**Laís:** Separamos nossos arquivos por restaurante, ou seja, cada restaurante é como se tivesse um cardápio próprio. Acessamos o do *Burger King* e encontramos todos os itens, preço e descrição de cada produto do Burger King em específico.

No entanto, se voltarmos à nossa API, temos uma URL que tem as informações de API, restaurante e JSON. Idealmente, era isso que usaríamos, colocaríamos em algum lugar para acessar. Por exemplo, seria api-restaurantes/burgerking. Acessaríamos o cardápio diretamente dessa URL.

Há várias maneiras de implementar isso utilizando o *Python* e vários *frameworks* do Python. No caso, temos, por exemplo, o *Django*, o *Flask* e o *FastAPI*. Por escolha de projeto, escolhemos o **FastAPI** para começar a criar uma rota e colocar essas informações que criamos em uma URL.

**Usando o *FastAPI***

**Guilherme:** A ideia é, daqui para frente, já começar a integrar esse ambiente de frameworks em Python específicos para *back-end*, para nós criarmos.

A primeira coisa que vamos precisar fazer, já que escolhemos o FastAPI, é criar um *endpoint* para cada um dos restaurantes, um para o *McDonald's*, um para o Burger King, um para o restaurante da Pizza Suprema, o outro da cantina, e assim por diante.

Porém, existe uma forma de criar uma API e deixá-la recebendo parâmetros especificamente. Mas antes de partirmos para esse cenário um pouco mais avançado, vamos criar apenas uma rota, um endpoint de API que fala, por exemplo, o prático "Olá, Mundo".

Primeiramente, vamos precisar instalar o FastAPI. Antes de instalar de fato o FastAPI com o pip, podemos analisar a documentação dele.

**Laís:** Vamos abrir o navegador e buscar por "FastAPI". Geralmente, é o primeiro link que aparece e o tutorial está logo no link "[*Tutorial - User Guide*](https://fastapi.tiangolo.com/tutorial/)".

**Guilherme:** Nessa página, a documentação fala um pouco sobre instalação, entre outras coisas, como a forma que subimos o servidor. A instalação é simples: basta usar o comando pip install fastapi.

**Laís:** No terminal, vamos colar esse comando.

pip install fastapi

Copiar código

Demora um pouco para instalar, o mesmo esquema que já acompanhamos antes. Quando terminar de instalar, ele notifica que deu tudo certo com a instalação.

**Guilherme:** O que fizemos? Pegamos um monte de código Python específico para conseguirmos criar o nosso próprio servidor, a nossa própria API. É uma solução que outro time de desenvolvimento já criou para conseguirmos utilizar.

**Laís:** Na documentação, também é mencionada a necessidade de instalar o uvicorn. Então, precisamos voltar no terminal e junto ao fastapi, instalar o uvicorn usando o comando abaixo:

pip install uvicorn

Copiar código

**Guilherme:** Já que instalamos dependências novas, faz sentido usar o comando pip freeze > requirements.txt para listar todas as dependências que precisamos.

pip freeze > requirements.txt

Copiar código

Agora vamos criar o nosso primeiro endpoint.

**Laís:** Criaremos um arquivo main.py para diferenciar do app.py que tínhamos criado. Geralmente, colocamos o app.py como padrão, mas até na própria documentação, ele menciona a criação de um arquivo main.py. Portanto, usaremos o main.py como padrão.

Como podemos começar a primeira rota?

**Guilherme:** A primeira coisa que vamos precisar fazer no arquivo main.py será importar o FastAPI. Então, podemos digitar a linha de código abaixo:

*main.py:*

**from** fastapi **import** FastAPI

Copiar código

Em seguida, vamos precisar de um objeto do FastAPI. Então, podemos digitar app = FastAPI().

**from** fastapi **import** FastAPI

app = FastAPI()

Copiar código

Da mesma forma que temos os decorators para falar se é uma property ou se é um class method, existem decorators específicos do FastAPI. Podemos criar um indicando que queremos disponibilizar um recurso através do get(). Então, pode ser @app.get().

Vamos colocar entre parênteses o caminho que queremos que esse recurso acesse. Colocaremos "/api/hello" entre aspas simples, só para exibir uma mensagem. Lembrando, não queremos uma página, queremos conteúdo, o recurso específico.

**from** fastapi **import** FastAPI

app = FastAPI()

@app.get('/api/hello')

Copiar código

Abaixo, podemos começar a criar uma função chamada def hello\_world(). Esperamos que essa função retorne para nós (return), entre chaves e entre aspas simples, o texto "*Hello World*".

**from** fastapi **import** FastAPI

app = FastAPI()

@app.get('/api/hello')

**def** **hello\_world**():

**return** {'Hello':'World'}

Copiar código

**Laís:** Esse endpoint não vai funcionar para qualquer pessoa que enviarmos, porque o servidor está na nossa máquina, no nosso computador. Então, vamos dizer que ele rodará localmente e não na internet.

Para esse projeto para rodar na internet, colocaríamos todo esse código que temos em um provedor de *cloud*, em um servidor de cloud também.

**Guilherme:** Para começar a rodar na nossa máquina, existe um comando que podemos usar para subir esse servidor, que é o seguinte:

uvicorn main:app --reload

Copiar código

**Laís:** Após executar no terminal, ele informa que está rodando no endpoint "[http://127.0.0.1:8000](http://127.0.0.1:8000/)", que chamamos de localhost.

**Guilherme:** Podemos pressionar "Ctrl" e clicar nesse link. Ele irá abrir e será exibida uma mensagem falando que não foi encontrado. Por quê?

**Laís:** Porque essa rota que criamos não existe. Criamos uma rota específica, um endpoint específico. No caso, teríamos que colocar api/hello no endereço da barra de navegação, que era o que tínhamos criado. Assim, aparece a mensagem "{"Hello":"World"}".

**Conclusão**

**Guilherme:** Entendemos como conseguimos criar um endpoint e indicar que, caso tenhamos uma requisição para determinado endereço, para determinada rota, queremos exibir um recurso específico. No caso, exibimos o "Hello World".

Nosso próximo desafio agora é como implementamos para exibir as informações dos cardápios com base nos restaurantes.