







Tutorial Django: Sistema RH com App Funcionários



O que vamos construir

Um Sistema RH Básico com Django que inclui:

-  Página web simples com formulário para cadastrar funcionários
 -  Lista básica que mostra os funcionários cadastrados
 -  Endpoint JSON para acessar os dados em formato API
 -  Design limpo com CSS simples
-



Configuração do Ambiente

Objetivo: Preparar o ambiente de desenvolvimento com todas as ferramentas necessárias.

Explicação: Vamos criar um ambiente virtual para isolar as dependências do nosso projeto. Isso evita conflitos com outras versões de bibliotecas que você possa ter no seu computador.

```
bash
```

```
# 1. Criar pasta do projeto
```

```
mkdir sistema-rh
```

```
cd sistema-rh
```

```
# 2. Criar ambiente virtual
```

```
python -m venv venv
```

```
# 3. Ativar ambiente virtual
```

```
# Windows:
```

```
venv\Scripts\activate
```

```
# Linux/Mac:
```

```
source venv/bin/activate
```

```
# 4. Instalar Django
```

```
pip install django
```

```
# 5. Verificar instalação
```

```
python -m django --version
```

Criando o Projeto Django

Objetivo: Criar a estrutura base do nosso projeto Django.

Explicação: Um projeto Django é como o "container" principal que vai conter todas as nossas aplicações. O comando `startproject` cria toda a estrutura necessária para começar.

```
bash
```

```
# Criar projeto Sistema RH
```

```
django-admin startproject sistema_rh .
```

```
# Testar servidor
```

```
python manage.py runserver
```

 Acesse: <http://127.0.0.1:8000>

Se ver a página do foguete , está funcionando!

Criando a App "funcionarios"

Objetivo: Criar uma aplicação específica para gerenciar funcionários.

Explicação: No Django, criamos "apps" para funcionalidades específicas. Isso mantém o código organizado. Nossa app `funcionarios` será responsável por tudo relacionado a funcionários.

```
bash
```

```
# Criar app para gerenciar funcionários
```

```
python manage.py startapp funcionarios
```

Registrar a App no Projeto

Explicação: Precisamos dizer ao Django que nossa nova app existe. Fazemos isso adicionando o nome da app na lista `INSTALLED_APPS` do arquivo de configurações.

Em `systema_rh/settings.py`:

```
python

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'funcionarios', # ← NOSSA APP DE FUNCIONÁRIOS
]
```

Criando Endpoint JSON

Objetivo: Criar uma API simples que retorna dados em formato JSON.

Explicação: Endpoints JSON são úteis para integração com outros sistemas ou aplicativos. Neste caso, qualquer pessoa (ou sistema) poderá acessar a lista de funcionários em formato estruturado.

Em `funcionarios/views.py`:

```
python

from django.http import JsonResponse

# Lista para armazenar funcionários
funcionarios = []
```

```
def api_funcionarios(request):  
    """  
    Endpoint que retorna lista de funcionários em JSON  
    Acesse: http://127.0.0.1:8000/api/funcionarios/  
    """  
    data = {  
        'funcionarios': funcionarios,  
        'total': len(funcionarios)  
    }  
  
    return JsonResponse(data)
```



Configurando as URLs

Objetivo: Configurar as rotas para acessar nossas páginas e API.

Explicação: O Django usa um sistema de URLs para direcionar as requisições para as views corretas. Cada app pode ter seu próprio arquivo de URLs, que depois é incluído no projeto principal.

Crie funcionarios/urls.py:

```
python  
  
from django.urls import path  
from . import views  
  
urlpatterns = [  
    path('api/funcionarios/', views.api_funcionarios,  
        name='api_funcionarios'),  
]
```

Em sistema_rh/urls.py:

```
python  
  
from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('', include('funcionarios.urls')),  
]
```

```
path('admin/', admin.site.urls),
path('', include('funcionarios.urls')), # ← URLs dos funcionários
```

```
]
```

🌐 Teste: <http://127.0.0.1:8000/api/funcionarios/>

Deve retornar JSON com lista vazia.

Criando Página Principal

Objetivo: Criar a página web onde os usuários vão interagir com o sistema.

Explicação: A view `pagina_principal` faz duas coisas: processa o formulário quando enviado (método POST) e mostra a página com a lista de funcionários (método GET). Usamos `redirect` para evitar que o formulário seja reenviado acidentalmente.

Em `funcionarios/views.py`, atualize:

```
python
```

```
from django.shortcuts import render, redirect
from django.http import JsonResponse
```

```
# Lista para armazenar funcionários
funcionarios = []
```

```
def pagina_principal(request):
    """
    Página principal do Sistema RH
    Acesse: http://127.0.0.1:8000/
    """
    if request.method == 'POST':
        # Processar formulário
        nome = request.POST.get('nome')
        cargo = request.POST.get('cargo')
        departamento = request.POST.get('departamento')

        if nome and cargo: # Campos obrigatórios
            novo_funcionario = {
```

```

        'id': len(funcionarios) + 1,
        'nome': nome,
        'cargo': cargo,
        'departamento': departamento
    }
    funcionarios.append(novo_funcionario)
    return redirect('pagina_principal')

context = {
    'funcionarios': funcionarios,
    'total': len(funcionarios)
}
return render(request, 'funcionarios/index.html', context)

def api_funcionarios(request):
    """
    Endpoint JSON da API
    """
    data = {
        'funcionarios': funcionarios,
        'total': len(funcionarios)
    }

    return JsonResponse(data)

```

Atualize funcionarios/urls.py:

```

python

from django.urls import path
from . import views

urlpatterns = [
    path('', views.pagina_principal, name='pagina_principal'),
    path('api/funcionarios/', views.api_funcionarios,
name='api_funcionarios'),

]

```

Configurando Arquivos Estáticos

Objetivo: Adicionar estilos CSS à nossa página.

Explicação: Arquivos estáticos (CSS, JavaScript, imagens) são servidos separadamente pelo Django. Precisamos configurar onde esses arquivos ficam e como são acessados.

Em sistema_rh/settings.py, verifique:

```
python

STATIC_URL = '/static/'

STATICFILES_DIRS = [BASE_DIR / 'static']
```

Crie a estrutura de pastas:

```
text

sistema_rh/
├── static/
│   ├── css/
│   │   └── style.css
├── templates/
│   └── funcionarios/
│       └── index.html
```

Crie static/css/style.css:

```
css

/* Estilos básicos para o Sistema RH */
body {
    font-family: Arial, sans-serif;
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
    background-color: #f5f5f5;
}

.header {
    background: white;
    padding: 20px;
    border-radius: 5px;
    margin-bottom: 20px;
    text-align: center;
```

```
}

.form-section {
    background: white;
    padding: 20px;
    border-radius: 5px;
    margin-bottom: 20px;
}

.form-group {
    margin-bottom: 15px;
}

.form-group label {
    display: block;
    margin-bottom: 5px;
    font-weight: bold;
}

.form-group input,
.form-group select {
    width: 100%;
    padding: 8px;
    border: 1px solid #ddd;
    border-radius: 3px;
}

.btn-submit {
    background: #007bff;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}

.lista-section {
    background: white;
    padding: 20px;
    border-radius: 5px;
}
```



```
.funcionario-item {  
    padding: 10px;  
    border: 1px solid #eee;  
    margin-bottom: 10px;  
    border-radius: 3px;  
}
```

```
.funcionario-id {  
    background: #6c757d;  
    color: white;  
    padding: 2px 8px;  
    border-radius: 10px;  
    font-size: 0.8em;  
}
```

Template HTML Simples

Objetivo: Criar a interface que os usuários vão ver e interagir.

Explicação: Templates Django usam uma sintaxe especial (entre {% %} e {{ }}) para inserir dados dinâmicos e lógica. O {% csrf_token %} é importante para segurança em formulários.

Crie templates/funcionarios/index.html:

```
html  
  
<!DOCTYPE html>  
<html lang="pt-br">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Sistema RH</title>  
    {% load static %}  
    <link rel="stylesheet" href="{% static 'css/style.css' %}">  
</head>  
<body>  
    <div class="header">  
        <h1> Sistema RH</h1>
```

```

    <p>Total de funcionários: <strong>{{ total }}</strong></p>
</div>

<!-- Formulário de Cadastro -->
<div class="form-section">
    <h2>Cadastrar Funcionário</h2>
    <form method="post">
        {% csrf_token %}

        <div class="form-group">
            <label for="nome">Nome:</label>
            <input type="text" id="nome" name="nome" required>
        </div>

        <div class="form-group">
            <label for="cargo">Cargo:</label>
            <input type="text" id="cargo" name="cargo" required>
        </div>

        <div class="form-group">
            <label for="departamento">Departamento:</label>
            <select id="departamento" name="departamento">
                <option value="">Selecione...</option>
                <option value="TI">TI</option>
                <option value="RH">RH</option>
                <option value="Vendas">Vendas</option>
                <option value="Marketing">Marketing</option>
            </select>
        </div>

        <button type="submit" class="btn-submit">Adicionar
Funcionário</button>
    </form>
</div>

<!-- Lista de Funcionários -->
<div class="lista-section">
    <h2>Funcionários Cadastrados</h2>

    {% if funcionarios %}
        {% for funcionario in funcionarios %}
            <div class="funcionario-item">

```

```
<strong>{{ funcionario.nome }}</strong>
<br>
{{ funcionario.cargo }}
{% if funcionario.departamento %}
- {{ funcionario.departamento }}
{% endif %}
<span class="funcionario-id">#{{ funcionario.id }}</span>
</div>
{% endfor %}
{% else %}
<p>Nenhum funcionário cadastrado.</p>
{% endif %}
</div>
</body>

</html>
```

Testando o Sistema

Objetivo: Verificar se tudo está funcionando corretamente.

Explicação: É importante testar cada parte do sistema para garantir que todas as funcionalidades estão working como esperado.

bash

```
python manage.py runserver
```

Checklist de Testes:

1. Página Principal

- Acesse: <http://127.0.0.1:8000/>
- Deve mostrar formulário e lista vazia

2. Cadastrar Funcionário

- Preencha:

- Nome: "Maria Silva"
- Cargo: "Analista"
- Departamento: "TI"
- Clique em "Adicionar Funcionário"
- Funcionário deve aparecer na lista com ID #1

3. 🛠️ Testar API JSON

- Acesse: <http://127.0.0.1:8000/api/funcionarios/>
- Deve retornar JSON com o funcionário cadastrado

4. ↻️ Adicionar Mais Funcionários

- Adicione 2-3 funcionários
- Verifique se a lista atualiza
- Verifique se a API mostra todos



Estrutura Final do Projeto

text

```
sistema_rh/
├── manage.py
├── sistema_rh/                # Configurações do projeto
│   ├── settings.py
│   └── urls.py
├── funcionarios/             # Nossa app
│   ├── views.py
│   └── urls.py
├── templates/
│   └── funcionarios/
│       └── index.html
├── static/
│   └── css/
│       └── style.css
```

Checklist Final do Aluno

Configuração

- Ambiente virtual criado e ativado
- Django instalado
- Projeto `sistema_rh` criado
- App `funcionarios` criado
- App registrada em `settings.py`







Funcionalidades

- Endpoint JSON funcionando (`/api/funcionarios/`)
- Página principal criada (`/`)
- Formulário adiciona funcionários
- Lista mostra funcionários cadastrados
- CSS básico aplicado

Testes

- Página principal carrega
- Formulário adiciona funcionário
- Lista atualiza automaticamente
- API JSON retorna dados corretos
- Múltiplos funcionários funcionam

O que foi aprendido:

-  Criar projeto e apps Django
-  Configurar URLs e views
-  Criar templates HTML
-  Usar arquivos estáticos (CSS)
-  Trabalhar com formulários
-  Criar endpoints API simples

 Sistema RH completo e funcionando!