Django

Prof. José Matheus

O que é uma API?

Uma API (Interface de Programação de Aplicações) estabelece as diretrizes para interação com outros sistemas de software. Desenvolvedores disponibilizam APIs para permitir que outras aplicações se comuniquem programaticamente com as suas. Por exemplo, uma aplicação de registro de horas oferece uma API que requisita o nome completo de um funcionário e um intervalo de datas. Após receber esses dados, a aplicação processa internamente a planilha de horas do funcionário, retornando o total de horas trabalhadas durante o período especificado.

O que é uma REST?

O Representational State Transfer (REST) é uma arquitetura de software que estabelece diretrizes para o funcionamento de uma API. Concebida para gerenciar a comunicação em redes complexas, como a internet, a REST oferece uma abordagem que permite comunicação confiável e eficiente em larga escala. Sua implementação facilita modificações e garante visibilidade e portabilidade entre plataformas para qualquer sistema de API.

Principais Características de APIs REST

Baseada em Padrões HTTP: Utiliza métodos HTTP padrão (GET, POST, PUT, DELETE) para operações;

Stateless (Sem Estado): Cada requisição do cliente contém todas as informações necessárias para entendê-la e processá-la;

Recursos Identificáveis: Os recursos (dados ou serviços) são identificados por URLs únicas.

Benefícios para Desenvolvedores

Simplicidade: Projeto simplificado com operações CRUD padronizadas;

Facilidade de Integração: Permite a integração eficiente entre sistemas;

Escalabilidade: Adapta-se bem a ambientes distribuídos e escaláveis.

Melhores Práticas ao Implementar APIs REST

Versionamento: Utilizar versões para garantir compatibilidade com mudanças;

Segurança: Aplicar autenticação e autorização adequadas;

Documentação Clara: Fornecer documentação compreensível para facilitar o uso por outros desenvolvedores.

Evolução e Atualização de APIs REST

Versionamento Gradual: Introduzir alterações gradualmente para evitar quebras bruscas;

Manutenção de Compatibilidade: Garantir suporte a versões anteriores para não impactar clientes existentes;

Comunicação Transparente: Informar desenvolvedores sobre mudanças planejadas e oferecer transição suave.

Cliente x Servidor

- **Sem Estado (Stateless):**

  REST: Cada requisição é independente, não armazena o estado do cliente no servidor entre solicitações.

  React: Os componentes React são geralmente stateful (mantêm estado interno), mas as requisições ao servidor são stateless. A persistência de estado no frontend é gerenciada internamente pelo React.

- **Métodos HTTP:**

  REST: Utiliza métodos HTTP padrão (GET, POST, PUT, DELETE) para realizar operações em recursos.

  React: Ao interagir com APIs no frontend, utiliza principalmente os métodos HTTP padrão para buscar, enviar, e manipular dados.

- **Comunicação com a API:**

  REST: Requisições são feitas para endpoints específicos da API para realizar operações nos recursos.

  React: Utiliza bibliotecas como axios ou fetch para fazer requisições HTTP aos endpoints da API e obter/atualizar dados.

Cliente x Servidor

- **Atualização do UI (User Interface):**

  REST: O servidor responde com dados que são utilizados para atualizar o UI conforme necessário.

  React: Após a resposta da API, o React atualiza dinamicamente os componentes relevantes do UI com os dados obtidos.

- **Gerenciamento de Estado:**

  REST: O estado é gerenciado no servidor e as requisições devem incluir todas as informações necessárias.

  React: Mantém estados internos para componentes, facilitando a reatividade e atualizações do UI.

- **Async/Await:**

  REST: Faz uso de conceitos assíncronos para lidar com operações que podem levar tempo, como requisições a banco de dados.

  React: As requisições a APIs também são assíncronas e muitas vezes são tratadas usando async/await para lidar com operações assíncronas de maneira mais síncrona no código.

Django Rest Framework

Django Rest Framework

O Django REST Framework é uma poderosa extensão do Django que facilita a criação de APIs RESTful para suas aplicações web. Ele fornece um conjunto de ferramentas e recursos para facilitar a implementação de endpoints da API, serialização de dados, autenticação, autorização e muito mais.

Django Rest Framework

Alguns recursos-chave do Django REST Framework

Serialização de dados: O DRF oferece uma camada de serialização de dados que permite converter objetos Python em formatos de dados como JSON, XML ou YAML, e vice-versa.

Viewsets e Roteamento: O DRF fornece abstrações chamadas Viewsets que mapeiam operações CRUD (Create, Retrieve, Update, Delete) para uma API.

Autenticação e autorização: O DRF oferece suporte a vários esquemas de autenticação, como token, sessão, autenticação básica, JWT (JSON Web Tokens) e OAuth.

Django Rest Framework

Paginação: O DRF facilita a paginação dos resultados da API, permitindo que você divida as respostas em páginas para melhorar o desempenho e a experiência do usuário.

Documentação automática: O DRF gera automaticamente documentação interativa da API com base em suas views, serializers e modelos.

Django Rest Framework: ViewSet x CBV

A diferença principal entre ViewSet e Class-Based View (CBV) no Django está relacionada à abordagem e à flexibilidade oferecida para a criação de views.

Class-Based View (CBV): As Class-Based Views são uma abordagem mais antiga e tradicional para definir views no Django. Elas são classes Python que encapsulam a lógica da view em métodos, e cada método corresponde a uma ação específica (por exemplo, GET, POST, etc.). As CBVs proporcionam uma estrutura organizada e reutilizável para as views.

Django Rest Framework: ViewSet x CBV

ViewSet: Os ViewSets fazem parte do Django Rest Framework (DRF) e são uma abordagem mais especializada para views, especialmente voltada para APIs RESTful. Eles são projetados para trabalhar com modelos de banco de dados e fornecem automaticamente operações CRUD (Create, Read, Update, Delete) para esses modelos.

Django Rest Framework: ViewSet x CBV

Abstração de CRUD:

CBV: Requer implementação manual das operações CRUD, geralmente utilizando métodos como get(), post(), put(), etc.

ViewSet: Fornece automaticamente operações CRUD baseadas no modelo, economizando trabalho manual.

Flexibilidade vs. Conveniência:

CBV: Oferece maior flexibilidade e controle sobre a lógica da view, permitindo personalizações específicas.

ViewSet: Mais conveniente para operações CRUD padrão, sendo especialmente útil ao construir APIs RESTful.

Django Rest Framework: ViewSet x CBV

Estrutura e Padrões:

CBV: Pode levar a uma estrutura mais complexa, exigindo mais código para funcionalidades específicas.

ViewSet: Siga as convenções do DRF, facilitando a criação rápida de endpoints RESTful e padrões consistentes.

Django Rest Framework: ViewSet x CBV

Escolha entre CBV e ViewSet:

Use Class-Based Views quando precisar de controle granular sobre a lógica da view e operações personalizadas.

Use ViewSets quando estiver construindo APIs RESTful e desejar operações CRUD automáticas baseadas em modelos.

Observação: *O Django Rest Framework também permite a criação de views híbridas, combinando características de CBVs e ViewSets para flexibilidade adicional.*

Django Rest Framework: Instalação

Passo 1: Instale o Django REST Framework (DRF) usando o pip

```
pip install djangorestframework
```

Passo 2: Adicione '**rest_framework**' à lista de aplicativos em seu arquivo settings.py

Passo 3: Crie um diretório chamado “api”, dentro de sua app “minha_app” e crie três arquivos: **serializers.py**, **router.py** e **viewsets.py**

Django Rest Framework: Serializer

Passo 4: Adicione o seguinte código ao arquivo minha_app/api/serializers.py:

PYTHON

```python
from rest_framework import serializers

from minha_app import models


class MeuModeloSerializer(serializers.ModelSerializer):

    class Meta:

        model = models.MeuModelo

        fields = '__all__'
```

Django Rest Framework: ViewSet

Passo 5: Adicione o seguinte código ao arquivo minha_app/api/viewsets.py:

PYTHON

```python
from rest_framework import viewsets

from minha_app import models

from minha_app.api import serializers


class MeuModeloViewSet(viewsets.ModelViewSet):

    queryset = models.MeuModelo.objects.all()

    serializer_class = serializers.MeuModeloSerializer
```

Django Rest Framework: Router

Passo 6: Adicione o seguinte código ao arquivo minha_app/api/router.py

PYTHON

```python
from rest_framework import routers

from minha_app.api import viewsets


meu_modelo_router = routers.DefaultRouter()

meu_modelo_router.register( 'meu_modelo', viewsets.MeuModeloViewSet)
```

Django Rest Framework: Router

Um **router** no Django Rest Framework é uma ferramenta que facilita a configuração de URLs para as views de uma API. Ele mapeia automaticamente as ações CRUD para URLs específicos, simplificando a criação de endpoints RESTful.

Por que usar um Router?

1. **Conveniência:** Reduz a necessidade de configurar manualmente URLs para cada view;
2. **Manutenção Simples:** Facilita a adição de novas views e endpoints à medida que o projeto cresce;
3. **Padronização:** Ajuda a manter uma estrutura consistente para a API.

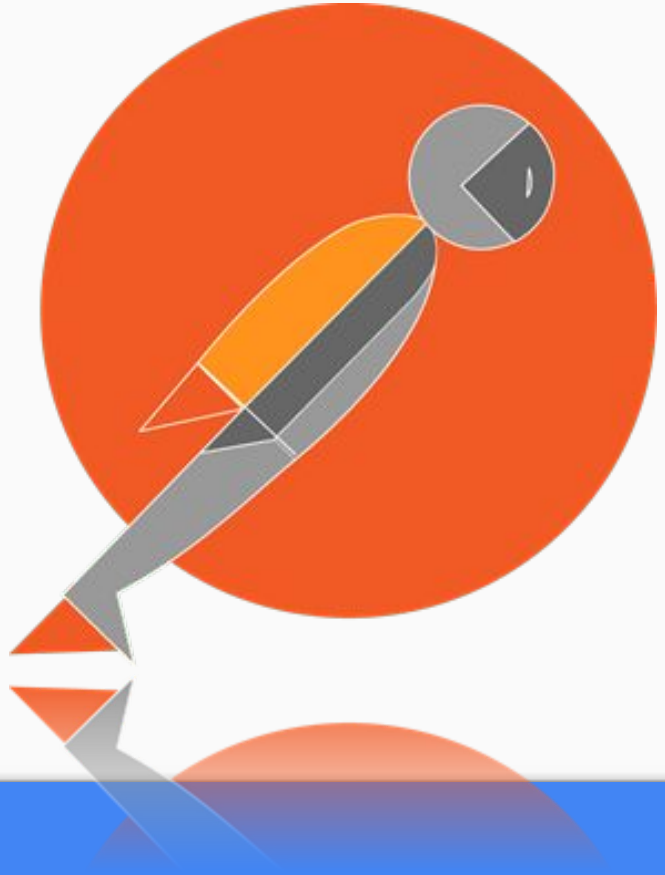
Django Rest Framework

Passo 7: Configuração de URL

Adicionar a seguinte configuração ao arquivo meu_projeto/urls.py

PYTHON

```python
from django.conf import settings

from django.urls import path, include

from django.conf.urls.static import static

from minha_app.api.router import
meu_modelo_router


urlpatterns = [

   # ... suas outras URLs …

   path('api/',
include(meu_modelo_router.urls)) ,

] + static(settings.MEDIA_URL ,
document_root=settings.MEDIA_ROOT)
```



Testando no Postman

Django Rest Framework

Configurando paginação e documentação:

Instale a biblioteca “Django filters”:

pip install django-filter

Adicionar ao settings, na parte dos APPS: ‘django_filters’

PYTHON

```python
REST_FRAMEWORK = {

    "DEFAULT_FILTER_BACKENDS" :
("django_filters.rest_framework.DjangoFilterBack
end",),

    "DEFAULT_SCHEMA_CLASS" :
"rest_framework.schemas.coreapi.AutoSchema" ,

    "DEFAULT_PAGINATION_CLASS" :
"rest_framework.pagination.PageNumberPagination"
,

    "NON_FIELD_ERRORS_KEY" : "error",

    "PAGE_SIZE": 5

}
```

Dúvidas?

Referências

- **Documentação do Django:** https://docs.djangoproject.com/
- **Documentação do Django Rest Framework:** https://www.django-rest-framework.org/
- **Pillow:** https://pypi.org/project/Pillow/
- **O que é a API RESTful?:** https://aws.amazon.com/pt/what-is/restful-api/