# Build Book - Webhook Demo

> **Projeto:** TechMahindra - Taboca
>
> **Preparado por:** IBM Build Labs
>
> **Entrega:** 25/10/2022

# Conteúdo



### Requisitos Mínimos:

Para executar esta aplicação você deve ter as seguintes dependências instaladas:

- Python
- Docker
- Banco de dados PostgreSQL

**Nota:** Certifique-se de que o pip também esteja instalado e disponível em seu PATH.

# 1. Escopo

O escopo deste aplicativo é servir como exemplo de como criar uma API Rest capaz de ser chamada pelo Watson Assistant.

# 2. Estrutura dos diretórios

```
├── utils
|       ├── get_reminders.py - Script para obter todos os e-mails do banco de
dados.
|      ├── insert_managers_reminders_table.py - Script para inserir gerentes no
DB.
|      └── pg_scripts - Scripts SQL para manipulação de banco de dados.
|            ├── create_table.sql - cria todas as tabelas necessárias.
|            ├── get_managers.sql - Obtém todos os gerentes registrados
|                  ├── insert_into_lembretes.sql - Insere dados na tabela de
lembretes
|            └── update_reminder.sql - Atualiza a tabela de lembretes
|
├── Dockerfile - Arquivo para construir uma imagem de contêiner
├── README.md
├── app.py - Aplicativo principal.
└── requirements.txt
```

# 3. Rotas Disponíveis

## 3.1 Rotas - `[GET]`

### 1. `/`

Esta rota retorna um "Hello World!" Esta é uma rota de teste.

### 2. `/api/v2/remember_managers`

Esta é uma rota utilizada pelo Cloud Functions, esta rota envia um e-mail para todas as pessoas na lista de lembretes.

Apenas as pessoas que foram lembradas menos de 3 vezes serão lembradas. caso contrário, eles serão excluídos do bd.

Resposta esperada:

`X E-mails enviados`

## 3.1 Rotas `[POST]`

### 1. `/api/v2/create_email`

Esta rota é utilizada para envio de e-mails com todos os dados necessários para um analista humano, para criação de um e-mail.

corpo de solicitação esperado:

```
{
"request": {
               "employee": {
                   "name": "John Doe",
                   "cpf":"123.987.123-45"
                 },
               "requester": {
                   "name": "John Doe",
                   "empresa":"taboca",
                   "cargo":"gerente",
                   "centro_de_custo":"Financeiro",
                   "matricula":"12345",
                   "email":"john_doe@test.com",
                   "localidade":"Alphaville"
                 }
               },
               "subject":"Help!"
               }
}
```

Resposta esperada:

Satus Code: `200- OK`

`Enviado`

# 4. Configurações do ambiente de teste

Caso não tenha um banco de dados PostgreSQL , crie um aqui

## 4.2 Configurações Variáveis de Ambiente

Para a aplicação funcionar conforme o esperado, é necessário criar o arquivo `.env`, utilize o arquivo `.env.example` como referência.

```
MAIL_USERNAME = seu_email
MAIL_PASSWORD = senha_email
PG_USERNAME = usuario_pg
PG_PASSWORD = senha_pg
PG_HOST = host_pg
PG_PORT = porta_pg
PG_DB = pg_db
```

`MAIL_USERNAME` - É o e-mail que disparará mensagens automáticas aos usuários. `MAIL_PASSWORD` - É a senha do e-mail. Para este caso, foi usado o gmail. Por questões de segurança o gmail não permite utilizar a sua senha padrão, é necessário criar uma senha de aplicativo. Caso tenha interesse em utilizar o Gmail, segue o link

`PG_USERNAME`- Usuário do Banco de Dados PostgreSQL `PG_PASSWORD` - Senha do usuário do Banco de Dados PostgreSQL `PG_HOST` - Hostname do banco de dados PostgreSQL `PG_PORT` - Porta utilizada pelo banco de dados PostgreSQL `PG_DB` - Nome do banco de dados PostgreSQL

## 4.1 Rodando a aplicação localmente

## 4.2. Instalando as dependências necessárias

Para instalar todas as dependências necessárias, rode

```
pip install -r ./requirements.txt
```

Você pode executar o aplicativo localmente executando `flask run` na pasta `./rest_api`.

Você deve ver a seguinte saída no console:

```
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

## 4.2 Construindo um container Docker localmente

### 4.2.1 Como gerar um arquivo requirements.txt

O requirements.txt é um arquivo que possui todas as dependências necessárias para rodar nossa aplicação, este arquivo pode ser gerado automaticamente com o seguinte comando:

```
pipreqs .
```

Se a execução foi bem sucedida, você deverá ver um novo arquivo chamado `requirements.txt`

### 4.2.3 Como construir uma imagem Docker

Para construir uma imagem docker, rode o seguinte comando

```
docker build -t <nome_da_sua_aplicação> .
```

exemplo:

```
docker build -t rest_api .
```

### 4.2.4 Executando sua imagem do Docker como um contêiner

Depois de construir sua imagem, você pode executá-la com o seguinte comando

```
docker run -d -p <porta_do_seu_pc>:<porta_interna_do_container> <nome_da_sua_aplicação>
```

exemplo:

```
docker run -d -p 5001:5000 rest_api
```

O comando acima tem dois argumentos:

- -d : Isso diz ao docker para ser executado no modo desanexado, ele não bloqueia seu terminal durante a execução do aplicativo.
- -p: diz ao docker para vincular a porta do seu computador à porta dos contêineres.

Após executar o comando acima, a aplicação deve estar acessível em `http://localhost:5001/`

### 4.2.5 Parando a aplicação

Depois de executar seu aplicativo, liste os contêineres em execução com:

```
docker ps
```

A saída esperada dever ser:

```
CONTAINER ID    IMAGE       COMMAND                     CREATED          STATUS
PORTS                   NAMES
1fe244221799    rest_api    "python3 -m flask ru…"    6 seconds ago    Up 5
seconds   0.0.0.0:5
```

Então, você pode parar a sua aplicação somente com os primeiros caracteres, conforme o exemplo abaixo.

```
docker kill 1fe
```

# 5. Referências

1. Docker
2. PostgreSQL