

PROGRAMAÇÃO BACK-END

django

The web framework for
perfectionists with deadlines.

PROGRAMAÇÃO BACK-END

Django Forms

PROGRAMAÇÃO BACK-END

Introdução

PROGRAMAÇÃO BACK-END

O **Django Forms** é uma das ferramentas do framework **Django**, permitindo que desenvolvedores criem e manipulem formulários HTML com facilidade e eficiência.

Automatização: Criação automática de campos a partir de modelos, evitando retrabalho.

Validação: Oferece mecanismos robustos para validar dados de entrada, garantindo que eles atendam aos critérios definidos.

PROGRAMAÇÃO BACK-END

Segurança: Protege contra diversos tipos de ataques, como ataques de injeção de SQL e Cross-Site Scripting (XSS).

Personalização: Permite estilização e personalização detalhada dos formulários, mantendo o código organizado.

PROGRAMAÇÃO BACK-END

Visão Geral

PROGRAMAÇÃO BACK-END

Visão geral e Facilidades do Django Forms

Model Forms são uma das ferramentas mais poderosas do Django, permitindo a **criação automática** de formulários a partir dos modelos definidos no **models.py**.

Sincronização com o modelo: Qualquer alteração no modelo reflete automaticamente no formulário.

PROGRAMAÇÃO BACK-END

Menos código: Elimina a necessidade de definir campos de formulário manualmente.

Validação automática: Baseado nas restrições e validações definidas no modelo.

Métodos úteis: Como `form.save()` que permite salvar os dados do formulário diretamente no banco de dados.

PROGRAMAÇÃO BACK-END

Formulário de Registro User

PROGRAMAÇÃO BACK-END

Usar o Model Form juntamente com o modelo padrão User do Django simplifica o processo de criação de um formulário de registro.

Passo 1: Modelo User Padrão. O Django já fornece um modelo User pronto para uso com campos como `first_name`, `last_name`, `username`, `email`, `password`, etc.

Passo 2: Criar o Formulário. Com base no modelo User, podemos criar um formulário de registro.

PROGRAMAÇÃO BACK-END

```
# models.py (Este arquivo é parte do Django,
# em django.contrib.auth.models
from django.contrib.auth.models import User

# forms.py
from django import forms

class RegistrationForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput())

    class Meta:
        model = User
        fields = [
            'first_name',
            'last_name',
            'username',
            'email',
            'password',
        ]
```

PROGRAMAÇÃO BACK-END

Processando Dados de
Formulário com POST

PROGRAMAÇÃO BACK-END

No Django, quando um formulário é submetido, podemos manipular e validar os dados enviados através do POST. Além disso, podemos reter certas informações entre requisições usando SESSION.

Passo 1: Enviando Dados via POST. Os dados submetidos por um formulário são acessados por request.POST; O formulário é instanciado novamente com esses dados, permitindo a validação e processamento.

PROGRAMAÇÃO BACK-END

Passo 2: Retendo Informações com SESSION.
Mensagens ou outros dados podem ser armazenados entre requisições usando **request.session**.

PROGRAMAÇÃO BACK-END

```
from django.shortcuts import redirect

def register_view(request):
    if request.POST:
        form = RegisterForm(request.POST)
        if form.is_valid():
            # processar dados se forem válidos
            user = form.save(commit=False)
            user.set_password(user.password)
            user.save()
            request.session['message'] = "Registro bem-sucedido!"
            return redirect('home.html')
    else:
        form = RegisterForm()

    return render(request, 'register_view.html', {'form': form})
```

PROGRAMAÇÃO BACK-END

Exemplos

PROGRAMAÇÃO BACK-END

- A função `redirect` é importada para poder redirecionar o usuário para uma página diferente após o registro bem-sucedido.
- A função da `view` que será chamada quando um usuário acessa a URL associada ao registro.
- O código verifica se o método de requisição é `POST`. Se for, significa que o usuário submeteu o formulário de registro.

PROGRAMAÇÃO BACK-END

- Se o método de requisição for POST, um formulário RegisterForm é instanciado com os dados do POST. O método `is_valid()` verifica se os dados submetidos são válidos de acordo com as regras definidas no formulário.
- Se o formulário for válido, ele é salvo e um objeto user é criado. O parâmetro `commit=False` significa que o formulário é salvo na variável user, mas ainda não é salvo no banco de dados.

PROGRAMAÇÃO BACK-END

- A senha do usuário é então definida usando o método `set_password`, e o usuário é salvo no banco de dados com `user.save()`.
- Uma mensagem é armazenada na sessão para informar ao usuário que o registro foi bem-sucedido.
- O usuário é então redirecionado para a página inicial ('home.html').

PROGRAMAÇÃO BACK-END

- Se o método de requisição não for POST (por exemplo, GET), um formulário RegisterForm vazio é criado. Isso é feito para exibir o formulário de registro na página.
- Finalmente, a página de registro register_view.html é renderizada, passando o formulário como contexto. Isso permite que o formulário seja exibido na página HTML.

PROGRAMAÇÃO BACK-END

Configurando Detalhes
do Formulário

PROGRAMAÇÃO BACK-END

- **Labels:** Utilizadas para identificar os campos do formulário.
- **Erros:** Exiba erros específicos do campo, ajudando o usuário a corrigir entradas inválidas.
- **Atributos dos Campos:** Podem ser personalizados para adicionar detalhes como classes, IDs, atributos data- e mais.
- **Texto de Ajuda:** Auxilia o usuário fornecendo informações adicionais sobre como preencher um campo.
- **Validação:** Django Forms fornece validação out-of-the-box. Erros são exibidos automaticamente ao lado dos campos respectivos.

PROGRAMAÇÃO BACK-END

```
{% extends 'base_templates/header.html' %}  
{% block title %} Registro {% endblock title %}  
{% block content %}  
    <div class="main-content center container">  
        <h2>Register</h2>  
    </div>  
    <div class="main-content container">  
        <form action="" method="POST">  
            {% csrf_token %}  
  
            <div class="form-content form-content-grid">  
                {% for field in form %}  
                    <div class="form-group">  
                        <label for="{{ field.id_for_label }}>{{ field.label }}</label>  
                        {{field}}  
  
                        {% if field.help_text %}  
                            <p class="help-text">{{ field.help_text }}</p>  
                        {% endif %}  
  
                        {{ field.errors }}  
                    </div>  
                {% endfor %}  
            </div>  
  
            <div class="form-group">  
                <button type="submit">Send</button>  
            </div>  
        </form>  
    </div>  
    {% endblock content %}
```

PROGRAMAÇÃO BACK-END

Sobrescrevendo Campos
no Django Forms

PROGRAMAÇÃO BACK-END

Por que sobrescrever?:

- Personalizar a exibição.
- Alterar validações padrão.
- Ajustar para necessidades específicas do projeto.

Como Sobrescrever: Substitua um campo em sua classe de formulário

PROGRAMAÇÃO BACK-END

```
class RegisterForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput())

    first_name = forms.CharField(
        max_length=100,
        widget=forms.TextInput(attrs={'class': 'special-input'}),
        label='Seu primeiro nome',
        help_text='Insira o primeiro nome',
    )

    email = forms.EmailField(
        label='Endereço de Email',
        widget=forms.EmailInput(attrs={
            'class': 'email-input',
            'placeholder': 'exemplo@email.com'
        }),
    )

    class Meta:
        model = User
        fields = [
            'first_name',
            'last_name',
            'username',
            'email',
            'password',
        ]
```

PROGRAMAÇÃO BACK-END

Validação de Campos

PROGRAMAÇÃO BACK-END

Objetivo da Validação:

- ✓ Evitar registros duplicados.
- ✓ Garantir que cada usuário tenha um e-mail único no sistema.
- ✓ Fornecer feedback imediato ao usuário sobre e-mails já em uso.
- ✓ Uso do clean_email

No Django, o método **clean_fieldname** permite a validação específica de campos.

O método **clean_email** é usado para validar o campo de e-mail.

PROGRAMAÇÃO BACK-END

```
def clean_email(self):
    email = self.cleaned_data.get('email', '')
    exists = User.objects.filter(email=email).exists()

    if exists:
        raise ValidationError(
            'User e-mail is already in use', code='invalid',
        )

    return email
```

PROGRAMAÇÃO BACK-END

Validators

PROGRAMAÇÃO BACK-END

O que são **Validators**?:

Funções ou classes específicas para verificar a validade dos dados.

- ✓ Podem ser reutilizadas em diferentes campos ou modelos.
- ✓ Oferecem uma maneira limpa e modular de garantir a integridade dos dados.
- ✓ Aplicando validators a um campo.

PROGRAMAÇÃO BACK-END

```
from django.core.exceptions import ValidationError
import re

class RegisterForm(forms.ModelForm):

    class Meta: ...

    def clean_email(self): ...

    def validate_complex_password(value):
        if not re.match(r'^([a-z]+[A-Z]+[0-9]+)', value):
            raise ValidationError(
                "A senha deve conter uma letra maiúscula, uma letra minúscula e um número.")

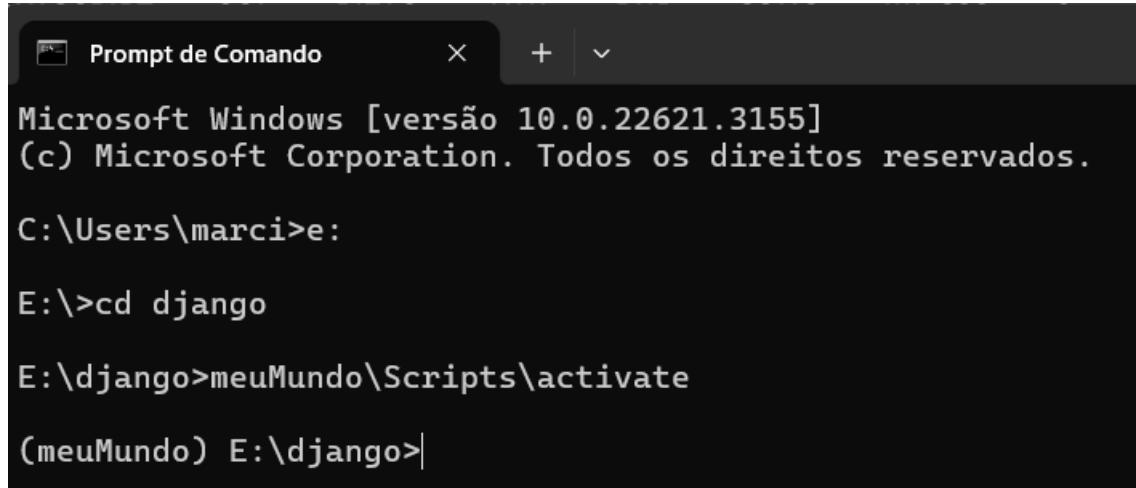
    password = forms.CharField(widget=forms.PasswordInput(), validators=[
        validate_complex_password])
```

PROGRAMAÇÃO BACK-END

Preparando para os testes

PROGRAMAÇÃO BACK-END

Rodar o script activate.bat



```
Prompt de Comando
Microsoft Windows [versão 10.0.22621.3155]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\marci>e:

E:\>cd django

E:\django>meuMundo\Scripts\activate

(meuMundo) E:\django>
```

Este passo é importante para configurar as variáveis de ambiente.

PROGRAMAÇÃO BACK-END

Iniciar Servidor Postgresql

PROGRAMAÇÃO BACK-END

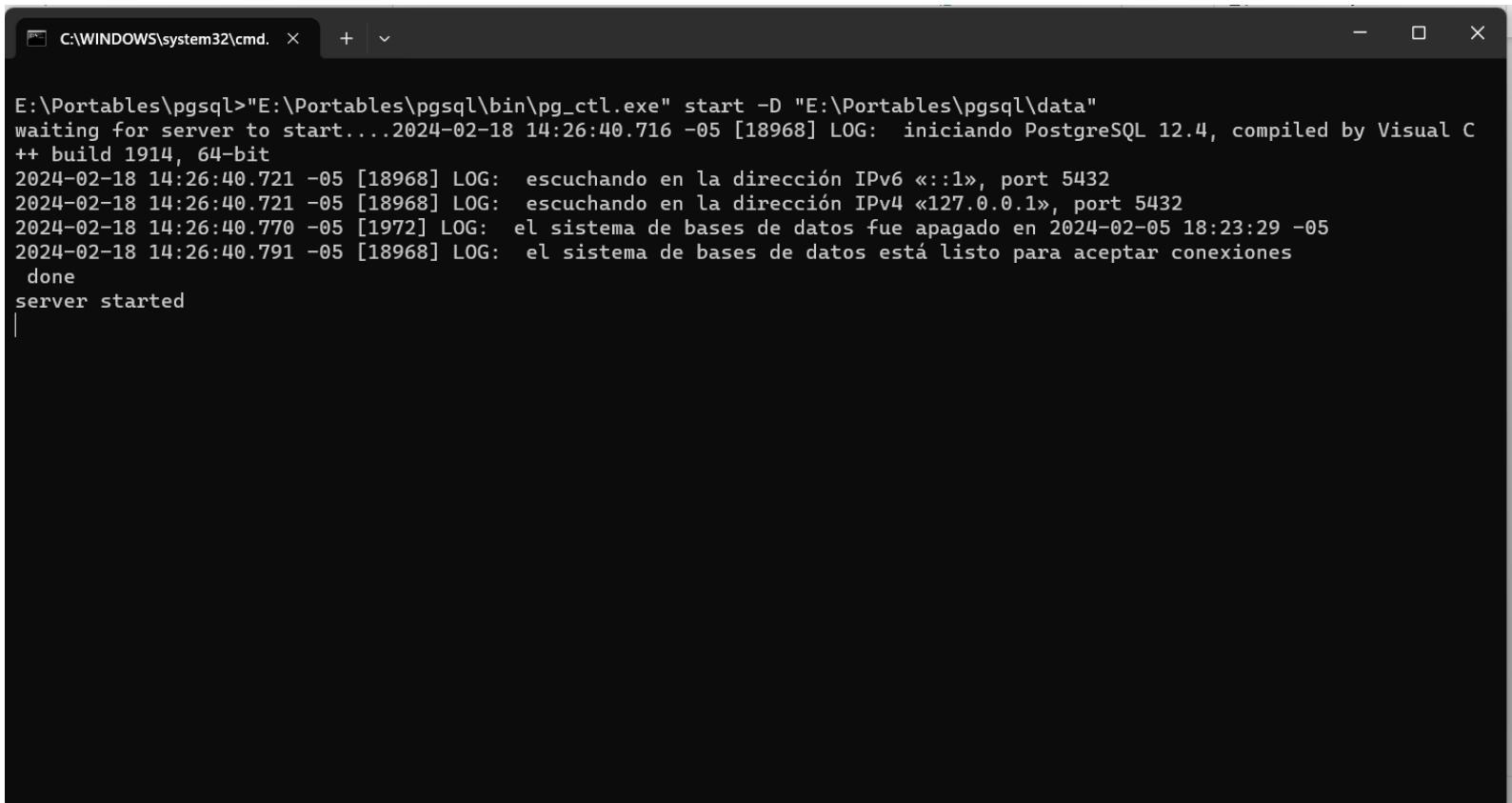
Na pasta `../pgsql` execute o arquivo `PostgreSQL-Start.bat` para iniciar o servidor



Nome	Data de modificação	Tipo	Tamanho
bin	11/08/2020 08:41	Pasta de arquivos	
data	18/02/2024 16:26	Pasta de arquivos	
doc	11/08/2020 08:41	Pasta de arquivos	
include	11/08/2020 08:41	Pasta de arquivos	
lib	11/08/2020 08:41	Pasta de arquivos	
pgAdmin 4	11/08/2020 08:41	Pasta de arquivos	
share	11/08/2020 08:41	Pasta de arquivos	
StackBuilder	11/08/2020 08:41	Pasta de arquivos	
Make Cluster.bat	09/01/2017 14:12	Arquivo em Lotes ...	1 KB
PgAdmin4.bat	28/04/2017 22:29	Arquivo em Lotes ...	1 KB
PostgreSQL-Restart.bat	06/04/2012 09:42	Arquivo em Lotes ...	1 KB
PostgreSQL-Start.bat	28/04/2017 21:46	Arquivo em Lotes ...	1 KB
PostgreSQL-Stop.bat	06/04/2012 09:43	Arquivo em Lotes ...	1 KB
Service - Delete.bat	29/08/2020 23:25	Arquivo em Lotes ...	1 KB
Service - Maker.bat	29/08/2020 23:26	Arquivo em Lotes ...	1 KB

PROGRAMAÇÃO BACK-END

Será aberta uma janela em Texto informando que o servidor iniciou (server started**).
Não feche esta janela**



A screenshot of a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.". The window contains the following text output from the command "E:\Portables\pgsql>E:\Portables\pgsql\bin\pg_ctl.exe start -D "E:\Portables\pgsql\data"":

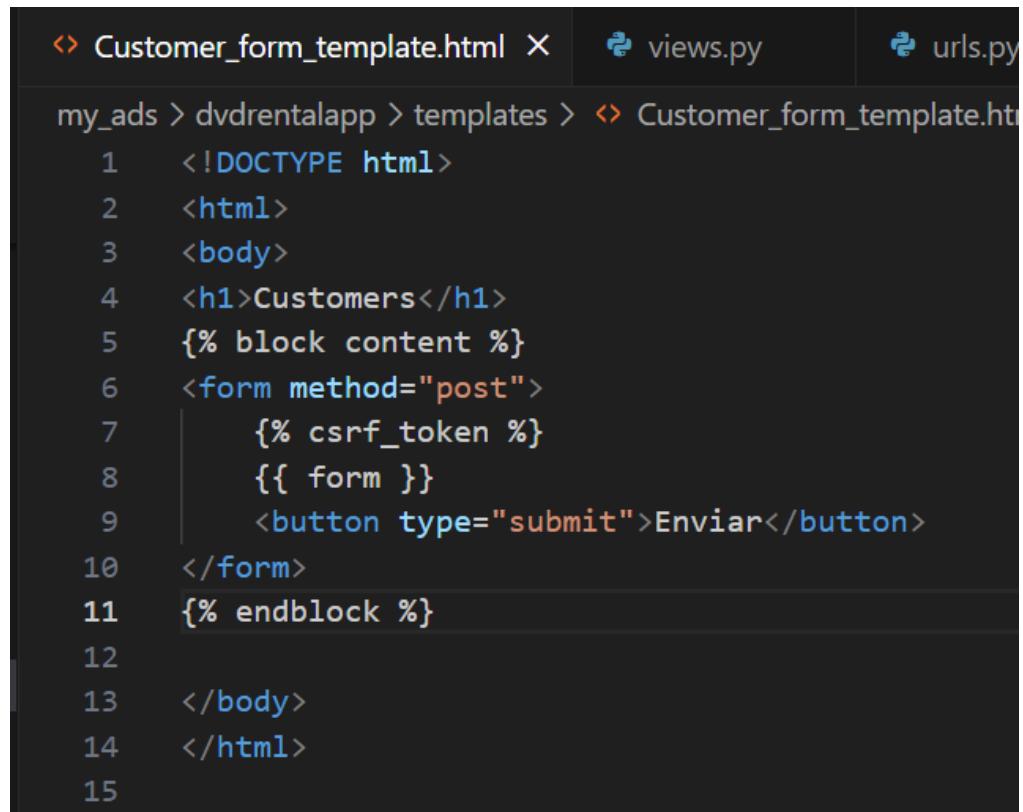
```
E:\Portables\pgsql>"E:\Portables\pgsql\bin\pg_ctl.exe" start -D "E:\Portables\pgsql\data"
waiting for server to start....2024-02-18 14:26:40.716 -05 [18968] LOG:  iniciando PostgreSQL 12.4, compiled by Visual C
++ build 1914, 64-bit
2024-02-18 14:26:40.721 -05 [18968] LOG:  escuchando en la dirección IPv6 «::1», port 5432
2024-02-18 14:26:40.721 -05 [18968] LOG:  escuchando en la dirección IPv4 «127.0.0.1», port 5432
2024-02-18 14:26:40.770 -05 [1972] LOG:  el sistema de bases de datos fue apagado en 2024-02-05 18:23:29 -05
2024-02-18 14:26:40.791 -05 [18968] LOG:  el sistema de bases de datos está listo para aceptar conexiones
  done
server started
|
```

PROGRAMAÇÃO BACK-END

Templates

PROGRAMAÇÃO BACK-END

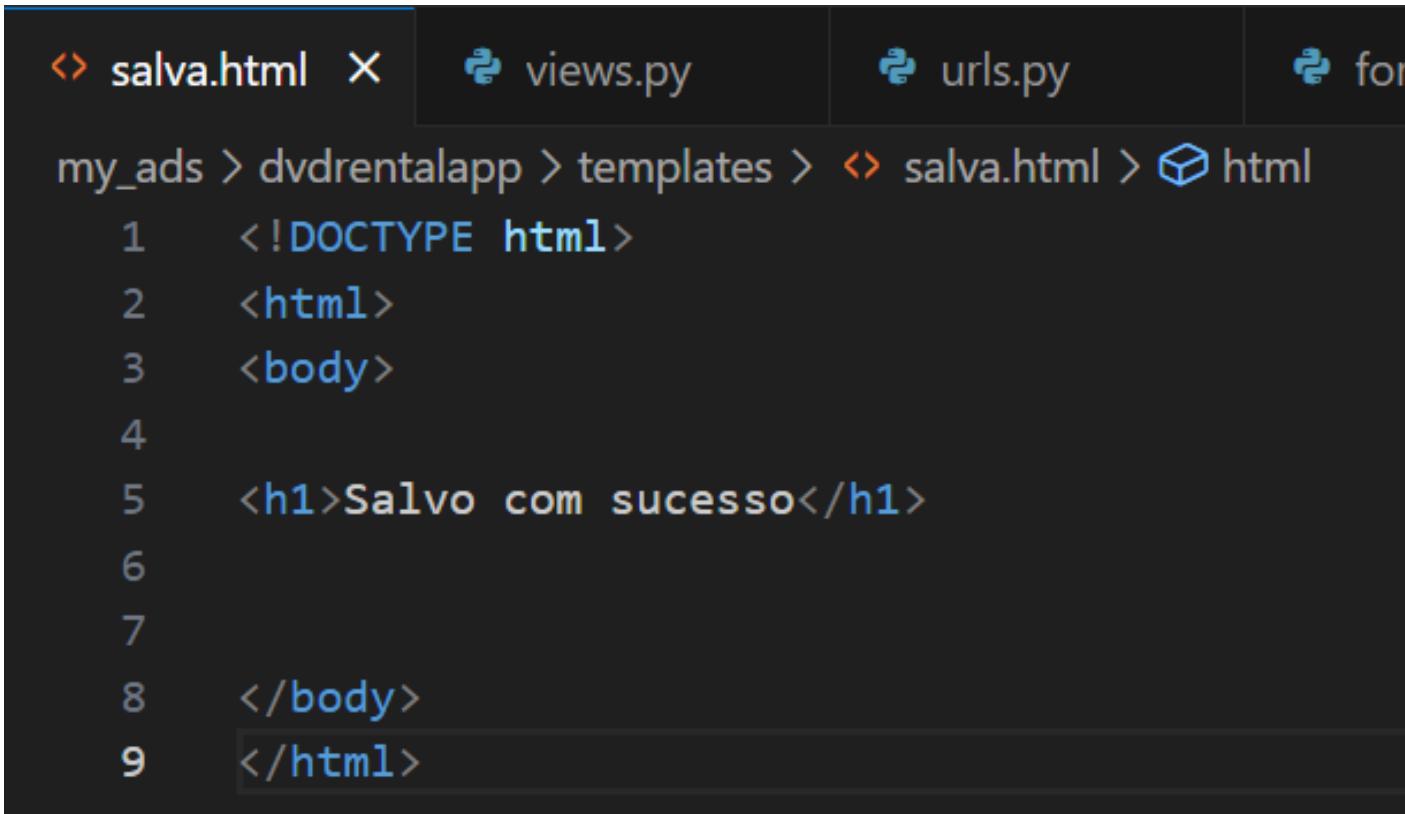
Utilizando a App **dvdrentalapp**. Na pasta **templates** crie o arquivo **Customer_form_template.html** e digite o seguinte conteúdo:



```
Customer_form_template.html × views.py urls.py
my_ads > dvdrentalapp > templates > Customer_form_template.html
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <h1>Customers</h1>
5  {% block content %}
6  <form method="post">
7      {% csrf_token %}
8      {{ form }}
9      <button type="submit">Enviar</button>
10 </form>
11 {% endblock %}
12
13 </body>
14 </html>
15
```

PROGRAMAÇÃO BACK-END

Crie também o arquivo **salva.html** e digite o seguinte conteúdo:



The screenshot shows a code editor interface with a dark theme. The top navigation bar includes tabs for 'salva.html' (which is currently active), 'views.py', 'urls.py', and 'forms.py'. Below the tabs, the file structure is displayed as 'my_ads > dvdrentalapp > templates > salva.html'. The main code area contains the following HTML code:

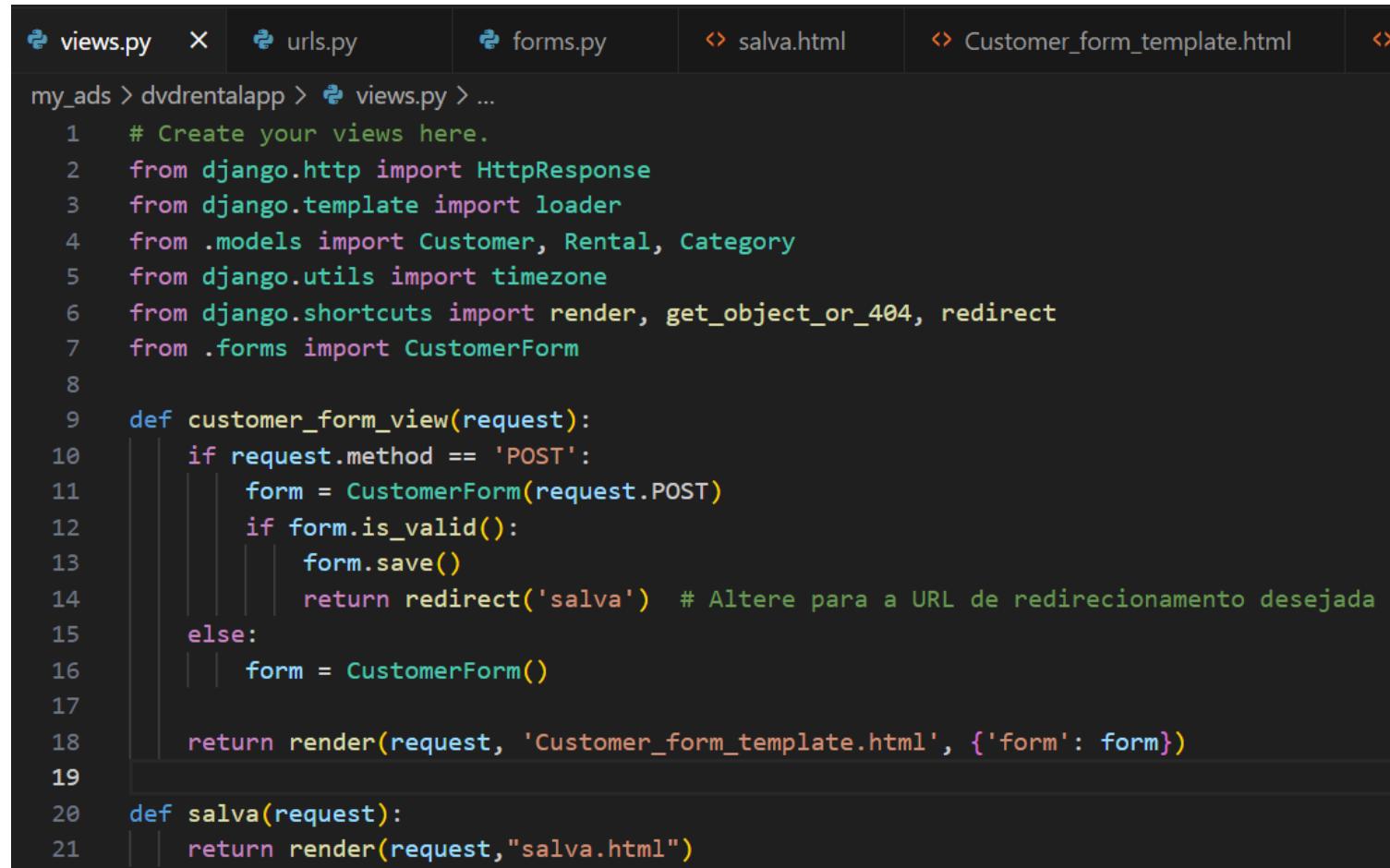
```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <h1>Salvo com sucesso</h1>
6
7
8  </body>
9  </html>
```

PROGRAMAÇÃO BACK-END

Arquivos views.py e urls.py

PROGRAMAÇÃO BACK-END

Na pasta `dvdrentalapp` adicione o seguinte conteúdo no arquivo `views.py` (linhas 7 a 21)



The screenshot shows a code editor window with the file `views.py` open. The tab bar at the top includes `views.py`, `urls.py`, `forms.py`, `salva.html`, and `Customer_form_template.html`. The code itself is a Python script with syntax highlighting for Django components like `HttpResponse`, `CustomerForm`, and `render`.

```
my_ads > dvdrentalapp > views.py > ...
1 # Create your views here.
2 from django.http import HttpResponseRedirect
3 from django.template import loader
4 from .models import Customer, Rental, Category
5 from django.utils import timezone
6 from django.shortcuts import render, get_object_or_404, redirect
7 from .forms import CustomerForm
8
9 def customer_form_view(request):
10     if request.method == 'POST':
11         form = CustomerForm(request.POST)
12         if form.is_valid():
13             form.save()
14             return redirect('salva') # Altere para a URL de redirecionamento desejada
15     else:
16         form = CustomerForm()
17
18     return render(request, 'Customer_form_template.html', {'form': form})
19
20 def salva(request):
21     return render(request, "salva.html")
```

PROGRAMAÇÃO BACK-END

Na pasta `dvdrentalapp` adicione o seguinte conteúdo no arquivo `urls.py` (linhas 15 e 16)

The screenshot shows a code editor with the file `urls.py` open. The file is part of a Django application structure, as indicated by the imports from `django.contrib.auth` and `django.urls`. The code defines a list of URL patterns for customer-related views. Lines 15 and 16 are specifically highlighted in blue, indicating they are the new additions required.

```
my_ads > dvdrentalapp > urls.py > ...
1  from django.contrib.auth import views as auth_views
2  from django.urls import path
3  from . import views
4
5  urlpatterns = [
6      path('customer/', views.customer, name='mycustomer'),
7      path('detalhes/<int:id>', views.detalhes, name='myDetalhe'),
8      path('edit_customer/<int:customer_id>/', views.edit_customer, name='edit_customer'),
9      path('add_category/', views.add_category, name='add_category'),
10     path('categories/', views.list_categories, name='list_categories'),
11     path('listacustomer/',views.listacustomer, name='listacustomer'),
12     path('listacustomer1/',views.listacustomer1, name='listacustomer1'),
13     path('login/', auth_views.LoginView.as_view(), name='login'),
14     path('logout/', auth_views.LogoutView.as_view(), name='logout'),
15     path('customer_form/', views.customer_form_view, name='customer_form_view'),
16     path('salva', views.salva, name='salva'),
17     # Adicione outras URLs conforme necessário
18 ]
```

PROGRAMAÇÃO BACK-END

**Na pasta `dvdrentalapp` crie o arquivo `forms.py`
adicone o seguinte conteúdo no arquivo:**

```
forms.py  X  salva.html  views.py  urls.py  Customer_form_template.html  login.html

my_ads > dvdrentalapp > forms.py > ...
1   from django.forms import ModelForm
2   from .models import Customer
3
4   class CustomerForm(ModelForm):
5       class Meta:
6           model = Customer
7           #fields = '__all__'
8           fields = ['first_name', 'last_name', 'email', 'address', 'activebool','store_id','create_date']
9
```

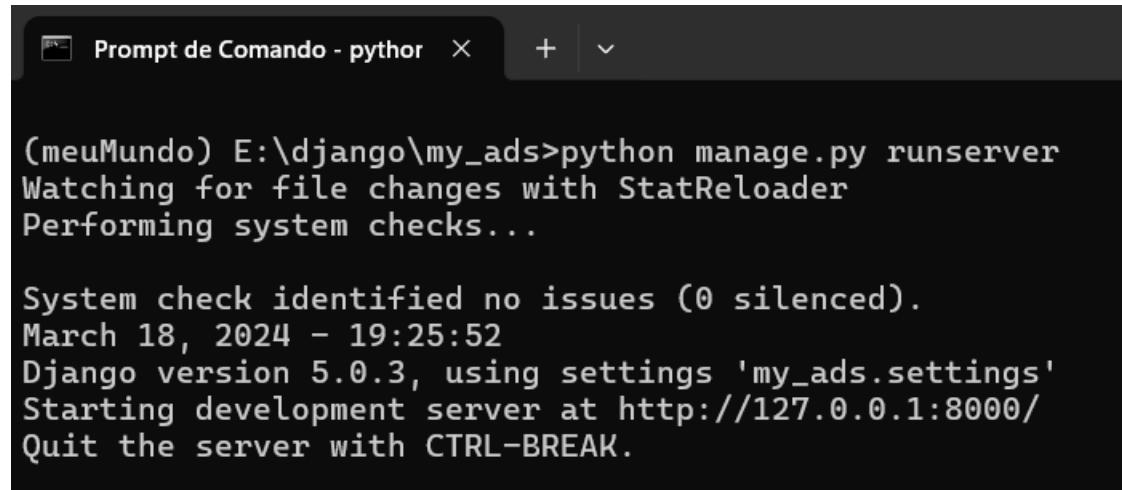
PROGRAMAÇÃO BACK-END

Iniciando o servidor
e testando

PROGRAMAÇÃO BACK-END

Execute na pasta my_ads o comando:

python manage.py runserver



```
(meuMundo) E:\django\my_ads>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

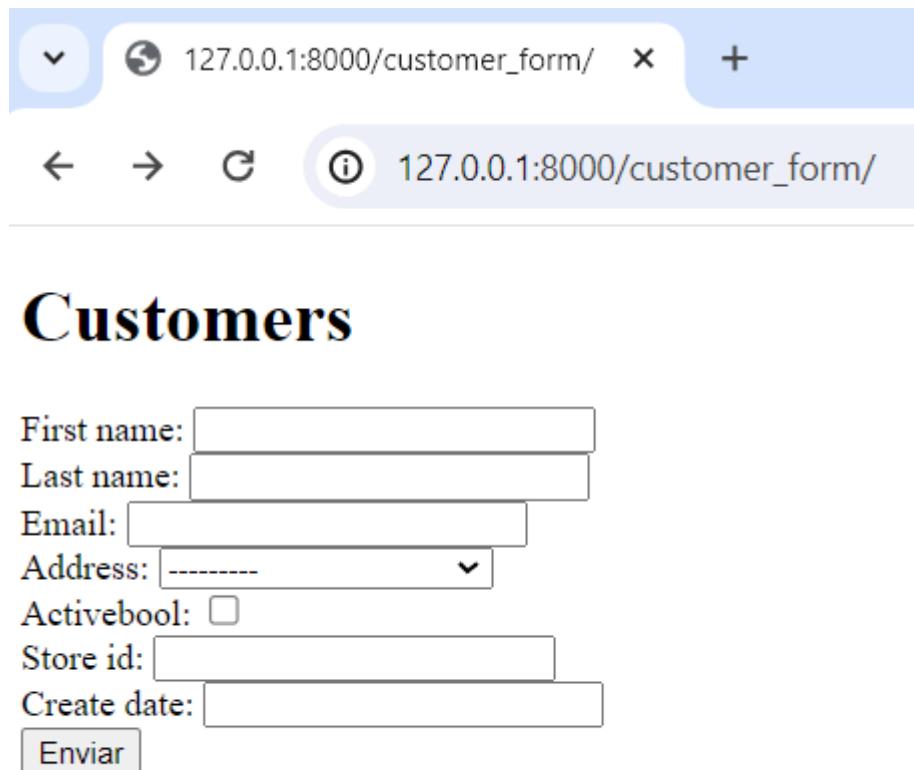
System check identified no issues (0 silenced).
March 18, 2024 - 19:25:52
Django version 5.0.3, using settings 'my_ads.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Vá ao navegador e digite:

http://127.0.0.1:8000/customer_form/

PROGRAMAÇÃO BACK-END

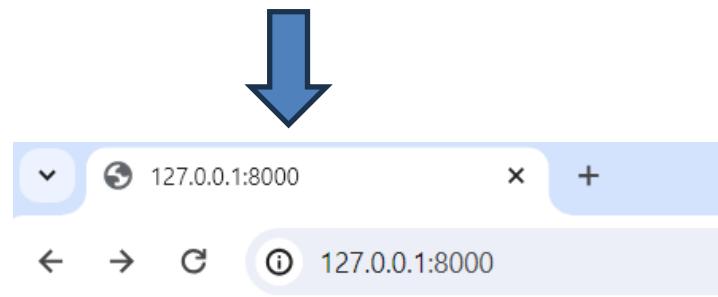
O resultado da página será o formulário para cadastrar um novo customer.



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:8000/customer_form/`. The page content is titled "Customers" and contains a form with the following fields:

- First name: [text input]
- Last name: [text input]
- Email: [text input]
- Address: [text input]
- Activebool: [checkbox]
- Store id: [text input]
- Create date: [text input]
- Enviar [button]

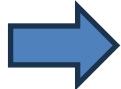
Ao inserir os dados e enviar será mudado para:



Salvo com sucesso

PROGRAMAÇÃO BACK-END

Na listagem aparecerá o novo customer



- 599 - Austin Cintron
- 524 - Jared Ely1
- 2 - Patricia Johnson1
- 600 - Marcio Johnson1
- 605 - Marcio Smith1

PROGRAMAÇÃO BACK-END

Tarefa

PROGRAMAÇÃO BACK-END

Faça as alterações para a tabela dos Category