Dia 3
Python FastAPI

Introdução a web

A web é baseada primariamente no protocolo HTTP que por especificação serve para transferir texto de um servidor para um cliente.

O texto que trafega do servidor para o cliente é chamado de Hiper Texto, ou seja, texto onde além de palavras e letras, contém também marcações semânticas e links para outros textos.

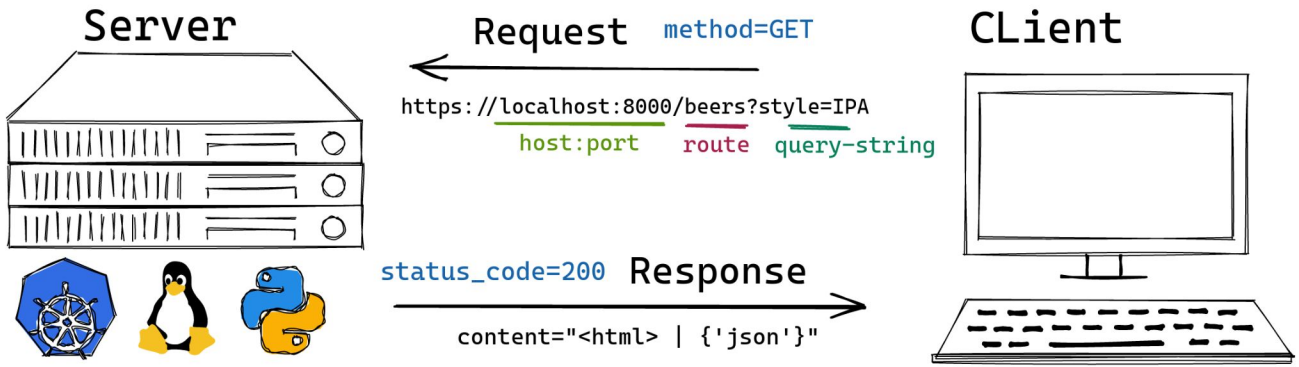
Esse foi o design original, hoje em dia usamos esse texto para trafegar todo tipo de conteúdo.

Principalmente HTML (junto com CSS e JS) e/ou JSON, além de streams de texto que representam arquivos binários como imagens, vídeos e áudios.

Introdução a web

As palavras chave que você vai utilizar ao trabalhar com web são:

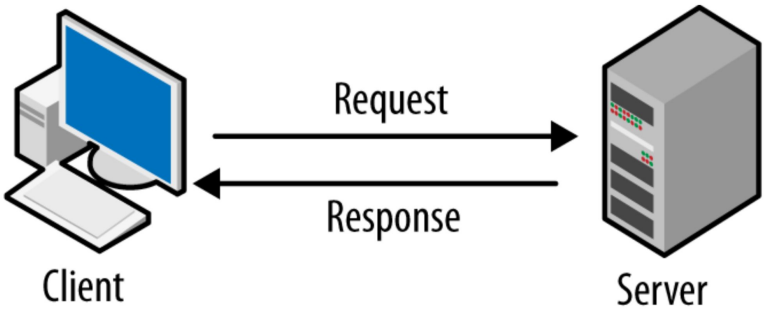
- Servidor
- Cliente
- URL
- Métodos HTTP
- Request
- Response
- Payload



O QUE É UMA API

APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos. Por exemplo, o sistema de software do instituto meteorológico contém dados meteorológicos diários.

O aplicativo meteorológico em seu telefone “fala” com este sistema por meio de APIs e mostra atualizações meteorológicas diárias no telefone.



O QUE SIGNIFICA API?

API significa Application Programming Interface (Interface de Programação de Aplicação). No contexto de APIs, a palavra Aplicação refere-se a qualquer software com uma função distinta.

A interface pode ser pensada como um contrato de serviço entre duas aplicações. Esse contrato define como as duas se comunicam usando solicitações e respostas. A documentação de suas respectivas APIs contém informações sobre como os desenvolvedores devem estruturar essas solicitações e respostas.

Como as APIs funcionam?

A arquitetura da API geralmente é explicada em termos de cliente e servidor. A aplicação que envia a solicitação é chamada de cliente e a aplicação que envia a resposta é chamada de servidor. Então, no exemplo do clima, o banco de dados meteorológico do instituto é o servidor e o aplicativo móvel é o cliente.

Existem quatro maneiras diferentes pelas quais as APIs podem funcionar, dependendo de quando e por que elas foram criadas.

APIs SOAP

Essas APIs usam o Simple Object Access Protocol (Protocolo de Acesso a Objetos Simples). Cliente e servidor trocam mensagens usando XML. Esta é uma API menos flexível que era mais popular no passado.

APIs RPC

Essas APIs são conhecidas como Remote Procedure Calls (Chamadas de Procedimento Remoto). O cliente conclui uma função (ou um procedimento) no servidor e o servidor envia a saída de volta ao cliente.

APIs WebSocket

API Websocket é outro desenvolvimento de API da Web moderno que usa objetos JSON para passar dados. Uma API WebSocket oferece suporte à comunicação bidirecional entre aplicativos cliente e o servidor. O servidor pode enviar mensagens de retorno de chamada a clientes conectados, tornando-o mais eficiente que a API REST.

APIs REST

Essas são as APIs mais populares e flexíveis encontradas na Web atualmente. O cliente envia solicitações ao servidor como dados. O servidor usa essa entrada do cliente para iniciar funções internas e retorna os dados de saída ao cliente. Vejamos as APIs REST em mais detalhes abaixo.

O QUE SÃO APIS REST?

REST significa Transferência Representacional de Estado. REST define um conjunto de funções como GET, PUT, DELETE e assim por diante, que os clientes podem usar para acessar os dados do servidor. Clientes e servidores trocam dados usando HTTP.

A principal característica da API REST é a ausência de estado. A ausência de estado significa que os servidores não salvam dados do cliente entre as solicitações. As solicitações do cliente ao servidor são semelhantes aos URLs que você digita no navegador para visitar um site. A resposta do servidor corresponde a dados simples, sem a renderização gráfica típica de uma página da Web.

Quais são os benefícios das APIs REST?

As APIs são usadas para integrar novas aplicações com sistemas de software existentes. Isso aumenta a velocidade de desenvolvimento porque cada funcionalidade não precisa ser escrita do zero. Você pode usar APIs para aproveitar o código existente.

Integração

Setores inteiros podem mudar com a chegada de uma nova aplicação. As empresas precisam responder rapidamente e oferecer suporte à rápida implantação de serviços inovadores. Elas podem fazer isso fazendo alterações no nível da API sem precisar reescrever todo o código.

Inovação

As APIs oferecem uma oportunidade única para as empresas atenderem às necessidades de seus clientes em diferentes plataformas. Por exemplo, a API de mapas permite a integração de informações de mapas por meio de sites, Android, iOS etc. Qualquer empresa pode fornecer acesso semelhante aos seus respectivos bancos de dados internos usando APIs gratuitas ou pagas.

Expansão

A API atua como um gateway entre dois sistemas. Cada sistema é obrigado a fazer alterações internas para que a API não seja afetada. Dessa forma, qualquer alteração futura de código feita por uma parte não afetará a outra parte.

Facilidade de manutenção

Quais são os diferentes tipos de API?

Elas são internos a uma empresa e são usadas apenas para conectar sistemas e dados dentro da empresa.

APIs privadas

Estas são abertas ao público e podem ser usadas por qualquer pessoa. Pode ou não haver alguma autorização e custo associado a esses tipos de APIs.

APIs públicas

Estas são acessíveis apenas por desenvolvedores externos autorizados para auxiliar as parcerias entre empresas.

APIs de parceiros

Estas combinam duas ou mais APIs distintas para atender a requisitos ou comportamentos complexos do sistema.

APIs compostas

O QUE É UM ENDPOINT DE API E POR QUE ELE É IMPORTANTE?

Os endpoints da API são os pontos de contato finais no sistema de comunicação da API. Estes incluem URLs de servidores, serviços e outros locais digitais específicos de onde as informações são enviadas e recebidas entre sistemas.

O que é FastAPI

O FastAPI é um framework Python que implementa o protocolo ASGI (Async Gateway Interface) além de ser super rápido por ser assíncrono ele utiliza as type annotations do Python 3 para garantir a definição de rotas e parâmetros.

https://fastapi.tiangolo.com/

Ideia inicial para Aula

Vamos criar uma API responsável por retornar em texto em formato `JSON` e faremos isso no arquivo chamado `api.py`.

OBS: Em uma aplicação maior a boa prática seria modularizar a `API`, mas para manter a simplicidade vamos manter em um único arquivo.

Criação ambiente Virtual

```
virtualenv -p python3.9 vev
```

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

Instalação Requerimentos

```
pip install fastapi
```

Primeiros passos

O arquivo FastAPI mais simples pode ser assim:

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Copie isso para um arquivo `api.py`

Execute assim: `uvicorn api:app --reload`

Confira

Abra seu navegador em http://0.0.0.1:8000.

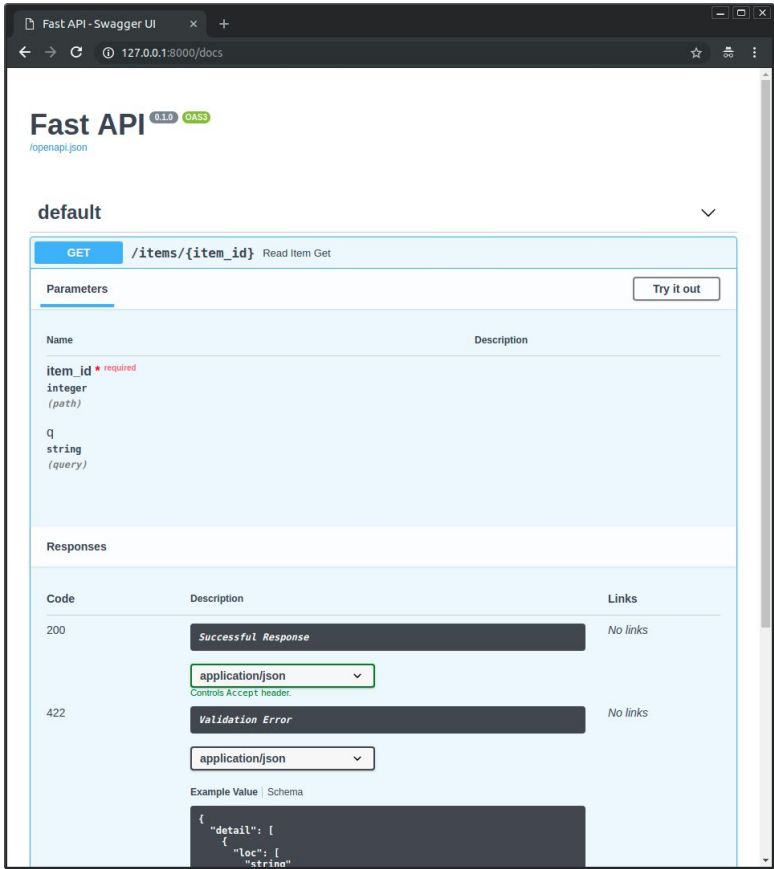
Você verá a resposta JSON como:

```
{"message": "Hello World"}
```

Documentos de API

Agora vá para http://0.0.0.1:8000/docs.

Você verá a documentação da API interativa automática (fornecida pelo Swagger):



Parâmetros

Você pode declarar "parâmetros" ou "variáveis" de caminho com a mesma sintaxe usada por strings de formato Python:

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id):
    return {"item_id": item_id}
```

O valor do parâmetro `item_id` será passado para sua função como argumento `item_id`.

Então, se você executar este exemplo e for para http://0.0.0.1:8000/items/test, você verá uma resposta de:

```
{"item_id":"test"}
```

Parâmetros com tipos

Você pode declarar o tipo de um parâmetro de caminho na função, usando anotações de tipo padrão do `Python`:

Nesse caso, `item_id` é declarado como um int.

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id: int):
    return {"item_id": item_id}
```

Conversão de dados

Se você executar este exemplo e abrir seu navegador em http://127.0.0.1:8000/items/3, você verá uma resposta de:

```
{"item_id":3}
```

Validação dos Dados:

Mas se você for para o navegador em http://127.0.0.1:8000/items/test, você verá um belo erro HTTP de:

Erro HTTP


```
{
    "detail": [
        {
            "loc": [
                "path",
                "item_id"
            ],
            "msg": "value is not a valid integer",
            "type": "type_error.integer"
        }
    ]
}
```

Parâmetros de consulta

Quando você declara outros parâmetros de função que não fazem parte dos parâmetros de caminho, eles são automaticamente interpretados como parâmetros de "consulta".

```python
from fastapi import FastAPI

app = FastAPI()

fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return fake_items_db[skip : skip + limit]
```

Parâmetros de consulta

A consulta é o conjunto de pares de valores-chave que seguem `?` em uma URL, separados por `&`.

Por exemplo, na URL:

`http://0.0.0.1:8000/items/?skip=0&limit=10`

...os parâmetros da consulta são:

- `skip`: com um valor de 0
- `limit`: com um valor de 10

Como fazem parte da URL, são strings "naturalmente".

Mas quando você os declara com tipos Python (no exemplo acima, como int), eles são convertidos para esse tipo e validados em relação a ele.

Corpo da solicitação

Quando você precisa enviar dados de um cliente (digamos, um navegador) para sua API, você os envia como um corpo de solicitação .

Um corpo de solicitação são dados enviados pelo cliente para sua API. Um corpo de resposta são os dados que sua API envia ao cliente.

Sua API quase sempre precisa enviar um corpo de resposta . Mas os clientes não precisam necessariamente enviar corpos de solicitação o tempo todo.

Para declarar um corpo de solicitação , você usa `Pydantic`

Corpo da solicitação

Primeiro, você precisa importar BaseModelde pydantic:

```
from typing import Union

from fastapi import FastAPI
from pydantic import BaseModel
```

Corpo da solicitação

Em seguida, você declara seu modelo de dados como uma classe que herda de BaseModel.

Use tipos padrão do Python para todos os atributos:

```python
from typing import Union

from fastapi import FastAPI
from pydantic import BaseModel


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None
```

Corpo da solicitação

Para adicioná-lo à sua operação, declare-o da mesma maneira que você declarou o caminho e os parâmetros de consulta:

...e declare seu tipo como o modelo que você criou, Item.

```
app = FastAPI()


@app.post("/items/")
async def create_item(item: Item):
    return item
```

Variáveis de Ambiente

Variáveis de ambiente são valores dinâmicos que afetam os programas ou processos que estão em execução em um servidor. Eles existem em todos os sistemas operacionais, sendo que os tipos podem variar. Variantes de ambientes podem ser criadas, editadas, salvadas e apagadas.

Como criar Variáveis de Ambiente?

Basicamente as variáveis de ambiente ficam em um arquivo chamado `.env` é um arquivo individual que contém os valores-chave para todas as variáveis ambientais necessárias em sua aplicação. O arquivo é armazenado localmente sem ser salvo no repositório/github, portanto, você não está colocando informações potencialmente confidenciais em risco.

Banco de Dados

Neste exemplo, usaremos o `SQLite` , pois ele usa um único arquivo e o Python possui suporte integrado. Então, você pode copiar este exemplo e executá-lo como está.

Mais tarde, para seu aplicativo de produção, você pode querer usar um servidor de banco de dados como o `MySQL` .

ORMs

FastAPI funciona com qualquer banco de dados e qualquer estilo de biblioteca para se comunicar com o banco de dados.

Um padrão comum é usar um "ORM": uma biblioteca de "mapeamento relacional de objeto".

Um ORM possui ferramentas para converter ("mapa") entre objetos no código e tabelas de banco de dados ("relações").

Com um ORM, normalmente você cria uma classe que representa uma tabela em um banco de dados SQL, cada atributo da classe representa uma coluna, com um nome e um tipo.

ORMs

Por exemplo, uma classe `Pet` pode representar uma tabela SQL `pets`.

E cada objeto de instância dessa `classe` representa uma linha no `banco de dados`.

Por exemplo, um objeto `orion_cat` (uma instância de `Pet`) pode ter um atributo `orion_cat.type` para a coluna type. E o valor desse atributo pode ser, por exemplo, `"cat"`.

ORMs

E o `ORM` fará todo o trabalho para obter as informações dos proprietários das tabelas correspondentes quando você tentar acessá-las.

Os `ORMs` comuns são, por exemplo:

- Django-ORM (parte do framework Django)
- SQLAlchemy ORM (parte do SQLAlchemy, independente do framework)
- Peewee (independente do framework).

Aqui veremos como trabalhar com `SQLAlchemy ORM`.

Criação Conexão ao Banco de Dados

Vamos nos referir ao arquivo `sql_app/database.py`.

Importar as partes do SQLAlchemy:

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
```

Criação Conexão ao Banco de Dados

Criar uma URL de banco de dados para SQLAlchemy:

```
SQLALCHEMY_DATABASE_URL = "sqlite:///./sql_app.db"
# SQLALCHEMY_DATABASE_URL = "postgresql://user:password@postgresserver/db"
```

Neste exemplo, estamos "conectando" a um banco de dados `SQLite` (abrindo um arquivo com o banco de dados SQLite).

O arquivo está localizado no mesmo diretório do arquivo `sql_app.db`.

É por isso que a última parte é `./sql_app.db`

Criação Conexão ao Banco de Dados

O primeiro passo é criar um "motor" ou ”engine” SQLAlchemy.

Mais tarde, usaremos essa “engine” em outros lugares do projeto.

```
engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
```

Criação Conexão ao Banco de Dados

Cada instância da classe `SessionLocal` será uma sessão de banco de dados. A classe em si ainda não é uma sessão de banco de dados.

Mas uma vez que criamos uma instância da classe `SessionLocal`, essa instância será a sessão real do banco de dados.

```
engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

Criação Conexão ao Banco de Dados

Agora vamos usar a função `declarative_base()` que retorna uma classe.

Mais tarde herdaremos desta classe para criar cada um dos modelos ou classes de banco de dados (os modelos ORM):

```
engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()
```

Criação de modelos de banco de dados

Vamos agora ver o arquivo `sql_app/models.py`.

Usaremos essa classe Base que criamos antes para criar os modelos SQLAlchemy.

Importar `Base` de database(o arquivo `database.py` acima).

```
from sqlalchemy import Boolean, Column, ForeignKey, Integer, String
from sqlalchemy.orm import relationship

from .database import Base
```

Essas classes são os modelos SQLAlchemy.

O atributo `__tablename__` informa ao SQLAlchemy o nome da tabela a ser usada no banco de dados para cada um desses modelos.

```
class Item(Base):
    __tablename__ = "items"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
    description = Column(String, index=True)
    owner_id = Column(Integer, ForeignKey("users.id"))

    owner = relationship("User", back_populates="items")
```

ATIVIDADE

eeeeeh!

Atividades

Link: https://github.com/CakeERP/cakeerp-talent-program-2022

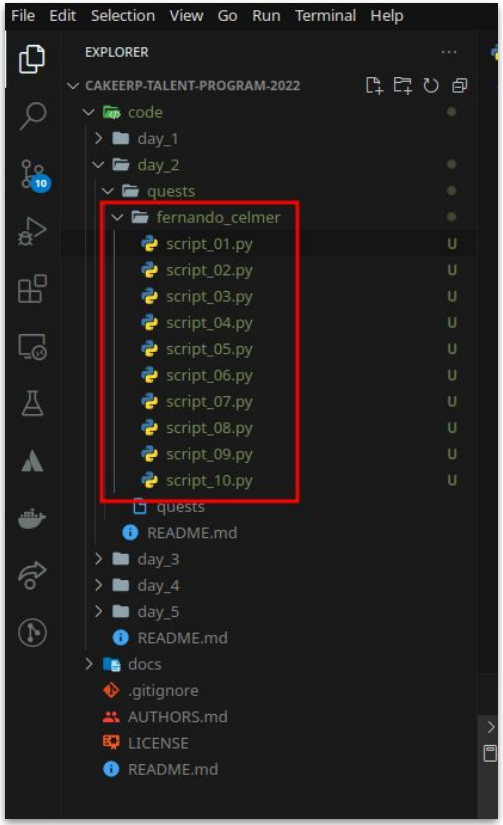
Onde fazer as atividades?

As atividades devem ser desenvolvidas dentro do repositório cakeerp-talent-program-2022 no seguinte local. `code > day_3 > quests > <seu_nome>.`

Como enviar as atividades?

Para envio das atividades deve-se realizar todo fluxo referente ao Github `(add/commit/push/pull request)`.

OBS: Não commitar o ambiente virtual!



Atividades

O que é preciso fazer?

1. Criar um arquivo main.py que contenha um endpoint de "/status" que retorna o seguinte JSON: {"status": "it's alive"}.
2. Criar um arquivo chamado database.py que realize a conexão a um banco de dados SQLite.
3. Criar um arquivo chamado models.py onde contenha um modelo de estrutura de uma tabela para o banco de dados com nome de "users" com as seguintes colunas:
    a. id: int
    b. name: string
    c. email: string
    d. description: string

LINKS

- FastAPI Documentation
- FastAPI Source Code
- O que é uma API?