Django

Começando

* Instalar o Django

#### pip install Django

* Criar um novo projeto

#### django-admin startproject exemplo

* Criar uma aplicação

#### py manage.py startapp exemplo\_app

* Iniciar o servidor interno (localhost:8000 ou 127.0.0.1:8000)

#### py manage.py runserver

Models

* Servem para trabalhar com banco de dados sem precisar usar SQL diretamente
* **Exemplo de model (vai na pasta models.py da aplicação):**

#### from django.db import models

#### class Person(models.Model):

#### name = models.CharField(max\_length=30)

#### age = models.IntegerField()

#### birthdate = models.DateField()

#### is\_active = models.BooleanField(default=True)

* CharField: coluna de VARCHAR;
* IntegerField: coluna de INTEGER;
* DateField: coluna de DATETIME;
* BooleanField: coluna de BOOLEAN.
* **Para criar a tabela efetivamente, rodar os comandos:**

#### python manage.py makemigrations

(cria o arquivo de migração)

#### python manage.py migrate

(aplica as mudanças no banco [commit])

* **Para realizar consultas (queries):**

#### all\_persons = Person.objects.all()

(traz todas as Persons de um model)

#### persons\_over\_30 = Person.objects.filter(age\_\_gte=30)

(traz todas as Persons com idade maior ou igual a 30)

#### persons\_sorted\_by\_name = Person.objects.order\_by(“name”)

(consulta com ordenação)

#### first\_ten\_persons = Person.objects.all()[:10]

(consulta com limite de resultados)

* **Para realizar operações CRUD:**
* **Criar um objeto:**

#### person = Person(name=”John”, age=25, birthdate=”2000-01-01”)

#### person.save()

* **Obter um objeto:**

#### person = Person.objects.get(name=”John”)

* **Atualizar um objeto:**

#### person = Person.objects.get(name=”John”)

#### person.age = 30

#### person.save()

* **Deletar um objeto:**

#### person = Person.objects.get(name=”John”)

#### person.delete()

Templates

* Misturam HTML e Django. Criar um arquivo HTML na pasta templates (criá-la se não existir) da aplicação.
* **Exemplo de template:**

#### <!DOCTYPE html>

#### <html lang=”pt-br”>

#### <head>

#### <meta charset=”utf-8” />

#### <meta name=”viewport” content=”width=device-width, initial-scale=1.0” />

#### <title>Meu aplicativo Django</title>

#### </head>

#### <body>

#### <h1>Hello {{name}}</h1>

#### </body>

#### </html>

* **Para renderizar o template (em views.py):**

#### from django.shortcuts import render

#### def greet(request, name):

#### return render(request, “index.html”, {“name”: name})

* **Para criar uma URL personalizada (em urls.py):**

#### from exemplo\_app.views import greet

#### urlpatterns = [

#### ...

#### path(“/greet/<str:name>/”, greet, name=”greet”),

#### ...

#### ]

Views (em views.py) - tipos

* **Function-based view:** função que recebe uma requisição e retorna uma resposta.

#### from django.http import HttpResponse

#### def my\_view(request):

#### return HttpResponse(“Hello, world!”)

* **Class-based view:** views baseadas em classes.

#### from django.views.generic import View

#### from django.http import HttpResponse

#### class MyView(View):

#### def get(self, request):

#### return HttpResponse(“Hello, World!”)

* **Template view:** retorna páginas HTML.

#### from django.views.generic.base import TemplateView

#### class MyView(TemplateView):

#### template\_name = “index.html”

* **View de formulário:** valida e processa dados de um formulário.

#### from django.views.generic.edit import FormView

#### from myapp.forms import MyForm

#### class MyView(FormView):

#### form\_class = MyForm

#### template\_name = “my\_template.html”

#### success\_url = “/success/”

#### def form\_valid(self, form):

#### form.send\_email()

#### return super().form\_valid(form)

* **View de listagem:** view que exibe uma lista de objetos.

#### from django.views.generic.list import ListView

#### from myapp.models import MyModel

#### class MyView(ListView):

#### model = MyModel

#### template\_name = “index.html”

* **View de detalhes:** view que exibe detalhes de um objeto específico.

#### from django.views.generic.detail import DetailView

#### from myapp.models import MyModel

#### class MyView(DetailView):

#### model = MyModel

#### template\_name = “index.html”

Formulários e validação de dados

* **Criando formulários:**

#### from django import forms

#### class PersonForm(forms.Form):

#### name = forms.CharField(max\_length=30)

#### age = forms.IntegerField()

#### birthdate = forms.DateField()

#### is\_active = forms.BooleanField(required=False)

* **Gerenciando dados de formulários:**
* **Enviando os dados para o servidor:**

#### def submit\_form(request):

#### name = request.POST.get(“name”)

#### age = request.POST.get(“age”)

#### birthdate = request.POST.get(“birthdate”)

#### is\_active = request.POST.get(“is\_active”)

#### # Processamento dos dados

* **Validando os dados**
* **Validação de e-mail:**

#### email = forms.EmailField()

* **Validação personalizada:**

#### def clean\_age(self):

#### age = self.cleaned\_data.get(“age”)

#### if age < 18:

#### raise forms.ValidationError(“You must be at least 18 years old to register.”)

#### return age

* **Enviando e validando os dados para o servidor:**

#### def submit\_form(request):

#### form = PersonForm(request.POST)

#### if form.is\_valid():

#### name = form.cleaned\_data.get(“name”)

#### age = form.cleaned\_data.get(“age”)

#### birthdate = form.cleaned\_data.get(“birthdate”)

#### is\_active = form.cleaned\_data.get(“is\_active”)

#### # Processamento dos dados

#### else:

#### # Exibir erros de validação

Autenticação e autorização (login)

* **Criando usuários:**

#### from django.contrib.auth.models import User

#### user = User.objects.create\_user(username=”johndoe”, password=”password”)

* **Criando super-usuários:**

#### from django.contrib.auth.models import User

#### user = User.objects.create\_superuser(username=”johndoe”, password=”password”, email=”johndoe@example.com”)

* **Gerenciando sessões:**
* **Fazer login:**

#### from django.contrib.auth import authenticate, login

#### def login\_view(request):

#### username = request.POST.get(“username”)

#### password = request.POST.get(“password”)

#### user = authenticate(request, username=username, password=password)

#### if user is not None:

#### login(request, user)

#### # Redireciona para página de sucesso

#### else:

#### # Retorna mensagem de erro

* **Fazer logout:**

#### from django.contrib.auth import logout

#### def logout\_view(request):

#### logout(request)

#### # Redireciona para página de login

* **Definindo permissões:**
* **Criando um grupo e atribuindo permissões:**

#### from django.contrib.auth.models import Group, Permission

#### from django.contrib.contenttypes.models import ContentType

#### group = Group.objects.create(name=”editors”)

#### content\_type = ContentType.objects.get\_for\_model(MyModel)

#### permission = Permission.objects.create(codename=”can\_edit”, name=”Can Edit”, content\_type=content\_type)

#### group.permissions.add(permission)

Administração do Django

* Interface gráfica para gerenciar o conteúdo do aplicativo.
* **Como acessar:**
* **Adicioná-lo as configurações (em settings.py)**

#### INSTALLED\_APPS = [

#### ...

#### “django.contrib.admin”,

#### ]

* **Criar um usuário com permissões de administrador**

#### python manage.py createsuperuser

* **Acessar a url /admin/ e introduzir usuário e senha**

REST Framework

* Biblioteca que cria APIs REST.
* **Instalar:**

#### pip install djangorestframework

* **Adicionar às configurações:**

#### INSTALLED\_APPS = [

#### ...

#### “rest\_framework” ,

#### ]

* **Criar as views de API:**

#### from rest\_framework.views import APIView

#### from rest\_framework.response import Response

#### class MyAPI(APIView):

#### def get(self, request):

#### data = {“hello”: “world”}

#### return Response(data)

* **Autenticação com token:**

#### from rest\_framework.authentication import TokenAuthentication

#### from rest\_framework.permissions import IsAuthenticated

#### class MyAPI(APIView):

#### authentication\_classes = [TokenAuthentication]

#### permission\_classes = [IsAuthenticated]

* **Serializer (converter dados Python em JSON ou XML e vice-versa):**

#### from rest\_framework import serializers

#### class MySerializer(serializers.Serializer):

#### name = serializers.CharField()

#### age = serializers.IntegerField()

* **Validando dados de entrada e gerando saídas formatadas com os serializers:**

#### class MyAPI(APIView):

#### def post(self, request):

#### serializer = MySerializer(data=request.data)

#### if serializer.is\_valid():

#### name = serializer.data.get(“name”)

#### age = serializer.data.get(“age”)

#### # Processamento dos dados

#### return Response(serializer.data)

#### return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

Serializers

* **Criando um serializer:**

#### from rest\_framework import serializers

#### class MySerializer(serializers.Serializer):

#### name = serializers.CharField()

#### age = serializers.IntegerField()

* **Usando o serializer:**

#### serializer = MySerializer(data={“name”: “John”, “age”: 30})

#### serializer.is\_valid()

#### serializer.save()

* **Serializers aninhados:**

#### class PersonSerializer(serializers.Serializer):

#### name = serializers.CharField()

#### age = serializers.IntegerField()

#### class GroupSerializer(serializers.Serializer):

#### name = serializers.CharField()

#### members = PersonSerializer(many=True)

* **Serializer com informação opcional:**

#### class MySerializer(serializers.Serializer):

#### name = serializers.CharField()

#### age = serializers.IntegerField()

#### address = serializers.CharField(required=False)

* **Serializer com métodos personalizados:**

#### class MySerializer(serializers.Serializer):

#### email = serializers.EmailField()

#### def validate\_email(self, value):

#### if User.objects.filter(email=value).exists():

#### raise serializers.ValidationError(“E-mail já está em uso”)

#### return value