# AUTOMATIZANDO PROCESSOS COM DJANGO-FULL-CRUD
# AUTOMATING PROCESSES WITH DJANGO-FULL-CRUD

Francisco Flávio Nogueira da Silva
UFERSA
Pau dos Ferros - RN +55
(84) 98189-5788
francisco.silva29241@alunos.ufersa.edu.br

Alysson Filgueira Milanez
UFERSA
Pau dos Ferros - RN
+55 (83) 99928-7821
alysson.milanez@ufersa.edu.br

## RESUMO

Automações de processos são sempre bem-vistas no mercado de desenvolvimento de software por conseguirem auxiliar no cumprimento dos prazos estabelecidos, na entrega de um produto de qualidade; tudo isso sem o esforço de implementar 100% de cada sistema. Desta forma, no presente trabalho apresenta-se um pacote para a linguagem de programação Python e o framework Django; o mesmo visa automatizar os processos de create, read, update e delete (CRUD) para sistemas web, sejam eles monolíticos ou não. Como resultado, obteve-se um pacote simples de ser utilizado, que necessita de apenas alguns comandos para o seu bom funcionamento; porém, com a capacidade de gerar arquivos (admins, forms, views, viewsets, serializers e urls) presentes no dia a dia dos desenvolvedores web que trabalham com essa tecnologia.

## Palavras-chave

Pacote; ferramenta; automatizar; Django.

## ABSTRACT

Process automations are always well regarded in the software development market because they can help meet deadlines, deliver a quality product; all without the effort of implementing 100% of each system. Thus, in the present work, a package for the Python programming language and the Django framework is presented; it aims to automate creation, reading, updating and deleting (CRUD) processes for web systems, whether they are monolithic or not. As a result, a package that is simple to use was obtained, which requires only a few commands for its proper functioning; however, with the ability to generate files (admins, forms, views, viewsets, serializers and urls) it is present in the daily lives of web developers who work with this technology.

## Keywords

Package; tool; automate; Django.



## CCS Concepts

•Software and its engineering → Development frameworks and environments.

## 1. INTRODUÇÃO

Atualmente existem diversos frameworks para inúmeras plataformas e linguagens de programação, dentre eles, o *Django* se destaca por conseguir entregar um desenvolvimento simples e poderoso para aplicações web feitas em Python [3]. Segundo a documentação do *Django*, "o *Django* facilita a criação de melhores aplicativos da *Web* com mais rapidez e menos código" [3].

A depender do nível de senioridade, experiência e aptidão para a programação, as pessoas conseguem perceber os padrões de desenvolvimento usados no *framework*, os arquivos que precisam ser criados, as configurações necessárias para cada novo projeto e os problemas parecidos que possam surgir ao longo do tempo.

De acordo com Maciel [5], o princípio de Pareto diz que 80% dos efeitos gerados surgem a partir de apenas 20% das causas. Se isso for aplicado para o desenvolvimento de projetos de software, tem-se que 80% das funcionalidades do sistema só são usadas por 20% dos usuários, ou seja, 80% dos usuários usam apenas 20% do software que foi desenvolvido. Tendo isso em vista, o *Django Full Crud* buscará automatizar uma parte desse desenvolvimento, fazendo com que os desenvolvedores possam focar 80% do seu trabalho em 20% das funcionalidades que gerarão mais resultados para o cliente e para a empresa.

Com isso, após se perceber a semelhança entre projetos, foi desenvolvida uma automação que permite ao desenvolvedor criar arquivos geralmente usados no desenvolvimento de um sistema ou de uma *Application Programming Interface (API)*, dentre eles: *admins, forms, serializers, templates, views, viewsets e urls*, tendo a possibilidade de escolha entre quais arquivos serão criados. Auxiliando na eficácia do código escrito e minimizando as chances de ocorrência de erros humanos que poderiam surgir durante o desenvolvimento.

E importante elucidar que o pacote pode ser um malefício para alguns desenvolvedores que estão no início de sua carreira, afinal, o mesmo cria de maneira automática códigos que outrora seriam escritos manualmente. Usar a automação

de maneira negligente pode acarretar prejuízos ao aprendizado e também diminuir a capacidade do desenvolvedor de produzir seus próprios códigos de maneira autônoma e original, sendo recomendado um certo nível de senioridade que não cabe ao presente trabalho informá-lo. Sendo assim, examine-se, pois, o homem a si mesmo [...] [1], e decida se lhe é conveniente usar o pacote ora proposto, *Django Full Crud*.

O presente artigo contribui com a comunidade acadêmica, especialmente na área da automação de processos. Possui potencial para melhorar a velocidade com que os desenvolvedores *web* (exclusivamente os que trabalham com a linguagem de programação *Python* e o *framework Django*) fazem o seu trabalho, além de conseguir ajudar entidades que desenvolvem softwares, fazendo com que o fluxo de trabalho seja mais otimizado.

O trabalho está organizado da seguinte maneira: na seção 2, apresenta-se o embasamento teórico necessário para a compreensão do artigo. A seção 3, por sua vez, mostra os trabalhos relacionados. Na seção 4, a metodologia usada para o desenvolvimento do pacote é discutida. Na seção 5, está contemplada a abordagem. Por fim, na seção 6, são apresentadas as considerações finais do trabalho e prospectos para trabalhos futuros.

## 2. FUNDAMENTAÇÃO TEÓRICA

A presente seção, irá apresentar os pontos necessários para a compreensão do presente trabalho. Serão explanados os seguintes conteúdos: Conhecimentos básicos sobre padrões de projeto (2.1), requisitos (2.2), paradigmas de programação (2.3), programação orientada a objetos (2.4), a linguagem de programação *Python* (2.5), desenvolvimento *web* (2.6), o *framework Django* (2.7), sobre *CRUD's* (2.8) e sobre banco de dados (2.9).

### 2.1 PADRÕES DE PROJETO

Você já ouviu a expressão "reinventar a roda"? E justa-´ mente isso que os padrões de projeto buscam evitar. Pressman e Maxim [8] informam que o projeto baseado em padrões cria uma aplicação por meio da busca de um conjunto de soluções comprovadas para um conjunto de problemas claramente delineados. Cada problema (e sua solução) é descrito por um padrão de projeto, que foi catalogado e investigado por outros engenheiros de software [...].

O *Django Full Crud* parte do princípio formado pelos padrões, ele visa criar um modelo inicial que seja genérico o suficiente para dar um pontapé inicial a algum projeto web, permitindo que os desenvolvedores consigam ser mais práticos e ágeis em seu trabalho.

### 2.2 REQUISITOS

Reinehr [9] define os requisitos em três partes distintas, sendo:

1. Uma condição ou capacidade necessária para um usuário resolver um problema ou alcançar um objetivo.
2. Uma condição ou capacidade que deve ser atendida ou tida por um sistema ou componente do sistema para satisfazer a um contrato, padrão, especificação ou outro documento formalmente imposto.
3. Uma representação documentada de uma condição ou capacidade conforme estabelecido em 1 e 2.

Os mesmos, ainda de acordo com Reinehr [9], podem ser catalogados em três diferentes tipos, requisitos funcionais, de capacidade (ou não funcionais) e de restrição. No mercado de software os projetos costumam possuir uma série de ações para o objetivo ser alcançado no fim, dentre elas está a elicitação de requisitos, assim como a definição de um paradigma de programação a ser usado no sistema em questão.

### 2.3 PARADIGMAS DE PROGRAMAÇÃO

Segundo Silva, Leite e Oliveira [12], toda linguagem de programação está construída sobre um paradigma. Mas, afinal, o que é um paradigma? Um paradigma representa um padrão de pensamento que guia um conjunto de atividades relacionadas; trata-se de um padrão que define um modelo para a resolução de problemas; basicamente, toda e qualquer linguagem de programação existente. Ainda segundo Silva, Leite e Oliveira [12] Os paradigmas de programação estão classificados em quatro diferentes tipos, que evoluíram ao longo das últimas décadas:

- Programação imperativa;
- Programação funcional;
- Programação lógica;
- Programação orientada a objetos.

Para o presente trabalho, será abordado de forma mais aprofundada o último paradigma mencionado, o da programação orientada a objetos (POO).

### 2.4 PROGRAMAÇÃO ORIENTADA A OBJETOS

Ainda de acordo com Silva, Leite e Oliveira [12], o conceito de programação orientada a objetos (POO) tem suas raízes na linguagem de programação Simula 67, criada por Alan Kay, o precursor dos conceitos desse paradigma de programação. Nesse tipo de paradigma, o projeto de software é separado principalmente por classes e interfaces, em que as classes, por sua vez, possuem atributos e métodos, que possuem ser ou não privados, conforme o que foi definido para a implementação do projeto. Alguns dos princípios desse paradigma podem ser adaptados a depender da linguagem que está sendo utilizada.

### 2.5 *PYTHON*

*Python* foi criada no final dos anos 1980, embora sua concepção tenha acontecido ainda mais no passado: suas raízes estão no time de desenvolvimento de outra linguagem, denominada ABC. Além de ser muito útil para automatizar tarefas, *Python* tem ganhado grande visibilidade nas áreas de desenvolvimento online de aplicações *web* e *machine learning*, pois sua sintaxe concisa permite fazer muito com menos código e suas regras de formatação rígida para o código-fonte favorecem a legibilidade dos programas [5].

### 2.6 DESENVOLVIMENTO WEB

De acordo com Roveda [10], desenvolvimento *web* é a área da tecnologia voltada à construção de sites, aplicativos,

softwares, bancos de dados e quaisquer outras ferramentas que, de certa forma, constroem a internet como a conhecemos hoje. Os profissionais destas áreas são os programadores, ou desenvolvedores *web*: pessoas capacitadas para compreender, manusear e se utilizar de linguagens de programação para construir sistemas complexos voltados ao serviço do usuário.

## 2.7 *DJANGO*

Segundo Maciel [5], um *framework* é um conjunto de classes implementadas em uma determinada linguagem de programação, que serve para facilitar a criação de aplicações. O termo muitas vezes serve para designar um conjunto de arquivos de bibliotecas, códigos-fonte ou compilados e até mesmo recursos, tais como ícones e folhas de estilo, que podem ser reutilizados.

Conhecido como "o *framework* para perfeccionistas com prazos a cumprir", o *Django* se destaca por sua vasta gama de aplicações na área do desenvolvimento *web*, todas desenvolvidas rapidamente, pois ele conta com uma implementação simples e diversos facilitadores.

O *Django* adota o padrão de projeto *MTV (Model, Template, View)*, uma variação do padrão *MVC (Model, View, Controller)*. Em termos simples, *Django* separa a camada de interface com o usuário de sua aplicação em três partes
[5]:

- A *model*, que se comunica com o banco de dados, por baixo dos panos, são classes *Python*, em palavras resumidas, são o núcleo de um projeto *Django*.
- O *template*, sendo aquilo que o usuário visualiza, aqui entram as linguagens *HTML, CSS* e *Javascript*.
- A *view*, sendo a parte encarregada de controlar a interação entre os dados (Modelo) e a apresentação (*Template*).

## 2.8 *CREATE, READ, UPDATE AND DELETE (CRUD)*

De acordo com Noleto [7], a sigla *CRUD* significa as iniciais das operações *create* (criação), *read* (leitura), *update* (atualização) e *delete* (exclusão). Essas quatro iniciais tratam a respeito das operações executadas em bancos de dados relacionais e não-relacionais. Essas operações pertencem ao agrupamento chamado de *Data Manipulation Language (DML)*, utilizado na linguagem *Structured Query Language (SQL)*.

## 2.9 BANCO DE DADOS

Segundo Date [2], um sistema de banco de dados é um sistema computadorizado de manutenção de registros. O banco de dados, por si só, pode ser considerado o equivalente eletrônico de um armário de arquivamento; ou seja, ele é um repositório ou recipiente para uma coleção de arquivos de dados computadorizados.

# 3. TRABALHOS RELACIONADOS

A presente seção, tem por objetivo apresentar uma visão geral dos estudos relacionados com o atual trabalho, visando demonstrar a relação entre os mesmos e mostrar que a contribuição do *Django Full Crud* é válida para o âmbito do mercado de desenvolvimento de software.

O trabalho de Iserhard [4], tem por objetivo contribuir para a área de automação de processos de negócio, fornecendo um *framework* que possa ser aplicado em diferentes tipos de organizações, visando a melhoria da eficiência operacional e a redução de custos por meio da automação de tarefas repetitivas e tediosas.

Nota-se a semelhança entre ele e o presente trabalho ao se comparar o fato de ambos tentarem trazer eficiência ao dia a dia, reduzindo assim os custos de algumas tarefas, focando especialmente na diminuição do tempo necessário para se implementá-las.

Salomi e Maciel [11] abordam a importância da gestão na área da saúde, destacando a utilização da gestão de documentos e a automação de processos como meios para alcançar melhorias tanto nos processos documentais dos pacientes quanto na organização na totalidade. O objetivo central do estudo é reunir dados e índices sobre o tema por meio de uma revisão da literatura.

Dessa forma, o trabalho apresenta evidências e dados relacionados à gestão de documentos e automação de processos na área da saúde, com a intenção de destacar a importância dessas práticas para melhorar a eficiência, qualidade e segurança dos serviços prestados, além de atingir um alto nível de maturidade na gestão da informação e tecnologia na saúde.

Bem como o trabalho de Salomi e Maciel [11], o presente artigo também tem por interesse melhorar a qualidade dos serviços prestados, removendo um pouco a responsabilidade do desenvolvedor de escrever códigos repetitivos e deixando isso como uma tarefa para o *Django Full Crud*.

Já o artigo de Manzueto [6] tem por objetivo apresentar as vantagens e aplicabilidade das tecnologias de automação, como a automação robótica de processos (RPA) e os *chatbots*, destacando sua capacidade de substituir tarefas repetitivas, melhorar a eficiência e reduzir custos.

O presente trabalho, por meio da demonstração do processo de desenvolvimento do pacote, também terá a forte possibilidade de substituir tarefas, melhorar eficiência e reduzir custos, seguindo os objetivos gerais das automações de processos.

Diversos outros trabalhos abordam automações de processos, conforme os estudos realizados, não são muitos os que focam precisamente na geração de códigos automatizados para o desenvolvimento *web*. Porém, por se tratarem de automações, é possível notar semelhanças com os artigos mencionados nos parágrafos anteriores desta mesma seção. Para uma melhor compreensão e visualização dos dados, constate a Tabela 1, em que são feitas comparações do *Django Full Crud* com os demais artigos já abordados.

# 4. METODOLOGIA

Nesta seção, aborda-se a metodologia usada para o desenvolvimento do *Django Full Crud*. Como o presente trabalho não elucidará a viabilidade do projeto a ser desenvolvido, o uso dos métodos de coleta de dados, amostras e análises foram dispensados.

Diante do vislumbre de uma possível solução para os problemas mencionados na seção 1, foi desenvolvido um pacote que tem por objetivo a geração de códigos padronizados para as operações dos *CRUD's*. O mesmo também visa configurar as rotas necessárias para dar acesso às páginas criadas e gerar os arquivos para a criação de *APIs*, em conjunto com o pacote *Django Rest Framework (DRF)*.

A metodologia ágil *Scrum* foi escolhida devido à sua flexibilidade e eficácia em projetos de desenvolvimento de software. Por meio de *sprints* quinzenais, as quais são iterações curtas e focadas, ou seja, cada uma durou 15 dias corridos, foi possível garantir um progresso constante e uma entrega de valor ao longo do tempo. A utilização dessa metodologia contribuiu para a agilidade no desenvolvimento do pacote *Django Full Crud*, proporcionando maior eficiência e adaptabilidade durante todo o processo.

Table 1: Comparação do presente trabalho com trabalhos relacionados

| | Visa diminuir custos operacionais | Visa automatizar uma tarefa repetitiva | Visa otimizar o tempo ao se implementar tarefas | Visa aumentar a qualidade de algum processo |
|---|---|---|---|---|
| Isehard [4] | X | X | X | X |
| Salomi e Maciel [11] | | | X | X |
| Manzueto [6] | X | | X | |
| Django-Full-Crud | X | X | X | X |

No início de cada *sprint*, foi realizado um planejamento detalhado das atividades a serem desenvolvidas. Serão definidos os objetivos da mesma, selecionadas as funcionalidades a serem implementadas em cada tarefa.

- Primeira *sprint*: Desenvolvimento da base do projeto e implementação das funcionalidades para os arquivos Configuração necessária para a criação de pastas e arquivos gerais. Ao final desta *sprint*, o pacote já realizava a geração de *admins*, incluindo os arquivos *init.py*.
- Segunda *sprint*: Desenvolvimento dos arquivos necessários para a criação dos *forms, views, templates* e rotas.
- Terceira *sprint*: Desenvolvimento dos arquivos necessários para a criação dos *viewsets* e *serializers.*

Foram utilizadas como base as regras e princípios mencionados pela *Python Enhancement Proposals (PEP8)*, guia de estilo oficial da linguagem, garantindo assim uma boa legibilidade, manutenibilidade e simplicidade, tanto nos *scripts* que serão construídos de maneira automática, como no código do projeto a ser desenvolvido. Todos esses pontos corroboraram para atingir o objetivo principal do trabalho: automatizar as tarefas cotidianas dos desenvolvedores de sistemas *web*.

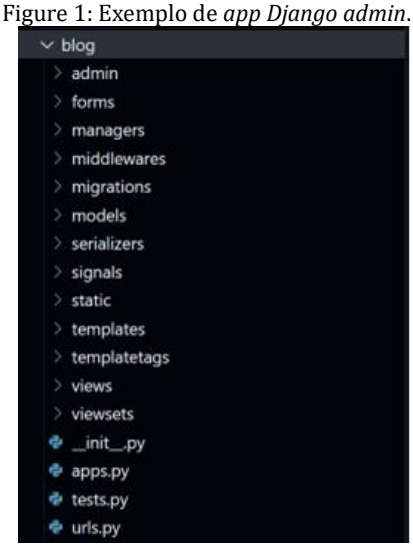
## 5. ABORDAGEM

A presente seção descreve todas as informações relacionadas ao *Django Full Crud*, desde o seu processo de desenvolvimento até a listagem de requisitos funcionais e não funcionais, além de trazer exemplos para a compreensão das capacidades do pacote.

### 5.1 CONTEXTUALIZAÇÃO

Projetos *Django* são divididos em *apps*, que representam partes do sistema, cada uma dessas partes possui uma série de pastas que garantem o seu bom funcionamento, para um exemplo de *app*, veja Figura 1.

Figure 1: Exemplo de *app Django admin*.




Com base nos conceitos da escrita em arquivos, estudou-se a viabilidade de, com alguns parâmetros específicos como o nome da *app* e *model*, aproveitar-se da organização padrão de sistemas feitos com o *framework* e abrir automaticamente as pastas e arquivos necessários, além de escrever conteúdo nos mesmos.Os códigos a serem escritos têm como base as

*models* e os seus respectivos atributos, além de se adaptarem ao tipo de arquivo. Alguns farão mais uso das definições feitas e outros simplesmente irão usar uma variável padrão do próprio *Django* para sinalizar o uso de todos os atributos possíveis *( all )*.

## 5.2 REQUISITOS FUNCIONAIS

Para atingir os objetivos mencionados na seção 2, foi criada uma lista de requisitos funcionais para garantir o bom funcionamento do pacote, seguem:

[RF001] O pacote deverá criar arquivos *admin* utilizando os campos definidos na *model*.

[RF002] O pacote deverá criar um arquivo *Python* em pleno funcionamento para o formulário da *model* em questão, onde todos os atributos devem ser definidos na propriedade *all.*

[RF03] O pacote deverá criar um arquivo *Python* em pleno funcionamento para o *serializer* da *model* em questão, onde todos os atributos devem ser definidos na propriedade *all* .

[RF004] O pacote deverá criar um arquivo *HTML* para a página de exclusão da *model* em questão, seguindo o padrão: *"<model name> delete.html"*. O arquivo deverá conter um formulário solicitando a confirmação do usuário para excluir a instância da *model* em questão.

[RF005] O pacote deverá criar um arquivo *HTML* para a página de detalhes da *model* em questão, seguindo o padrão: *"<model name> detail.html"*. Onde todos os atributos devem ser adicionados no *template*, visando facilitar o desenvolvimento futuro.

[RF006] O pacote deverá criar um único arquivo *HTML* para a página criação e edição da *model* em questão, seguindo o padrão: *"<model name> form.html"*. Onde todos os atributos devem ser adicionados no *template*, visando facilitar o desenvolvimento futuro.

[RF007] O pacote deverá criar um arquivo *HTML* para a página de listagem da *model* em questão, seguindo o padrão: *"<model name> list.html"*. Onde todos os atributos devem ser listados em uma tabela, sendo o cabeçalho da mesma o *verbose name* do atributo presente na *model*.

[RF008] O pacote deverá criar um arquivo *Python* em pleno funcionamento para a *view* de criação da *model* em questão, seguindo o padrão: "*<model name> create view.py*".

[RF009] O pacote deverá criar um arquivo *Python* em pleno funcionamento para a *view* de exclusão da *model* em questão, seguindo o padrão: "*<model name> delete view.py*".

[RF010] O pacote deverá criar um arquivo *Python* em pleno funcionamento para a *view* de detalhes da *model* em questão, seguindo o padrão: "*<model name> detail view.py*".

[RF011] O pacote deverá criar um arquivo *Python* em pleno funcionamento para a *view* de listagem da *model* em questão, seguindo o padrão: "*<model name> list view.py*".

[RF012] O pacote deverá criar um arquivo *Python* em pleno funcionamento para a *view* de edição da *model* em questão, seguindo o padrão: *"<model name> update view.py*".

[RF013] O pacote deverá criar um arquivo *Python* em pleno funcionamento para o *viewset* da *model* em questão, onde todos os atributos devem ser definidos na propriedade *all* .

[RF014] O pacote deverá criar todos os arquivos *init .py* presentes em todas as pastas e subpastas necessárias. Adicionando todas as classes criadas pelo mesmo.

[RF015] O pacote deverá criar o arquivo *urls.py* da app relacionada à *model* em questão. Adicionando todos os *path's* das *views* e *viewsets* criadas pelo menos.

[RF016] O pacote deverá possibilitar a execução de um comando através do *manage.py* do próprio *Django*, onde o desenvolvedor poderá informar dois parâmetros, sendo eles: *app* e *model*.

[RF017] O pacote deverá realizar a criação de todos os arquivos mencionados anteriormente para uma única *model* caso o usuário execute o comando principal da seguinte forma:
*python manage.py full crud app name ModelName*.

[RF018] O pacote deverá realizar a criação de todos os arquivos mencionados anteriormente para todas as *models* de uma *app* caso o usuário execute o comando principal da seguinte forma: *python manage.py full crud app name*.

[RF019] O pacote deverá realizar a criação de todos os arquivos mencionados anteriormente para todas as *models* do projeto caso o usuário execute o comando principal da seguinte forma: *python manage.py full crud*.

## 5.3 REQUISITOS NÃO FUNCIONAIS

Além dos requisitos funcionais, foram criados os seguintes requisitos não funcionais:

[RNF001] Com relação à criação dos *path's*, o pacote deverá garantir que nenhum dos *path's* anteriormente presentes no arquivo *urls.py* serão sobrescritos.

[RNF002] Com relação à criação dos arquivos *init .py*, o pacote deverá garantir que nenhuma das importações anteriormente presentes nos arquivos sejam sobrescritas.

Tendo em vista os requisitos listados, pode-se perceber que o pacote aqui proposto é uma ferramenta útil para o dia a dia de um desenvolvedor. Podendo dar suporte para tarefas cotidianas e triviais com a geração de todos os arquivos outrora mencionados, para uma melhor compreensão, consultar a Figura 2.

## 5.4 EXEMPLO PRÁTICO

Visando garantir a boa compreensão do funcionamento do pacote, a presente subseção aspira descrever um exemplo prático, mostrando como utilizar o pacote em um projeto *Django*.

Partindo do ponto em que o desenvolvedor possua um projeto e uma *app* no modelo da Figura 3 e Figura 4, respectivamente, o mesmo precisa desenvolver as *models*, definindo todos os atributos das mesmas.

No presente exemplo, as models *Person*, *Country* e *Interest*, foram desenvolvidas, a fim de representar uma pessoa, um país e um interesse, conforme exemplificam as Figuras 5, 6 e 7, respectivamente.

Após ter criado todas as *models*, o desenvolvedor precisa instalar o pacote com o comando: *"pip install*

*django-fullcrud"*, adicionar o mesmo na lista de *INSTALLED APPS*, presente no arquivo *settings.py* e também criar uma pasta chamada *".vscode"*, dentro dela, ele precisará adicionar um arquivo chamado *"django full crud.json"* e nele, adicionar o nome do seu projeto (Figuras 8 e 9).

Com as configurações feitas, o desenvolvedor pode executar três comandos diferentes:

- python manage.py full crud <app name><ModelName>
- python manage.py full crud <app name>
- python manage.py full crud

Figure 2: Ilustração do fluxo para geração dos arquivos necessários para uma única *model*

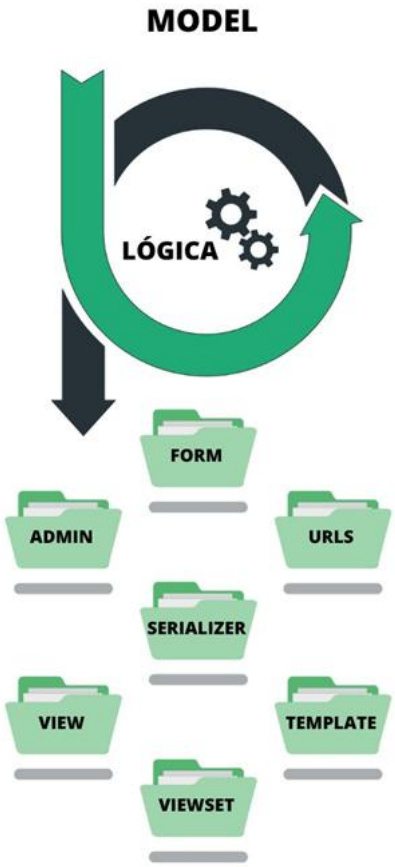



Figure 3: Exemplo de projeto

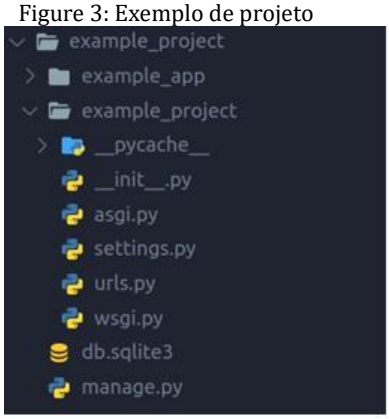



Figure 4: Exemplo de *app*

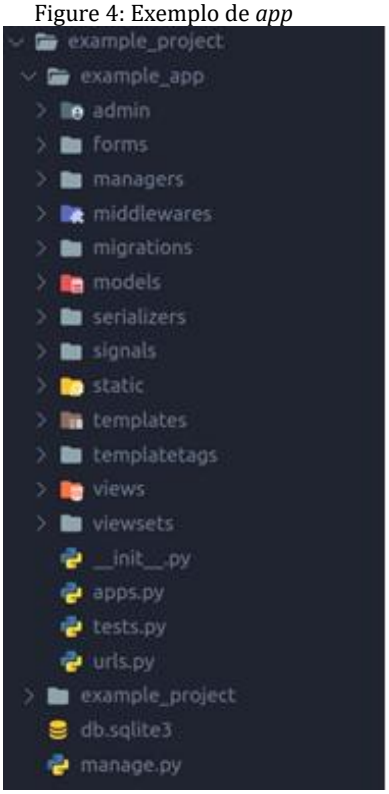



Figure 5: Exemplo da *model Person*

```
class Person(models.Model):
    first_name = models.CharField(
        verbose_name="First Name",
        max_length=100,
    )

    last_name = models.CharField(
        verbose_name="Last Name",
        max_length=100,
    )

    age = models.IntegerField(
        verbose_name="Age",
    )

    email = models.EmailField(
        verbose_name="Email",
    )

    birth_date = models.DateField(
        verbose_name="Birth Date",
    )

    nationality = models.ForeignKey(
        Country,
        on_delete=models.CASCADE,
        verbose_name="Nationality",
    )

    interests = models.ManyToManyField(
        Interest,
        verbose_name="Interests",
    )

    def __str__(self):
        """Return the representation of the object as string.."""
        return f"{self.first_name} {self.last_name}"

    class Meta:
        """Defines meta attributes of the main class."""

        app_label = "example_app"
        verbose_name = "Person"
        verbose_name_plural = "People"
```

Figure 6: Exemplo da *model Country*

```python
from django.db import models


class Interest(models.Model):
    name = models.CharField(
        verbose_name="Interest Name",
        max_length=100,
    )

    def __str__(self):
        """Return the representation of the object as string.."""
        return self.name

    class Meta:
        """Defines meta attributes of the main class."""

        app_label = "example_app"
        verbose_name = "Interest"
        verbose_name_plural = "Interests"
```

Figure 7: Exemplo da *model Interest*

```python
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    # Apps
    "example_app",
    # Packages
    "django_full_crud",
]
```

Figure 8: Exemplo de como adicionar o pacote na *INSTALLED APPS*


Figure 9: Exemplo da pasta *.vscode* e do arquivo *django full crud.json*

```python
from django.db import models


class Country(models.Model):
    name = models.CharField(
        verbose_name="Country Name",
        max_length=100,
    )

    def __str__(self):
        """Return the representation of the object as string."""
        return self.name

    class Meta:
        """Defines meta attributes of the main class."""

        app_label = "example_app"
        verbose_name = "Country"
        verbose_name_plural = "Countries"
```

O primeiro comando executa o *Django Full Crud* em uma *model* específica, o segundo em uma *app*, e o terceiro no projeto inteiro.

Veja a Figura 10 com um exemplo do primeiro comando, para obter uma melhor compreensão, os demais geram os mesmos arquivos com um escopo maior.

## 6. CONCLUSÃO

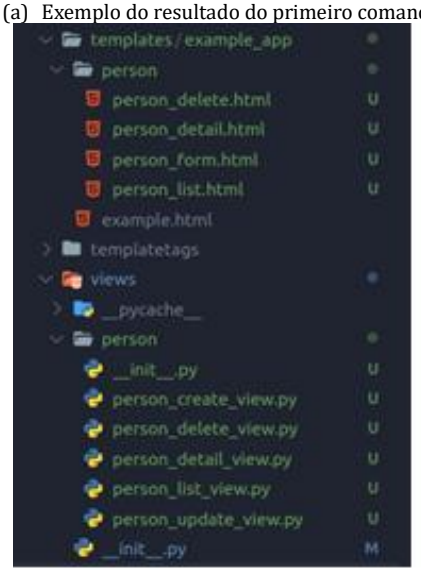
O intuito do trabalho em demonstrar o desenvolvimento do pacote, bem como suas capacidades, foi atingido e, pode-se observar que o mesmo possui o potencial necessário para ser, de certa forma, um inicializador de projetos, onde toda a base e desenvolvimento inicial de um sistema pode ser feito com o *Django Full Crud*.

O mesmo vem para o mercado de software a fim de contribuir com a área de automação, atingindo especialmente o dia a dia de desenvolvedores *web* que utilizam as tecnologias *Python* e *Django*, melhorando a velocidade de desenvolvimento e trazendo um auxílio para os códigos repetitivos.

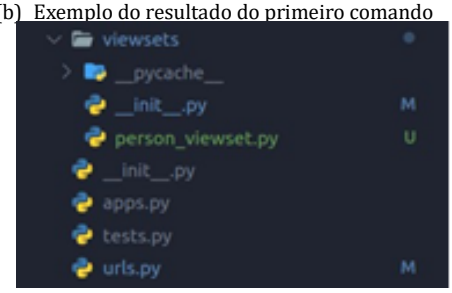
Como trabalhos futuros, sugere-se elaborar um estudo do pacote, em que sejam coletadas e medidas estatísticas relevantes para checar a viabilidade do uso dele, haja vista que o presente trabalho elucidou apenas os pontos voltados para o desenvolvimento do *Django Full Crud*, e não abordou nenhuma medição para checar se o mesmo realmente faz a diferença durante o desenvolvimento de algum projeto.

Figure 10: Exemplo do resultado do primeiro comando

(a) Exemplo do resultado do primeiro comando




(b) Exemplo do resultado do primeiro comando




## 7. REFERENCES


[1] *Bíblia*. 2011. Bíblia.

[2] *C. J. Date. Introdução a Sistemas de Bancos de Dados. Grupo GEN, 1 edition, 2004.*

[3] *Django Software Foundation. Django makes it easier to build better web apps more quickly and with less code. Disponível em:*

[4] *https://www.djangoproject.com/, 2023. Acesso em 21 fev. 2023.*

[5] *D. Iserhard. Proposta de framework para automação de processos em instituições federais de ensino superior. Master's thesis, Universidade Federal do Rio Grande do Sul, 2021.*

[6] *F. M. d. B. Maciel. Python e Django. Editora Alta Books, 1 edition, 2020.*

[7] *M. S. Manzueto. Automação de processos: a influência dos softwares de automação de processos nas rotinas organizacionais. Disponível em:*

[8] *https://www2.dbd.puc-rio.br/pergamum/tesesabertas/1412538_2016_completo.pdf, 2016. Acesso em 11 jun. 2023.*

[9] *C. Noleto. Crud: as 4 operações básicas do banco de dados! Disponível em: https://blog.betrybe.com/tecnologia/crud-operacoes-basicas/, 2021. Acesso em 21 abr. 2023.*

[10] *R. S. Pressman and B. R. Maxim. Engenharia de Software Uma Abordagem Profissional. Bookman, 8 edition, 2016.*

[11] *S. Reinehr. Engenharia de requisitos. Grupo A, 1 edition, 2020.*

[12] *U. Roveda. Desenvolvimento web: o que é e como ser um desenvolvedor web. Disponível em: https:*

[13] *//kenzie.com.br/blog/desenvolvimento-web/, 2020. Acesso em 15 abr. 2023.*

[14] *M. J. A. Salomi and R. F. Maciel. Gestão de documentos e automação de processos em uma instituição de saúde sem papel. Journal of Health Informatics, 8(1), jan. 2016.*

[15] *F. M. Silva, M. C. D. Leite, and D. B. Oliveira.*

[16] *Paradigmas de programação. Grupo A, 1 edition, 2019.*