas Figuras 65 e 66.

Figura 65 – botão imprimir.

```
<div class="control">
  <button type="button" class="button is-link" onclick="geraPdf('pdf', 'formListaAtividades');">
    Imprimir
  </button>
</div>
```

Fonte: A autora(2022)

Figura 66 – Botão buscar.

```
<div class="control">
  <button type="button" class="button is-link" onclick="geraPdf('', 'formListaAtividades');">
    Buscar
  </button>
</div>
```


Fonte: A autora(2022)

A Figura 67 mostra a página de relatórios de atividades contendo as atividades cadastradas no sistema e o botão imprimir. E a Figura 68 mostra o *PDF* que foi gerado.

Figura 67 – Página de relatórios de atividades.



Nome da Atividade:

ATIVIDADES CADASTRADAS

Título da Atividade	Descrição	Data de Início	Data de Encerramento	Prioridade	Categoria	Ações
Atividade Um	Descrição Um	23 de Dezembro de 2022 às 16:00	23 de Dezembro de 2022 às 19:00	Média	Laboratório	 

Fonte: A autora(2022)

Figura 68 – PDF gerado das atividades cadastradas.

1 / 1
100%
 

Sistema de Atividades

ATIVIDADES CADASTRADAS

Título da Atividade	Descrição	Data de Início	Data de Encerramento	Prioridade	Categoria
Projeto Um	Descrição do Projeto Um	13 de Agosto de 2022 às 19:20	25 de Agosto de 2022 às 22:20	Média	Externo
Atividade Um	Descrição da atividade um	4 de Agosto de 2022 às 18:30	12 de Agosto de 2022 às 20:55	Alta	Organização

Fonte: A autora(2022)

O *CRUD* é um conjunto de operações que pertence ao agrupamento chamado de *Data Manipulation Language (DML)* ou *Linguagem de Manipulação de Dados*. Este, trata-se de um grupo de comandos de *linguagem SQL* que é utilizado para Criar, Ler, Alterar e Deletar informações dentro do banco de dados (COODESH, 2022).

No sistema desenvolvido neste estudo é possível criar, ler, alterar e deletar dados. A classe utilizada como exemplo é a classe da aplicação atividade.

Dentro da classe foi utilizado o *form_class* que passa o formulário base criado que

é o *AtividadeModelForm*, é necessário também informar qual modelo irá ser utilizado em *model*, o nome do *template* e o *sucess_url* que quando for executada a inclusão de uma nova atividade, este irá redirecionar o usuário diretamente para a *URL RelatorioAtividade*. Esta classe pode ser vista na Figura 69.

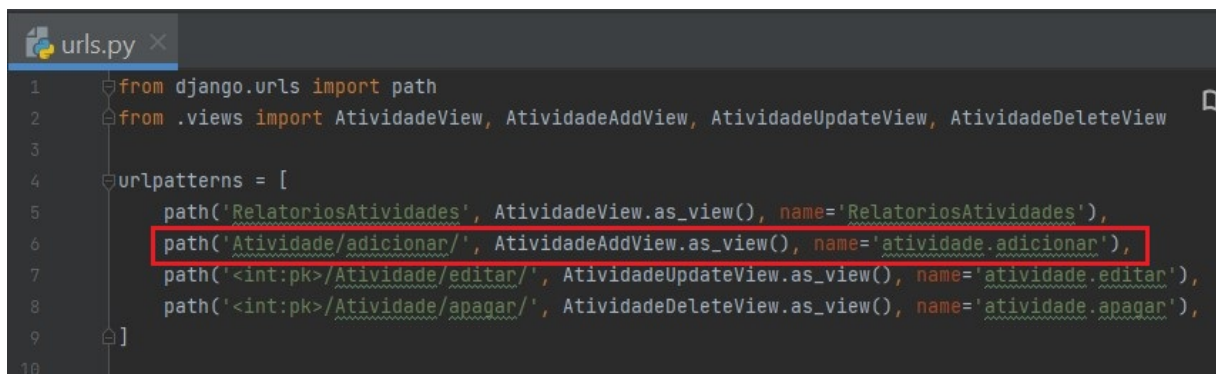
Figura 69 – A classe *AtividadeAddView*.

```
class AtividadeAddView(CreateView):
    form_class = AtividadeModelForm
    model = Atividade
    template_name = 'AtividadeForm.html'
    success_url = reverse_lazy('RelatoriosAtividades')
```

Fonte: A autora(2022)

Após criada a *class AtividadeAddView(CreateView)*, é necessário criar um *path* para a nova *URL*. Foi criada um *path Atividade/adicionar/*, chamando a *view Atividade-AddView* e o nome é *atividade.adicionar*. Este processo pode ser visto na Figura 70.

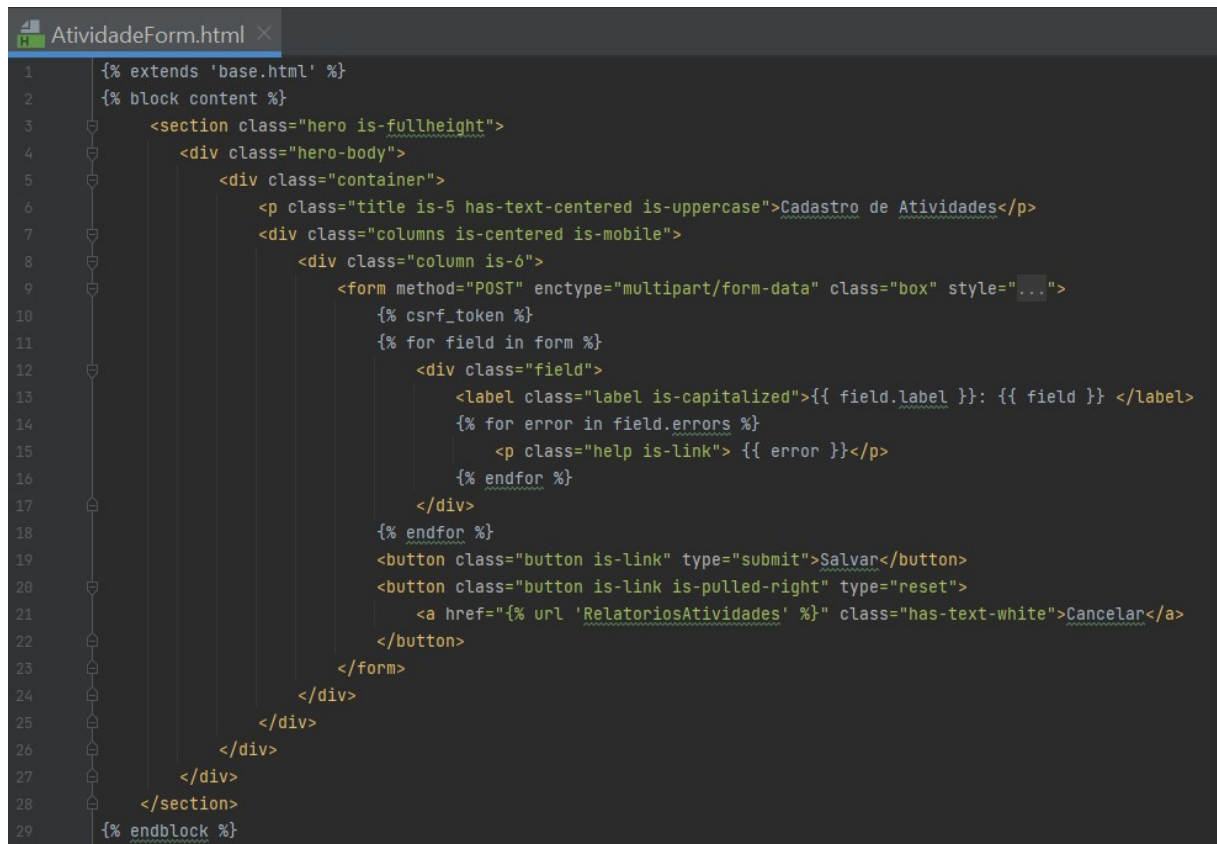
Figura 70 – *path* no arquivo *urls.py*.



```
urls.py
1 from django.urls import path
2 from .views import AtividadeView, AtividadeAddView, AtividadeUpdateView, AtividadeDeleteView
3
4 urlpatterns = [
5     path('RelatoriosAtividades', AtividadeView.as_view(), name='RelatoriosAtividades'),
6     path('Atividade/adicionar/', AtividadeAddView.as_view(), name='atividade.adicionar'),
7     path('<int:pk>/Atividade/editar/', AtividadeUpdateView.as_view(), name='atividade.editar'),
8     path('<int:pk>/Atividade/apagar/', AtividadeDeleteView.as_view(), name='atividade.apagar'),
9 ]
```

Fonte: A autora(2022)

Após este processo, cria-se o *template AtividadeForm.html*, contendo o formulário para inserir as informações que deseja salvar, com os botões *salvar* e *cancelar*. O *template AtividadeForm.html* pode ser visto na Figura 71.

Figura 71 – *Tempalte AtividaForm.html*.


```

1  {% extends 'base.html' %}
2  {% block content %}
3      <section class="hero is-fullheight">
4          <div class="hero-body">
5              <div class="container">
6                  <p class="title is-5 has-text-centered is-uppercase">Cadastro de Atividades</p>
7                  <div class="columns is-centered is-mobile">
8                      <div class="column is-6">
9                          <form method="POST" enctype="multipart/form-data" class="box" style="...">
10                             {% csrf_token %}
11                             {% for field in form %}
12                                 <div class="field">
13                                     <label class="label is-capitalized">{{ field.label }}: {{ field }} </label>
14                                     {% for error in field.errors %}
15                                         <p class="help is-link">{{ error }}</p>
16                                     {% endfor %}
17                                 </div>
18                             {% endfor %}
19                             <button class="button is-link" type="submit">Salvar</button>
20                             <button class="button is-link is-pulled-right" type="reset">
21                                 <a href="{% url 'RelatoriosAtividades' %}" class="has-text-white">Cancelar</a>
22                             </button>
23                         </form>
24                     </div>
25                 </div>
26             </div>
27         </section>
28     {% endblock %}

```

Fonte: A autora(2022)

No sistema, basta preencher um dos formulários de cadastro e clicar em salvar, ao clicar em salvar, o "Administrador" que é quem realiza os cadastros é redirecionado para a página de Relatórios. Aqui, foi utilizado como exemplo o cadastro de pessoas. O processo de preenchimento do formulário e o salvar pode ser visto na Figura 72 e 73.

Figura 72 – Formulário preenchido na página de cadastro de pessoas.

CADASTRO DE PESSOAS

Nome:

Email:

Telefone:

Matrícula:

Perfil:

Fonte: A autora(2022)

Figura 73 – Nome que foi preenchido no formulário de cadastro de pessoas.

PESSOAS CADASTRADAS

Nome	E-mail	Telefone	Matrícula	Perfil	Ações
Julia Maria	julia@exemplo.com	(55) 4444-4444	201926891574	Administrador	✎ ✖
Maria Silva	maria@exemplo.com	(55) 6666-6666	201928041587	Técnico	✎ ✖
Maya Bishop	maya@gmail.com	(00) 0000-0000	202226041012	Usuários	✎ ✖
Luíza Tavares	luiza@gmail.com	(55)9 99999999	101010101010	Administrador	✎ ✖

Fonte: A autora(2022)

Para realizar uma alteração, foi necessário importar apenas o *UpdateView*, pois as outras importação feitas para o *CreateView* foram reutilizadas. O *UpdateView* é uma *view* utilizada para alterar os objetos.

Após realizar a importação, foi necessário criar no arquivo *views.py* da aplicação atividade uma classe denominada *class AtividadeUpdateView(UpdateView)*. Dentro da classe foi utilizado o *form_class* que passa o formulário base criado que é o *AtividadeModelForm*, é necessário também informar qual modelo irá ser utilizado em *model*, o nome

do *template* e o *sucess_url* que quando for executada a alteração de uma atividade, este irá redirecionar o usuário diretamente para a URL *RelatorioAtividade*. Esta classe pode ser vista na Figura 74.

Figura 74 – *class AtividadeUpdateView(UpdateView)*

```
class AtividadeUpdateView(UpdateView):
    form_class = AtividadeModelForm
    model = Atividade
    template_name = 'AtividadeForm.html'
    success_url = reverse_lazy('RelatoriosAtividades')
```

Fonte: A autora(2022)

Em seguida, é necessário criar um *path* para a nova URL. Foi criado um *path* `<int:pk>/Atividade/editar/`, chamando a view *AtividadeUpdateView* e o nome é *atividade.editar*. Quando for realizar uma alteração é preciso de um valor inteiro, que aqui é chamado de *pk* que corresponde a *Primary Key* dentro da classe. Este processo pode ser visto na Figura 75.









Figura 75 – *path* no arquivo *urls.py*.

```
urls.py
1 from django.urls import path
2 from .views import AtividadeView, AtividadeAddView, AtividadeUpdateView, AtividadeDeleteView
3
4 urlpatterns = [
5     path('RelatoriosAtividades', AtividadeView.as_view(), name='RelatoriosAtividades'),
6     path('Atividade/adicionar/', AtividadeAddView.as_view(), name='atividade.adicionar'),
7     path('<int:pk>/Atividade/editar/', AtividadeUpdateView.as_view(), name='atividade.editar'),
8     path('<int:pk>/Atividade/apagar/', AtividadeDeleteView.as_view(), name='atividade.apagar'),
9 ]
```

Fonte: A autora(2022)

Para realizar uma alteração, é necessário acessar a página de um relatório, ao abrir basta clicar no ícone para ser redirecionada para página do formulário para realizar a alteração. O ícone que faz o redirecionamento para a página de formulário pode ser visto na Figura 76.








Figura 76 – Ícone para realizar alteração.

Ações
 
 
 
 

Fonte: A autora(2022)

Ao clicar sobre o ícone, o "Administrador" será redirecionado para a página de formulário e assim poderá realizar as alterações que desejar e após isso basta clicar em salvar. Este processo pode ser visto na Figura 77.

Figura 77 – Alteração de Informações.

PESSOAS CADASTRADAS					
Nome	E-mail	Telefone	Matrícula	Perfil	Ações
Julia Maria	julia@exemplo.com	(55) 4444-4444	201926891574	Administrador	 
Maria Silva	maria@exemplo.com	(55) 6666-6666	201928041587	Técnico	 
Maya Bishop	maya@gmail.com	(00) 0000-0000	202226041012	Usuários	 
Luíza Tavares	luiza@gmail.com	(55)9 99999992	101010101010	Usuários	 

Fonte: A autora(2022)

Em seguida, foi necessário criar no arquivo *views.py* da aplicação atividade uma classe denominada *class AtividadeDeleteView(DeleteView)*. Dentro da classe foi necessário informar qual modelo irá ser utilizado em *model*, o nome do *template* e o *sucess_url* que quando for executada a inclusão de uma nova atividade, este irá redirecionar o usuário diretamente para a *URL RelatorioAtividade*. Esta classe pode ser vista na Figura 78.

Figura 78 – *class AtividadeDeleteView(DeleteView)*

```
class AtividadeDeleteView(DeleteView):
    model = Atividade
    template_name = 'atividade.apagar.html'
    success_url = reverse_lazy('RelatoriosAtividades')
```

Fonte: A autora(2022)

Após, é necessário criar um *path* para a nova URL. Foi criado o *path* `<int:pk>/Atividade/apagar/`, chamando a *view* *AtividadeDeleteView* e o nome é *atividade.apagar*. Quando for realizar uma exclusão é preciso de um valor inteiro, que aqui é chamado de *pk*, este é utilizado para localizar o registro no banco de dados e apagá-lo. Este processo pode ser visto na Figura 79.

Figura 79 – *path* no arquivo *urls.py*.

```
urls.py
1 from django.urls import path
2 from .views import AtividadeView, AtividadeAddView, AtividadeUpdateView, AtividadeDeleteView
3
4 urlpatterns = [
5     path('RelatoriosAtividades', AtividadeView.as_view(), name='RelatoriosAtividades'),
6     path('Atividade/adicionar/', AtividadeAddView.as_view(), name='atividade.adicionar'),
7     path('<int:pk>/Atividade/editar/', AtividadeUpdateView.as_view(), name='atividade.editar'),
8     path('<int:pk>/Atividade/apagar/', AtividadeDeleteView.as_view(), name='atividade.apagar'),
9 ]
```

Fonte: A autora(2022)

Para realizar uma alteração, é necessário acessar a página de um relatório, ao abrir basta clicar no ícone para ser redirecionada para página do formulário para realizar a alteração. O ícone que faz o redirecionamento para a página de formulário pode ser visto na Figura 80.

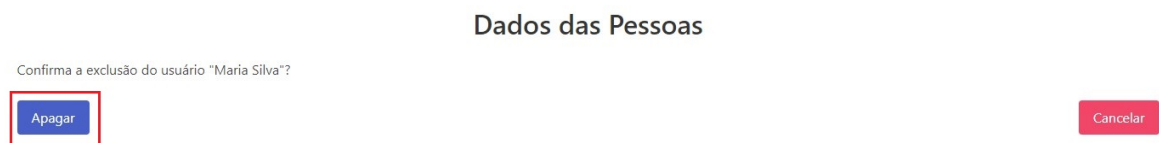
Figura 80 – Ícone para realizar exclusão.



Fonte: A autora(2022)

Ao clicar sobre o ícone, o "Administrador" será redirecionado para a página de exclusão, onde conterà o nome do dado que o "Administrador" deseja excluir, após isso basta clicar em apagar e o dado será apagado. Este processo pode ser visto na Figura 81.

Figura 81 – Exclusão de um dado.



Fonte: A autora(2022)

Após realizar todas estas etapas o *CRUD* do sistema desenvolvido neste estudo esta concluído.

3.3.1 Migração do SQLite para PostgreSQL

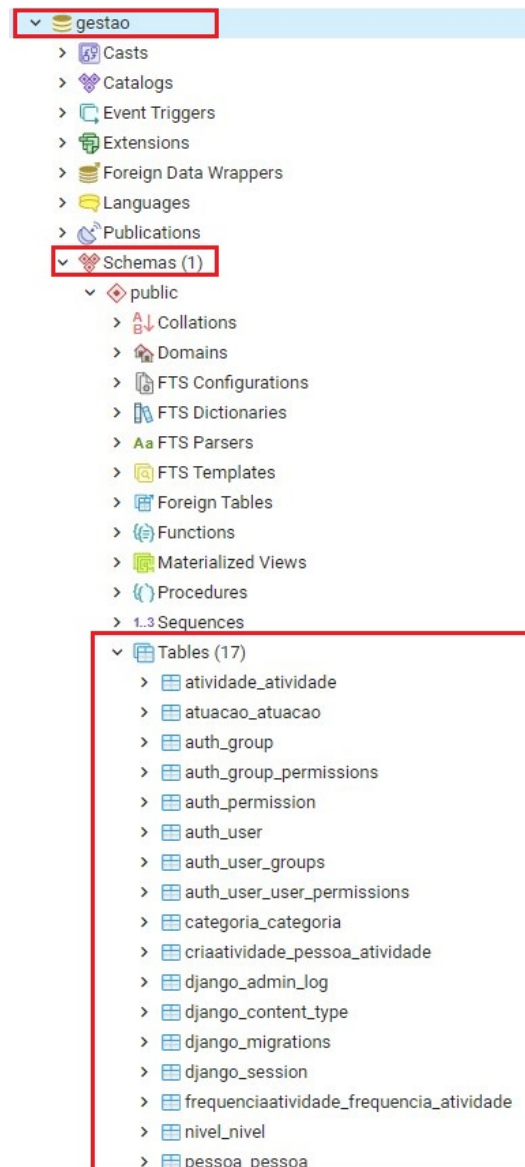
Para realizar a migração da aplicação desenvolvida neste estudo para o *postgreSQL*, é preciso inicialmente realizar a instalação do *end in* do *postgreSQL* para isso utiliza-se o comando *pip install psycpg2* e em seguida é necessários atualizar o arquivo *requirements.txt* com o comando *pip freeze > requirements.txt*. Após isso, é necessário acessar o *postgreSQL* e criar uma nova base de dados. Assim que criada a base de dados é preciso

ir ao arquivo *settings.py* da aplicação e ir pra dentro do *DATABASES* e alterá-lo.

O *ENGINE* passa a ter acesso ao *postgreSQL*, o *NAME* se refere ao nome da base de dados criada no *postgreSQL*, *USER* e *PASSWORD* se referem ao nome usuário e a senha definida na instalação do *postgreSQL*. Após isso, é necessário acessar o projeto e ir até a pasta *migrations* de cada aplicação e apagar todos os arquivos existentes nessa pasta com exceção do arquivo *__init__.py*. Em seguida pode ser feita também a exclusão do arquivo *db.sqlite3*.

Após isso, é necessário executar o comando *python manage.py makemigrations* e o *python manage.py migrate*, com isso, toda a base de dados será recriada com todos os modelos no *postgreSQL*. A base de dados da aplicação desenvolvida neste estudo pode ser vista na Figura 82.

Figura 82 – Base de dados da aplicação.



Fonte: A autora(2022)

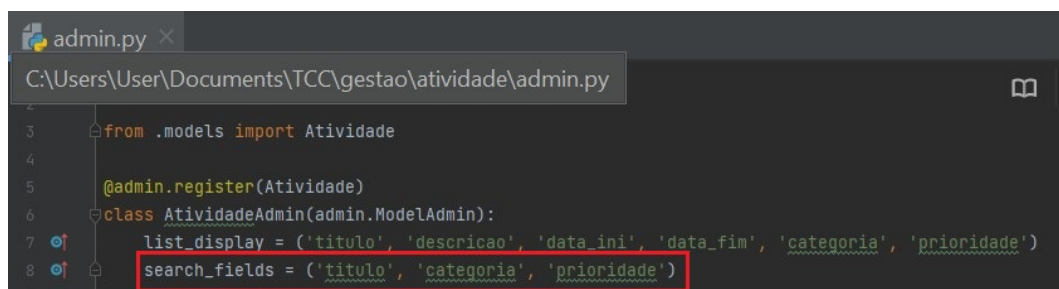
Após realizar a migração da aplicação para o *postgreSQL*, é necessário criar um novo super usuário com o comando *python manage.py createsuperuser*, pois as tabelas que continham o super usuário foram excluídas.

3.3.2 Problemas no Desenvolvimento

Ao longo do processo de desenvolvimento da aplicação que se desenvolveu com este estudo, ocorreram alguns erros, alguns deles ocorreram com mais recorrência, como por exemplo, a realização de buscas.

Em todos os relatórios é possível a realização de buscas, deve-se prestar muita atenção no arquivo *admin.py*, pois é nele que é adicionado o *search_fields*, utilizado para criar campos de busca para objetos na interface, dentro dele deve-se definir os campos de busca. Deve-se tomar cuidado na hora de definir estes campos pois eles devem possuir a mesma nomenclatura que foi definida dentro do modelo, ou seja, no arquivo *models.py* da aplicação. Caso alguma destas nomenclaturas não coincidirem com o que está no arquivo *models.py*, ao clicar em "buscar", ocorrerá um erro e não será feita a busca. Na Figura 83 pode ser vista a definição do *search_fields* na aplicação *atividade*.

Figura 83 – Definição do *search_fields* na aplicação *atividade*.



Fonte: A autora(2022)

Além disso, no arquivo *views.py*, deve-se definir a classe que será utilizada, onde esta será uma subclasse da *ListView*, deve-se em seguida definir o modelo da classe e o nome do *template*, deve-se atentar aqui para que o nome do *template* esteja correto, caso esteja errado, ocorrerá um erro de busca.

Para realizar a busca utiliza-se o método *QuerySet*, dentro do *QuerySet*, deve-se definir uma variável chamada "busca" igual a `self.request.GET.get('buscar')`, ou seja, ele utiliza o valor da variável "buscar" que vem do *HTML* e armazena dentro da variável "buscar" e neste caso defini-se uma *QuerySet* que irá utilizar o que já existe na superclasse que é a própria *ListView* fazendo com que ele busque o que já existe dentro deste método. Além disso, aplica-se sobre este *QuerySet* um filtro, testando se dentro da variável "buscar"

existe algum valor igual, ou seja, o que foi buscado pelo usuário, caso exista algum valor, este valor será retornado para o usuário que realizou a busca, este processo pode ser visto na Figura 84.

No filtro de busca, é necessário ficar atento ao que será buscado dentro da interface para passar a nomenclatura correta dentro da *QuerySet*, como por exemplo na aplicação *atividade* onde a busca é feita pelo título da atividade então utiliza-se `qs = qs.filter(titulo__icontains=buscar)`. O arquivo *views.py* com a definição da *QuerySet* e o filtro, podem ser vistos na Figura 84.

Figura 84 – Arquivo *views.py* com a definição da *QuerySet* e o filtro.

```
class AtividadeView(ListView):
    model = Atividade
    template_name = 'RelatoriosAtividades.html'

    def get_queryset(self, *args, **kwargs):
        buscar = self.request.GET.get('buscar')
        qs = super(AtividadeView, self).get_queryset(*args, **kwargs)
        if buscar:
            qs = qs.filter(titulo__icontains=buscar)

        return qs
```

Fonte: A autora(2022)

Para o campo de busca ser exibido no sistema é necessário primeiro adicioná-lo ao arquivo *admin.py* da aplicação em que deseja realizar buscas com a linha de código *search_fields* e quais são atributos da classe que podem ser pesquisados. A Figura 85 mostra o arquivo *admin.py* a linha de código *search_fields* e a Figura 86 mostra o campo de busca da página administrativa do *Django*.

Figura 85 – Arquivo *admin.py*.

```
admin.py
1 from django.contrib import admin
2
3 from .models import Atividade
4
5 @admin.register(Atividade)
6 class AtividadeAdmin(admin.ModelAdmin):
7     list_display = ('titulo', 'descricao', 'data_ini', 'data_fim', 'categoria', 'prioridade')
8     search_fields = ('titulo', 'categoria', 'prioridade')
```

Fonte: A autora(2022)

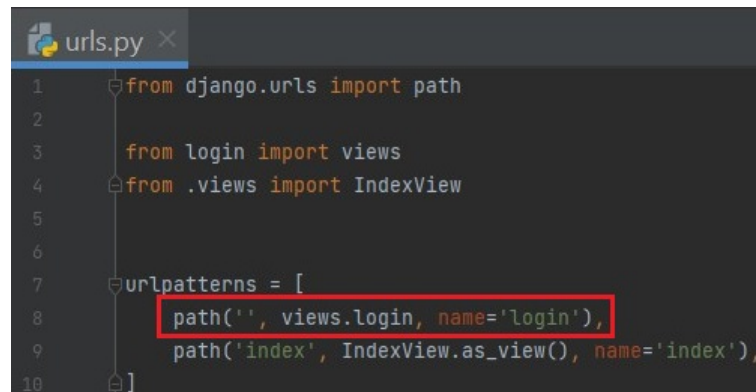
Figura 86 – Campo de busca da página administrativa do *Django*.

A imagem mostra uma interface web com o título "Selecione Atuação para modificar". Abaixo do título, há um campo de busca com um ícone de lupa à esquerda e o botão "Pesquisar" à direita. Este campo de busca está circulado por um retângulo vermelho. Abaixo do campo de busca, há uma seção com o rótulo "Ação:" seguida de um menu suspenso e o botão "Ir". À direita desta seção, está o texto "0 de 3 selecionados". Abaixo disso, há uma tabela com o cabeçalho "DESCRIÇÃO" e três linhas de dados, cada uma com um ícone de seleção (quadrado) e o texto "Descrição da atuação três", "Descrição da atuação dois" e "Descrição da atuação um". No rodapé da tabela, está o texto "3 Atuações".

Fonte: A autora(2022)

Outro problema que ocorreu no processo de desenvolvimento deste estudo, é que para retornar à página inicial foi utilizado a *url 'index'* no título do site, porém quando estava em outra página, como por exemplo, relatórios atividades, ao clicar em “Laboratório de Pesquisa da UFSM” ao invés de redirecionar de relatório para página inicial, o sistema redirecionava para a página de *login* mesmo o usuário já estando logado no sistema. Para que o sistema fosse redirecionado para página inicial ao invés da página de *login* foi necessário alterar o arquivo *urls.py* da aplicação *home*, que é a aplicação principal do sistema. Este processo pode ser visto na Figura 87. Neste caso foi definido uma rota para o *login*, para diferenciar qual rota esta sendo solicitada, sendo assim, ao clicar em “Laboratório de Pesquisa da UFSM” ou em “Início” quando estiver em outra página, como por exemplo, em algum dos relatórios, o usuário será redirecionado para a página inicial e não para página de *login*.

Figura 87 – Arquivo *urls.py* da aplicação *home* com a definição da rota de *login*.



```

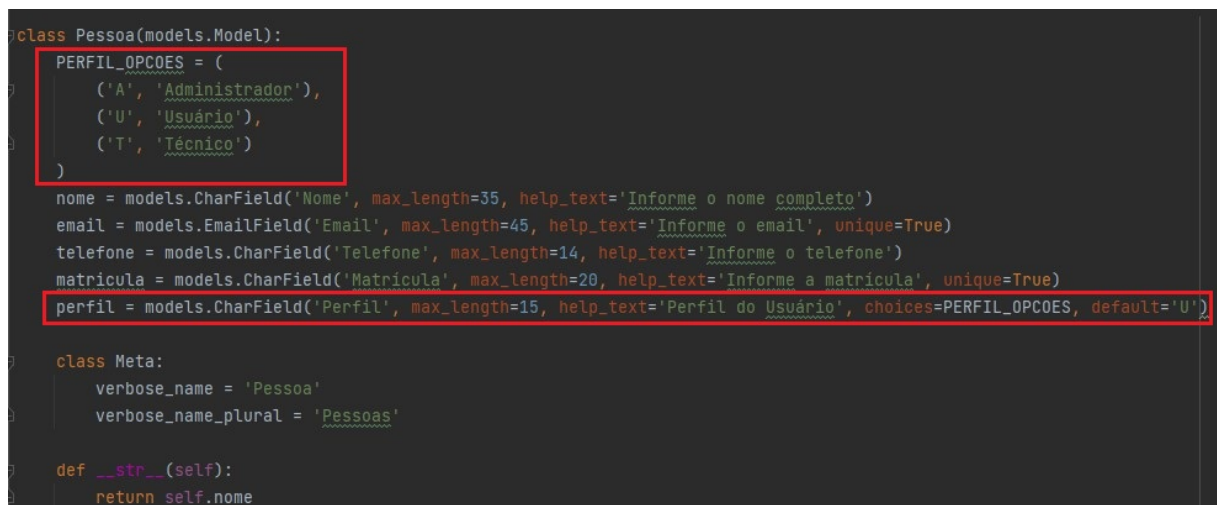
1 from django.urls import path
2
3 from login import views
4 from .views import IndexView
5
6
7 urlpatterns = [
8     path('', views.login, name='login'),
9     path('index', IndexView.as_view(), name='index'),
10 ]

```

Fonte: A autora(2022)

Em algumas classes dentro da aplicação desenvolvida neste estudo, possuem opções para os usuários escolherem, como pode ser visto na Figura 88.

Figura 88 – Arquivo *views.py* da aplicação *peessoa* com as opções de perfil.



```

class Pessoa(models.Model):
    PERFIL_OPCOES = (
        ('A', 'Administrador'),
        ('U', 'Usuário'),
        ('T', 'Técnico')
    )
    nome = models.CharField('Nome', max_length=35, help_text='Informe o nome completo')
    email = models.EmailField('Email', max_length=45, help_text='Informe o email', unique=True)
    telefone = models.CharField('Telefone', max_length=14, help_text='Informe o telefone')
    matricula = models.CharField('Matricula', max_length=20, help_text='Informe a matricula', unique=True)
    perfil = models.CharField('Perfil', max_length=15, help_text='Perfil do Usuário', choices=PERFIL_OPCOES, default='U')

    class Meta:
        verbose_name = 'Pessoa'
        verbose_name_plural = 'Pessoas'

    def __str__(self):
        return self.nome

```

Fonte: A autora(2022)

Além disso, para que fosse possível que o nome das opções fossem exibidos por completo na interface e não somente a sua inicial, foi necessário definir isto dentro do *template* responsável por carregar este nome, ou seja, neste caso nas tabelas onde ficam todos os nomes cadastrados que são os *templates* de relatórios. Aqui no caso será utilizado como exemplo o *template* de "Relatórios de Pessoas". No campo onde trás o nome de pessoa ao invés de utilizar apenas *RelatoriosPessoas.perfil* que é o nome de que foi

dado para as opções, deve-se utilizar o `RelatoriosPessoas.get_perfil_display`, com esta alteração já seria possível exibir o nome completo das opções de perfil dentro da interface. Além disso, pode-se também utilizar linhas de código, esta foi a forma utilizada neste estudo. o *template RelatoriosPessoas.html* da aplicação *pessoa*, mostrando como deve ser definida para apresentar o nome completo dentro da interface pode ser visto na Figura 89.

Figura 89 – O *template RelatoriosPessoas* da aplicação *pessoa*.

```
<td>{{ RelatoriosPessoas.nome }}</td>
<td>{{ RelatoriosPessoas.email }}</td>
<td>{{ RelatoriosPessoas.telefone }}</td>
<td>{{ RelatoriosPessoas.matricula }}</td>
<td>
    {% if RelatoriosPessoas.perfil == 'A' %}
        {{ 'Administrador' }}
    {% elif RelatoriosPessoas.perfil == 'T' %}
        {{ 'Técnico' }}
    {% else %}
        {{ 'Usuários' }}
    {% endif %}
</td>
```

Fonte: A autora(2022)

Quando a opção selecionada for "A" será exibido o nome de "Administrador", se a opção for "T" será exibido o nome de "Técnico" e se não for nenhuma das opções, o nome exibido será de "Usuário".

Na interface de "criar atividade", é possível adicionar um arquivo no campo denominado como "arquivo", porém além de adicionar um arquivo em "criar atividade", é necessário conseguir realizar o *download* deste arquivo nos relatórios das atividades porém isto não era possível pois quando adicionado o arquivo e salvo, dentro do diretório "gestao", local onde ficam as aplicações desenvolvidas neste estudo, aparecia uma pasta chamada *uploads* e dentro dessa pasta um arquivo em formato *.py*, com o nome no arquivo que foi adicionado a interface "criar atividade". Para que fosse possível realizar o *download* deste arquivo quando acessasse a página de relatórios de atividades foi necessário alterar a linha responsável por exibir este arquivo na interface. Este processo pode ser visto na Figura 90.

Figura 90 – apresenta o *template RelatoriosPessoas* da aplicação *pessoa*, mostrando como deve ser definida apresentar o nome completo dentro da interface

```
<td><a href="{ RelatoriosPessoasAtividades.arquivo.url}" download>Arquivo</a></td>
```

Fonte: A autora(2022)

Ao invés de utilizar *RelatoriosPessoasAtividades.arquivo* utiliza-se "*RelatoriosPessoasAtividades.arquivo.url*" *download* dentro de um *<a>* que se atributo ao *href* cria-se um *link* que leva para outras páginas dentro do sistema desenvolvido ou para outro *site*.

As páginas de cadastro de pessoas e cadastro de *login* de pessoas, não são unificadas. Estas são páginas independentes, para trabalhos futuros utilizando o *Framework* de *back-end Django*, seria interessante unificar estas páginas, assim seria possível realizar o cadastro de um usuário no sistema e ao mesmo tempo cadastrar o *login* para que o usuário tenha acesso ao sistema na mesma página. A unificação destas páginas não foi realizada pois demandaria mais tempo de estudo. A Figura 91 e 92 mostram as páginas de cadastro de pessoas e cadastro de login de pessoas.

Figura 91 – Cadastro de pessoas

CADASTRO DE PESSOAS

Nome:

Email:

Telefone:

Matrícula:

Perfil:

Fonte: A autora(2022)

Figura 92 – Cadastro de *login* de pessoas

O formulário, intitulado "CADASTRAR LOGIN DO USUÁRIO", está contido numa caixa azul. Ele possui os seguintes campos e elementos:

- Nome Completo:** Um campo de texto com o valor "João" e um ícone de pessoa à esquerda.
- E-mail:** Um campo de texto com o valor "exemplo@gmail.com" e um ícone de envelope à esquerda.
- Informações de Login:** Um bloco amarelo contendo o texto: "A matrícula cadastrada aqui será usada no campo de identificação do usuário na hora de acessar a sua conta."
- Matrícula:** Um campo de texto com o valor "202224081019" e um ícone de pessoa à esquerda.
- Senha:** Um campo de texto com um ícone de cadeado à esquerda.
- Confirmar Senha:** Um campo de texto com um ícone de cadeado à esquerda.
- Botão:** Um botão azul no rodapé com o texto "Cadastrar Usuário".

Fonte: A autora(2022)

Outra tarefa que seria interessante realizar futuramente ao utilizar o *Framework* de *back-end Django*, é tornar possível a definição de tipo de usuário como: "Administrador", "Técnico" e "Usuário" na hora de realizar o cadastro de *login* de pessoas. Atualmente só é possível definir o tipo de usuário na página administrativa do *Django*. A definição de tipo de usuário na hora do cadastro de *login* de pessoas não foi realizada pois demandaria mais tempo de estudo. A Figura 92 mostra a página de cadastro de login de pessoas.

4 CONCLUSÃO

Baseado no que foi apresentado ao longo deste estudo, é possível perceber que a utilização de *Frameworks* facilita muito o desenvolvimento de sistemas web. Com o uso de *Frameworks* é possível realizar tarefas que antes eram consideradas muito complexas, tornando-as mais fácil desde o momento de sua realização até a sua compreensão.

A utilização dos *Frameworks Bulma CSS* e *Django* fornecem ao desenvolvedor web uma grande economia de tempo, pois tarefas que demorariam dias para serem realizadas podem ser concluídas em poucos minutos ou horas, cabendo ao desenvolvedor web utilizá-los da maneira mais adequada no seu projeto.

Foram cumpridos todos os objetivos propostos neste estudo, sendo eles, estudar sobre os *Frameworks Bulma CSS* e *Django* utilizando a linguagem de programação *Python*, analisar o funcionamento de cada *Framework*, aplicar o estudo do Framework de front-end Bulma CSS para a criação das interfaces do sistema web desenvolvido neste estudo e aplicar o estudo do Framework de back-end Django utilizando a linguagem de programação *Python* para criação de uma aplicação para o controle de horários, atividades e pessoas realizado no curso Técnico em Informática do Colégio Politécnico da Universidade Federal de Santa Maria (UFSM);

Além disso, o presente estudo constituiu-se em uma pesquisa de caráter descritivo visando descrever a utilização dos *Frameworks Bulma CSS* e *Django* e o processo de desenvolvimento da aplicação que se desenvolveu neste estudo.

Para futuras pesquisas sobre o tema, sugerem-se abordagens que foquem no desenvolvimento web, visto que muitas dos *Frameworks* pesquisados fornecem uma grande economia de tempo gerando grande produtividade aos desenvolvedores.

REFERÊNCIAS BIBLIOGRÁFICAS

ALURA. **Saiba tudo sobre o IDE - Integrated Development Environment**. 2022. <<https://www.alura.com.br/artigos/o-que-e-uma-ide>>. [Online; acessado em 20-Outubro-2022].

BACK, J. **Utilizando pip freeze corretamente**. 2020. <<https://jozimarback.medium.com/utilizando-pip-freeze-corretamente-f9a305c691c0>>. [Online; acessado em 12-Dezembro-2022].

BOOCH, G. **UML: guia do usuário**. [S.l.]: Elsevier Brasil, 2006.

BOOTSTRAP. **Build fast, responsive sites with Bootstrap**. 2022. <<https://getbootstrap.com/>>. [Online; acessado 03-March-2022].

BULMA. **Bulma: the modern CSS framework that just works.** 2022. <<https://bulma.io/>>. [Online; acessado 05-Março-2022].

BULMA, D. **Documentation: Everything you need to create a website with Bulma.** 2021. <<https://bulma.io/documentation/>>. [Online; acessado 23-Dezembro-2021].

BYLEARN. **7 Aplicativos feitos com Python que você com certeza usa no seu dia a dia**. 2020. <<https://dojo.bylearn.com.br/python/7-aplicativos-feitos-com-python/>>. [Online; accessed 8-Janeiro-2022].

CAMARGO, V. V. de; MASIERO, P. C. Frameworks orientados a aspectos. **Anais do 19o Simpósio Brasileiro de Engenharia de Software (SBES2005)**, p. 3, 2005.

CODE, V. S. **IntelliSense**. 2022. <<https://code.visualstudio.com/docs/editor/intellisense>>. [Online; acessado em 03-Novembro-2022].

COODESH. **O que é CRUD?** 2022. <<https://coodesh.com/blog/dicionario/o-que-e-crud/>>. [Online; acessado em 09-Dezembro-2022].

DEVMEDIA. O que é front-end e back-end? 2009. <[https://www.devmedia.com.br/artigo-engenharia-de-software-10-documento-de-requisitos/11909#:~::~%20texto%20%20documento%20de%20%20requisitos%20%20delimita,qualidade%20que%20%20devem%20ser%20%20suportados.](https://www.devmedia.com.br/artigo-engenharia-de-software-10-documento-de-requisitos/11909#:~:%20texto%20%20documento%20de%20%20requisitos%20%20delimita,qualidade%20que%20%20devem%20ser%20%20suportados.)> [Online; acessado em 24-Setembro-2022].

____. **Modelagem de dados: 1:N ou N:N?** 2022. <<https://www.devmedia.com.br/modelagem-1-n-ou-n-n/38894>>. [Online; acessado em 16-Novembro-2022].

DEVOPS, M. **O Que É Django (Python) E Como Usar No Desenvolvimento Web**. 2020. <<https://mundodevops.com/blog/django-python/>>. [Online; acessado 05-Março-2022].

DJANGO. **Django**. 2022. <<https://www.djangoproject.com/>>. [Online; acessado 06-Dezembro-2022].

_____. **Django num relance**. 2022. <<https://django-portuguese.readthedocs.io/en/1.0/intro/overview.html>>. [Online; acessado 06-Dezembro-2022].

_____. **Documentação- Model**. 2022. <<https://docs.djangoproject.com/en/4.1/topics/db/models/>>. [Online; acessado em 08-Dezembro-2022].

_____. **O site admin do Django**. 2022. <<https://django-portuguese.readthedocs.io/en/1.0/ref/contrib/admin.html>>. [Online; acessado em 31-Outubro-2022].

_____. **Why does this project exist?** 2022. <<https://docs.djangoproject.com/en/4.1/faq/general/#why-does-this-project-exist>>. [Online; acessado 19-Janeiro-2022].

DJANGO, D. **Escrevendo seu primeiro aplicativo Django, parte 2**. 2022. <<https://docs.djangoproject.com/en/4.1/intro/tutorial02/>>. [Online; acessado em 12-Dezembro-2022].

FLASK. **User's Guide**. 2022. <<https://flask.palletsprojects.com/en/2.2.x/>>. [Online; acessado em 27-Outubro-2022].

GUEDES, G. T. Uml 2. **Uma Abordagem Prática**", São Paulo, Novatec, 2009.

_____. **UML 2-Uma abordagem prática**. [S.l.]: Novatec Editora, 2018.

HAMMOND, M.; ROBINSON, A. **Python programming on win32: Help for windows programmers**. [S.l.]: "O'Reilly Media, Inc.", 2000. 1–3 p.

HOSTGATOR. **Editor de código: confira as 10 melhores opções**. 2021. <<https://www.hostgator.com.br/blog/melhores-opcoes-editor-de-codigo/>>. [Online; acessado em 20-Janeiro-2022].

ISOLUTION. **Bulma Tutorial**. 2022. <<https://isolution.pro/pt/t/bulma/bulma-quick-guide/bulma-guia-rapido>>. [Online; acessado 30-Dezembro-2021].

JETBRAINS. **Get started**. 2022. <<https://www.jetbrains.com/help/pycharm/2021.3/quick-start-guide.html>>. [Online; acessado em 07-Março-2022].

_____. **Recursos do PyCharm**. 2022. <<https://www.jetbrains.com/pt-br/pycharm/features/>>. [Online; acessado em 03-Novembro-2022].

LABORATORIO, C. de. **O modelo MTV no Django**. 2021. <<https://cadernodelaboratorio.com.br/o-modelo-mtv-no-django/>>. [Online; acessado em 20-Janeiro-2022].

LARAVEL. **Meet Laravel**. 2022. <<https://laravel.com/docs/9.x/installation>>. [Online; acessado em 27-Outubro-2022].

MANZANO, J. A. N. **Introdução à linguagem Python**. [S.l.]: Novatec Editora, 2018.

MATERIALIZIZE. **Getting Started**. 2022. <<https://materializecss.com/getting-started.html>>. [Online; acessado 05-Março-2022].

MENDONÇA, R. A. R. de. Levantamento de requisitos no desenvolvimento ágil de software. **Semana da Ciência e Tecnologia da PUC Goiás**, p. 12, 2014.

MOZILLA. **Tutorial Django Parte 4: Site de administração do Django**. 2022. <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django/Admin_site#registrando_uma_classe_modeladmin>. [Online; acessado em 08-Dezembro-2022].

_____. **Tutorial Django Parte 9: Trabalhando com formulários**. 2022. <<https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django/Forms>>. [Online; acessado em 31-Outubro-2022].

PIP. **Getting Started**. 2022. <<https://pip.pypa.io/en/stable/getting-started/>>. [Online; acessado em 31-Outubro-2022].

PYTHON, D. **Why was Python created in the first place?** 2022. <<https://docs.python.org/3/faq/general.html>>. [Online; acessado 11-Janeiro-2022].

PYTHON, L. **History and License**. 2022. <<https://docs.python.org/3/license.html>>. [Online; acessado 28-Fevereiro-2022].

PYTHON, S. **sqlite3— Interface DB-API 2.0 para bancos de dados SQLite**. 2022. <<https://docs.python.org/3/library/sqlite3.html>>. [Online; acessado 12-Dezembro-2022].

RAMOS, P. A. V. **Desenvolvimento Web com Python e Django**. [S.l.]: Artmed editora, 2018.

ROSSUM, G. V.; DRAKE, F. L. **An introduction to Python**. [S.l.]: Network Theory Ltd. Bristol, 2003.

RUBY. **Getting Started**. 2022. <<https://ruby-doc.org/>>. [Online; acessado em 27-Outubro-2022].

STUDYBAY. **O que é pesquisa descritiva: conceito, tipos e características**. 2022. <<https://mystudybay.com.br/pesquisa-descritiva/?ref=e49b1b78b89220fa>>. [Online; acessado em 27-Outubro-2022].

TREINAWEB, A. **O que é uma API?** 2020. <<https://www.treinaweb.com.br/blog/o-que-e-uma-ide-ambiente-de-desenvolvimento-integrado>>. [Online; acessado 20-Janeiro-2022].

TREINAWEB app. **Criando um app com Django**. 2020. <<https://www.treinaweb.com.br/blog/criando-um-app-com-django#:~:text=apps.py%3A%20Arquivo%20respons%C3%A1vel%20pela,regas%20de%20testes%20da%20aplica%C3%A7%C3%A3o.>>>. [Online; acessado 31-Outubro-2022].

TREINAWEB, I. **O que é uma IDE (Ambiente de Desenvolvimento Integrado)?** 2020. <<https://www.treinaweb.com.br/blog/o-que-e-uma-ide-ambiente-de-desenvolvimento-integrado>>. [Online; acessado 20-Janeiro-2022].

TREINAWEB, V. **VS Code - O que é e por que você deve usar?** 2021. <<https://www.treinaweb.com.br/blog/vs-code-o-que-e-e-por-que-voce-deve-usar>>. [Online; acessado 20-Janeiro-2022].

TRYBE. **Framework: o que é, como ele funciona e para que serve?** 2020. <<https://blog.betrybe.com/framework-de-programacao/o-que-e-framework/>>. [Online; acessado 02-Março-2022].

VUE.JS. **The Progressive JavaScript Framework**. 2022. <<https://vuejs.org/>>. [Online; acessado 05-Março-2022].