Crie uma aplicação Web com FastHTML e Python Puro (Sem JavaScript!)

<https://www.youtube.com/watch?v=_o31SB3NLFk>

⭐ **TIMESTAMPS:**

* 00:00 – Introdução
* 00:48 – Instalando o FastHTML
* 01:04 – Executando o código de exemplo localmente
* 01:38 – Entendendo o FastHTML com código de exemplo
* 03:45 – Estilizando a aplicação com Pico CSS
* 04:32 – Criando uma aplicação multi-página
* 05:08 – Construindo o formulário do livro de visitas
* 10:13 – Conectando a um banco de dados Postgres com Supabase
* 12:30 – Buscando dados do banco de dados
* 14:07 – Inserindo novos dados no banco de dados
* 15:58 – Manipulando envios de formulários com FastHTML
* 17:35 – Adicionando um Favicon
* 17:55 – Implementando a aplicação web no Vercel
* 20:28 – Testando a aplicação ao vivo
* 20:39 – Encerramento

Introdução (0:00)

Neste tutorial, vamos construir uma aplicação web do zero usando o FastHTML, um novo pacote Python. A aplicação será um livro de visitas digital onde os usuários podem deixar mensagens. As mensagens serão armazenadas em um banco de dados e, acredite, tudo isso será feito em Python puro, sem uma linha de JavaScript!

Conhecimento básico de HTML será útil, mas não se preocupe, vamos cobrir tudo que você precisa saber para começar com o FastHTML e construir seus próprios projetos. No final, vamos até mesmo implementar nossa aplicação web gratuitamente, para que você tenha seu próprio livro de visitas online.

Instalando o FastHTML (0:48)

1. Acesse a documentação oficial do FastHTML (link na descrição).
2. Copie o comando de instalação e execute-o no seu terminal ou prompt de comando:

pip install fasthtml

Executando o código de exemplo localmente (1:04)

1. Copie o código de exemplo da documentação do FastHTML:
2. from fasthtml import FastHTML, Request, Response
3. app = FastHTML()
4. @app.get("/")
5. def read\_root():
6. return """
7. <div>
8. <p hx-get="/change" hx-target="#content">Hello World!</p>
9. <div id="content"></div>
10. </div>
11. """
12. @app.get("/change")
13. def change():
14. return "<p>Nice to be here!</p>"
15. if \_\_name\_\_ == "\_\_main\_\_":
16. import uvicorn

uvicorn.run(app, host="0.0.0.0", port=5001)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

1. Cole o código em um arquivo Python (ex: main.py).
2. Inicie o servidor web:

python main.py

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. Acesse a aplicação em http://localhost:5001/ no seu navegador.

Entendendo o FastHTML com código de exemplo (1:38)

O FastHTML combina o poder do FastAPI e do HTMX:

* **FastAPI:** Serve como o servidor web, a espinha dorsal da nossa aplicação.
* **HTMX:** Uma biblioteca JavaScript que estende a funcionalidade do HTML, tornando-o mais interativo sem escrever JavaScript diretamente.

Analisando o código de exemplo:

* **Importações:** Importamos as classes FastHTML, Request e Response do FastHTML.
* **Inicialização:** Criamos uma instância do FastHTML (app) e definimos duas rotas: / (raiz) e /change.
* **Rota raiz (/):** Retorna um HTML simples com um parágrafo (<p>) contendo "Hello World!". O atributo hx-get do HTMX está configurado para enviar uma requisição GET para a rota /change quando o parágrafo é clicado. O atributo hx-target especifica que o conteúdo retornado da rota /change deve substituir o elemento com o ID "content".
* **Rota /change:** Retorna um HTML com um parágrafo contendo "Nice to be here!".
* **Execução do servidor:** O código usa o Uvicorn para iniciar o servidor web.

Estilizando a aplicação com Pico CSS (3:45)

O FastHTML já inclui estilos básicos usando o Pico CSS:

* **Elemento title:** Envolve o conteúdo em um contêiner e aplica automaticamente o Pico CSS.

Você pode explorar outras bibliotecas CSS como Tailwind CSS, mas neste tutorial, vamos usar o Pico CSS.

Criando uma aplicação multi-página (4:32)

Já temos uma aplicação multi-página! Usamos o HTMX para buscar o conteúdo da segunda página dinamicamente. Para criar uma navegação tradicional com links, basta usar um elemento <a> regular:

<a href="/change">Ir para a página 2</a>

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Html

Construindo o formulário do livro de visitas (5:08)

1. **Criando o formulário:**
2. from fasthtml import FastHTML
3. app = FastHTML()
4. MAX\_NAME\_LENGTH = 50
5. MAX\_MESSAGE\_LENGTH = 200
6. def render\_message(message):
7. return f"""
8. <article class="card">
9. <header class="card-header">
10. <p class="card-header-title">
11. {message["name"]}
12. </p>
13. </header>
14. <div class="card-content">
15. <div class="content">
16. {message["message"]}
17. </div>
18. </div>
19. <footer class="card-footer">
20. <p class="card-footer-item">
21. <small><em class="has-text-grey">{message["timestamp"]}</em></small>
22. </p>
23. </footer>
24. </article>
25. """
26. def render\_message\_list(messages):
27. return f"""
28. <div>
29. {' '.join([render\_message(message) for message in messages])}
30. </div>
31. """
32. @app.get("/")
33. def read\_root():
34. return f"""
35. <title>Meu Livro de Visitas ✍️</title>
36. <div class="container">
37. <h1>Deixe uma mensagem!</h1>
38. <form method="POST" hx-post="/submit\_message" hx-target="#messageList" hx-swap="innerHTML" hx-reset="this">
39. <fieldset role="group">
40. <label for="name">Nome:</label>
41. <input type="text" id="name" name="name" maxlength="{MAX\_NAME\_LENGTH}" required>
42. </fieldset>
43. <fieldset role="group">
44. <label for="message">Mensagem:</label>
45. <textarea id="message" name="message" maxlength="{MAX\_MESSAGE\_LENGTH}" required></textarea>
46. </fieldset>
47. <button type="submit">Enviar</button>
48. </form>
49. <hr>
50. <div id="messageList">
51. {render\_message\_list(get\_messages())}
52. </div>
53. </div>
54. """
55. def get\_messages():
56. # Implementação da busca de mensagens do banco de dados virá aqui
57. return []
58. if \_\_name\_\_ == "\_\_main\_\_":
59. import uvicorn

uvicorn.run(app, host="0.0.0.0", port=5001)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* + Usamos elementos HTML como <form>, <input>, <textarea> e <button> para criar o formulário.
  + O atributo hx-post no formulário envia uma requisição POST para a rota /submit\_message ao ser enviado.
  + hx-target="#messageList" atualiza o conteúdo da div com o ID messageList com a resposta da requisição.
  + hx-swap="innerHTML" substitui o conteúdo HTML interno do alvo.
  + hx-reset="this" limpa o formulário após o envio.
  + MAX\_NAME\_LENGTH e MAX\_MESSAGE\_LENGTH definem o comprimento máximo dos campos de nome e mensagem, respectivamente.

1. **Criando as funções auxiliares render\_message e render\_message\_list:**
   * render\_message(message): recebe um dicionário message contendo as informações da mensagem (nome, mensagem e timestamp) e retorna o HTML formatado para exibir a mensagem em um cartão.
   * render\_message\_list(messages): recebe uma lista de dicionários messages e retorna o HTML formatado para exibir todas as mensagens.
2. **Buscando mensagens do banco de dados:**
   * A função get\_messages será implementada posteriormente para buscar as mensagens do banco de dados. Por enquanto, ela retorna uma lista vazia.

Conectando a um banco de dados Postgres com Supabase (10:13)

1. **Criando uma conta no Supabase:**
   * Acesse o site do Supabase (link na descrição) e crie uma conta gratuita.
   * Crie uma nova organização e um novo projeto no Supabase.
2. **Criando a tabela messages:**
   * Acesse o editor de tabelas no seu projeto do Supabase.
   * Crie uma nova tabela chamada messages com as seguintes colunas:
     + id: inteiro, chave primária, autoincremento
     + name: texto
     + message: texto
     + timestamp: texto
3. **Configurando as variáveis de ambiente:**
   * Crie um arquivo .env na raiz do seu projeto.
   * Copie a chave da sua API do Supabase e cole no arquivo .env:

SUPABASE\_KEY=<SUA\_CHAVE\_SUPABASE>

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

* + Copie a URL do seu projeto do Supabase e cole no arquivo .env:

SUPABASE\_URL=<SUA\_URL\_SUPABASE>

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

1. **Instalando as dependências:**

pip install supabase python-dotenv pytz

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Conectando ao banco de dados:**
2. import os
3. from supabase import create\_client, Client
4. from dotenv import load\_dotenv
5. # Carrega as variáveis de ambiente do arquivo .env
6. load\_dotenv()
7. # Obtém as variáveis de ambiente
8. supabase\_url: str = os.environ.get("SUPABASE\_URL")
9. supabase\_key: str = os.environ.get("SUPABASE\_KEY")
10. # Inicializa o cliente Supabase

supabase: Client = create\_client(supabase\_url, supabase\_key)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

Buscando dados do banco de dados (12:30)

1. **Implementando a função get\_messages:**
2. def get\_messages():
3. # Busca todas as mensagens da tabela 'messages' ordenadas por 'id'
4. response = supabase.table("messages").select("\*").order("id").execute()
6. # Retorna a lista de mensagens

return response.data

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* + Usamos o método table() para acessar a tabela messages.
  + O método select("\*") busca todas as colunas.
  + order("id") ordena os resultados por id em ordem crescente.
  + execute() executa a consulta.
  + response.data contém a lista de mensagens retornadas pela consulta.

Inserindo novos dados no banco de dados (14:07)

1. **Criando a função auxiliar generate\_timestamp:**
2. import datetime
3. import pytz
4. # Define o formato do timestamp
5. TIMESTAMP\_FORMAT = "%Y-%m-%d %H:%M:%S"
6. def generate\_timestamp():
7. # Obtém o timezone de Berlin
8. berlin\_tz = pytz.timezone("Europe/Berlin")
10. # Obtém o datetime atual no timezone de Berlin
11. current\_datetime\_berlin = datetime.datetime.now(berlin\_tz)
13. # Formata o datetime como string

return current\_datetime\_berlin.strftime(TIMESTAMP\_FORMAT)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* + generate\_timestamp retorna o timestamp formatado no fuso horário de Berlim. Você pode ajustar o fuso horário conforme necessário.

1. **Criando a função add\_message:**
2. def add\_message(name: str, message: str):
3. # Gera um timestamp
4. timestamp = generate\_timestamp()
6. # Insere a nova mensagem na tabela 'messages'
7. supabase.table("messages").insert({
8. "name": name,
9. "message": message,
10. "timestamp": timestamp

}).execute()

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* + add\_message insere uma nova mensagem na tabela messages com o nome, mensagem e timestamp fornecidos.

1. **Criando a rota /submit\_message:**
2. @app.post("/submit\_message")
3. async def submit\_message(request: Request):
4. # Obtém os dados do formulário
5. form\_data = await request.form()
6. name = form\_data.get("name")
7. message = form\_data.get("message")
8. # Adiciona a mensagem ao banco de dados
9. add\_message(name, message)
10. # Retorna a lista de mensagens atualizada

return render\_message\_list(get\_messages())

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* + submit\_message é chamada quando o formulário é enviado.
  + request.form() obtém os dados do formulário enviado.
  + add\_message adiciona a nova mensagem ao banco de dados.
  + render\_message\_list(get\_messages()) retorna a lista de mensagens atualizada para atualizar a página.

Adicionando um Favicon (17:35)

1. Crie uma pasta assets na raiz do seu projeto.
2. Adicione um arquivo de imagem favicon à pasta assets (ex: favicon.ico).
3. Adicione o seguinte código HTML na tag <head> do seu template:

<link rel="icon" href="/assets/favicon.ico" type="image/x-icon">

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Html

Implementando a aplicação web no Vercel (17:55)

1. **Criando um repositório Git:**
   * Inicialize um repositório Git na raiz do seu projeto:

git init

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Bash

* + Crie um arquivo .gitignore para evitar o commit de arquivos sensíveis (como .env):
  + .env
  + \_\_pycache\_\_/

\*.pyc

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

* + Adicione os arquivos ao stage e faça o commit inicial:
  + git add .

git commit -m "Commit inicial"

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Criando um arquivo requirements.txt:**
   * Gere um arquivo requirements.txt com as dependências do seu projeto:

pip freeze > requirements.txt

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Enviando o código para o GitHub:**
   * Crie um novo repositório no GitHub.
   * Adicione o repositório remoto do GitHub ao seu repositório local.
   * Envie o código para o GitHub:

git push origin master

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Implementando no Vercel:**
   * Crie uma conta no Vercel (link na descrição).
   * Importe o repositório do GitHub para o Vercel.
   * Configure as variáveis de ambiente (SUPABASE\_KEY e SUPABASE\_URL) no painel do Vercel.
   * Implemente a aplicação.

Testando a aplicação ao vivo (20:28)

* Acesse o endereço da sua aplicação implementada no Vercel.
* Teste o formulário de livro de visitas e verifique se as mensagens estão sendo salvas e exibidas corretamente.

Encerramento (20:39)

Parabéns! Você construiu uma aplicação web dinâmica com FastHTML e Python puro, sem escrever nenhuma linha de JavaScript. Explore os recursos do FastHTML e do HTMX para criar aplicações web ainda mais poderosas e interativas.

User

arrow\_upwardarrow\_downwardedit

more\_vert

faça o codigo completo com todos os arquivos e dependencia e me explique o passa a passo para fazer o deploy

Model

33,4s

arrow\_upwardarrow\_downwardedit

more\_vert

Código Completo e Tutorial de Deploy para o Livro de Visitas FastHTML

Aqui está o código completo da aplicação web de livro de visitas, junto com as instruções passo a passo para implementar no Vercel.

Estrutura de Arquivos

livro-de-visitas/

├── assets/

│ └── favicon.ico

├── .env

├── .gitignore

├── main.py

└── requirements.txt

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

main.py

import os

import datetime

import pytz

from fasthtml import FastHTML, Request

from supabase import create\_client, Client

from dotenv import load\_dotenv

# Carrega as variáveis de ambiente do arquivo .env

load\_dotenv()

# Obtém as variáveis de ambiente

supabase\_url: str = os.environ.get("SUPABASE\_URL")

supabase\_key: str = os.environ.get("SUPABASE\_KEY")

# Inicializa o cliente Supabase

supabase: Client = create\_client(supabase\_url, supabase\_key)

app = FastHTML()

MAX\_NAME\_LENGTH = 50

MAX\_MESSAGE\_LENGTH = 200

# Define o formato do timestamp

TIMESTAMP\_FORMAT = "%Y-%m-%d %H:%M:%S"

def generate\_timestamp():

# Obtém o timezone de Berlin

berlin\_tz = pytz.timezone("Europe/Berlin")

# Obtém o datetime atual no timezone de Berlin

current\_datetime\_berlin = datetime.datetime.now(berlin\_tz)

# Formata o datetime como string

return current\_datetime\_berlin.strftime(TIMESTAMP\_FORMAT)

def add\_message(name: str, message: str):

# Gera um timestamp

timestamp = generate\_timestamp()

# Insere a nova mensagem na tabela 'messages'

supabase.table("messages").insert({

"name": name,

"message": message,

"timestamp": timestamp

}).execute()

def get\_messages():

# Busca todas as mensagens da tabela 'messages' ordenadas por 'id'

response = supabase.table("messages").select("\*").order("id").execute()

# Retorna a lista de mensagens

return response.data

def render\_message(message):

return f"""

<article class="card">

<header class="card-header">

<p class="card-header-title">

{message["name"]}

</p>

</header>

<div class="card-content">

<div class="content">

{message["message"]}

</div>

</div>

<footer class="card-footer">

<p class="card-footer-item">

<small><em class="has-text-grey">{message["timestamp"]}</em></small>

</p>

</footer>

</article>

"""

def render\_message\_list(messages):

return f"""

<div>

{' '.join([render\_message(message) for message in messages])}

</div>

"""

@app.get("/")

def read\_root():

return f"""

<title>Meu Livro de Visitas ✍️</title>

<div class="container">

<h1>Deixe uma mensagem!</h1>

<form method="POST" hx-post="/submit\_message" hx-target="#messageList" hx-swap="innerHTML" hx-reset="this">

<fieldset role="group">

<label for="name">Nome:</label>

<input type="text" id="name" name="name" maxlength="{MAX\_NAME\_LENGTH}" required>

</fieldset>

<fieldset role="group">

<label for="message">Mensagem:</label>

<textarea id="message" name="message" maxlength="{MAX\_MESSAGE\_LENGTH}" required></textarea>

</fieldset>

<button type="submit">Enviar</button>

</form>

<hr>

<div id="messageList">

{render\_message\_list(get\_messages())}

</div>

</div>

"""

@app.post("/submit\_message")

async def submit\_message(request: Request):

# Obtém os dados do formulário

form\_data = await request.form()

name = form\_data.get("name")

message = form\_data.get("message")

# Adiciona a mensagem ao banco de dados

add\_message(name, message)

# Retorna a lista de mensagens atualizada

return render\_message\_list(get\_messages())

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=5001)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

.env

SUPABASE\_URL=<SUA\_URL\_SUPABASE>

SUPABASE\_KEY=<SUA\_CHAVE\_SUPABASE>

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

* Substitua <SUA\_URL\_SUPABASE> e <SUA\_CHAVE\_SUPABASE> pelos valores reais da sua conta do Supabase.

.gitignore

.env

\_\_pycache\_\_/

\*.pyc

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

requirements.txt

fastapi

uvicorn

fasthtml

supabase

python-dotenv

pytz

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

Deploy no Vercel

Pré-requisitos

* Conta do GitHub
* Conta do Vercel
* Projeto do Supabase configurado

Passos

1. **Crie um repositório Git:**
   * Abra o terminal e navegue até a pasta do seu projeto.
   * Inicialize um repositório Git: git init
   * Adicione os arquivos ao stage: git add .
   * Faça o commit inicial: git commit -m "Commit inicial"
2. **Crie um repositório no GitHub:**
   * Acesse o GitHub e crie um novo repositório.
   * Siga as instruções para conectar seu repositório local ao repositório do GitHub.
   * Envie o código para o GitHub: git push origin master
3. **Importe o projeto para o Vercel:**
   * Acesse o Vercel e faça login na sua conta.
   * Clique em "Import Project" e selecione "Import Git Repository".
   * Selecione o repositório do GitHub que você acabou de criar.
4. **Configure as variáveis de ambiente:**
   * Na seção "Settings" do seu projeto no Vercel, vá para a aba "Environment Variables".
   * Crie duas variáveis de ambiente: SUPABASE\_URL e SUPABASE\_KEY.
   * Cole os valores correspondentes da sua conta do Supabase nos campos de valor.
5. **Implemente a aplicação:**
   * Clique em "Deploy" e aguarde o processo de implementação ser concluído.
   * Acesse a URL fornecida pelo Vercel para visualizar sua aplicação online!

Observações

* Certifique-se de ter criado a tabela messages no seu projeto do Supabase com as colunas corretas.
* Você pode personalizar a aparência da sua aplicação editando o HTML e o CSS.
* O FastHTML e o HTMX oferecem diversas outras funcionalidades para criar aplicações web ainda mais poderosas e interativas. Consulte a documentação para saber mais.

User

arrow\_upwardarrow\_downwardedit

more\_vert

porfavor explique cada funcionalidade por parte para que eu compreenda o codigo e possa executar e entender melhor, seja legal comigo

Model

33,9s

arrow\_upwardarrow\_downwardedit

more\_vert

Entendendo o código do Livro de Visitas FastHTML passo a passo 😄

Vamos desvendar esse código juntos, explicando cada parte para que você possa executá-lo e entendê-lo como um profissional! 💪

**Parte 1: Configurando o ambiente e importando bibliotecas**

import os

import datetime

import pytz

from fasthtml import FastHTML, Request

from supabase import create\_client, Client

from dotenv import load\_dotenv

# Carrega as variáveis de ambiente do arquivo .env

load\_dotenv()

# Obtém as variáveis de ambiente

supabase\_url: str = os.environ.get("SUPABASE\_URL")

supabase\_key: str = os.environ.get("SUPABASE\_KEY")

# Inicializa o cliente Supabase

supabase: Client = create\_client(supabase\_url, supabase\_key)

app = FastHTML()

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

1. **Importando o necessário:**
   * os: Permite interagir com o sistema operacional, como ler variáveis de ambiente.
   * datetime: Trabalhar com datas e horas, essencial para o timestamp das mensagens.
   * pytz: Lidar com fusos horários, garantindo que o timestamp seja exibido corretamente.
   * FastHTML: A estrela do show! Permite criar aplicações web com Python puro.
   * Request: Usado para acessar os dados enviados pelo usuário através de formulários.
   * create\_client, Client: Ferramentas para conectar e interagir com o banco de dados Supabase.
   * load\_dotenv: Carrega as variáveis de ambiente do arquivo .env, mantendo suas informações seguras.
2. **Conectando ao Supabase:**
   * load\_dotenv(): Carrega as variáveis de ambiente do arquivo .env.
   * supabase\_url, supabase\_key: Armazenam a URL e a chave da sua conta do Supabase, respectivamente, lidas do arquivo .env.
   * supabase: Cria um cliente Supabase usando a URL e a chave, estabelecendo a conexão com o banco de dados.
3. **Criando a aplicação FastHTML:**
   * app = FastHTML(): Cria uma instância da aplicação FastHTML, que será responsável por lidar com as requisições e respostas da web.

**Parte 2: Definindo constantes e funções auxiliares**

MAX\_NAME\_LENGTH = 50

MAX\_MESSAGE\_LENGTH = 200

# Define o formato do timestamp

TIMESTAMP\_FORMAT = "%Y-%m-%d %H:%M:%S"

def generate\_timestamp():

# ... (código da função)

def add\_message(name: str, message: str):

# ... (código da função)

def get\_messages():

# ... (código da função)

def render\_message(message):

# ... (código da função)

def render\_message\_list(messages):

# ... (código da função)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

1. **Constantes:**
   * MAX\_NAME\_LENGTH: Define o tamanho máximo do nome do usuário (50 caracteres).
   * MAX\_MESSAGE\_LENGTH: Define o tamanho máximo da mensagem (200 caracteres).
   * TIMESTAMP\_FORMAT: Define o formato de exibição do timestamp (Ano-Mês-Dia Hora:Minuto:Segundo).
2. **Funções auxiliares:**
   * generate\_timestamp(): Gera um timestamp formatado no fuso horário de Berlim.
   * add\_message(name, message): Adiciona uma nova mensagem ao banco de dados com o nome, mensagem e timestamp fornecidos.
   * get\_messages(): Busca todas as mensagens do banco de dados, ordenadas por ID.
   * render\_message(message): Recebe um dicionário com informações de uma mensagem e retorna o HTML formatado para exibi-la em um cartão.
   * render\_message\_list(messages): Recebe uma lista de mensagens e retorna o HTML formatado para exibir todas elas, usando a função render\_message para cada mensagem individual.

**Parte 3: Definindo as rotas da aplicação**

@app.get("/")

def read\_root():

# ... (código da função)

@app.post("/submit\_message")

async def submit\_message(request: Request):

# ... (código da função)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

1. **Rota principal (/):**
   * @app.get("/"): Decora a função read\_root() para que seja chamada quando o usuário acessar a raiz da aplicação (ex: https://seudominio.com/).
   * read\_root(): Retorna o HTML da página principal, que inclui:
     + Título da página.
     + Um formulário para o usuário inserir seu nome e mensagem.
     + Uma lista com as mensagens já cadastradas, obtidas com a função get\_messages().
2. **Rota de envio de mensagens (/submit\_message):**
   * @app.post("/submit\_message"): Decora a função submit\_message() para que seja chamada quando o usuário enviar o formulário (requisição POST).
   * async def submit\_message(request: Request): Define a função assíncrona submit\_message() que recebe a requisição (request) como argumento.
     + form\_data = await request.form(): Obtém os dados do formulário enviado pelo usuário.
     + name, message: Extrai o nome e a mensagem dos dados do formulário.
     + add\_message(name, message): Chama a função add\_message() para inserir a nova mensagem no banco de dados.
     + return render\_message\_list(get\_messages()): Retorna a lista de mensagens atualizada para ser exibida na página.

**Parte 4: Executando a aplicação**

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=5001)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* Este bloco de código é responsável por iniciar o servidor web Uvicorn e executar a aplicação FastHTML.
  + if \_\_name\_\_ == "\_\_main\_\_":: Garante que o código dentro deste bloco só seja executado quando o script for executado diretamente (não importado como um módulo).
  + import uvicorn: Importa a biblioteca Uvicorn.
  + uvicorn.run(app, host="0.0.0.0", port=5001): Inicia o servidor Uvicorn com as seguintes configurações:
    - app: A instância da aplicação FastHTML criada anteriormente.
    - host="0.0.0.0": Permite que a aplicação seja acessada de qualquer endereço IP (útil para desenvolvimento e deploy).
    - port=5001: Define a porta em que a aplicação estará disponível (você pode escolher outra porta, se desejar).

Pronto para começar!

Com este guia detalhado, você está pronto para executar, entender e até mesmo personalizar seu próprio livro de visitas online com FastHTML e Python. 🤘

Lembre-se de configurar o banco de dados Supabase, criar o arquivo .env com suas credenciais e instalar as bibliotecas necessárias antes de executar a aplicação.

Divirta-se programando! 🎉

User

arrow\_upwardarrow\_downwardedit

more\_vert

porfavor discuta com maior profundidade: Parte 3: Definindo as rotas da aplicação

Model

19,1s

warning

arrow\_upwardarrow\_downwardedit

more\_vert

Mergulhando fundo na Parte 3: Definindo as Rotas da Aplicação 🗺️

As rotas são como os caminhos dentro da sua aplicação web que determinam qual código será executado quando o usuário acessar uma URL específica. No FastHTML, definimos essas rotas usando **decoradores** e **funções**.

Vamos analisar cada rota do nosso Livro de Visitas:

1. Rota Principal (/) - Exibindo o Livro de Visitas

@app.get("/")

def read\_root():

# ... (código da função)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* **@app.get("/")**: Este é o **decorador** que conecta a função read\_root() à rota raiz (/).
  + @app.get(...): Indica que esta rota responde a requisições **GET**, o método HTTP usado para **obter dados** do servidor (como ao acessar uma página web).
  + "/": Define a **rota raiz**, ou seja, a página inicial da aplicação. Quando o usuário acessar o endereço base (ex: https://seudominio.com/), esta rota será acionada.
* **def read\_root():**: Esta é a **função** responsável por gerar o conteúdo da página principal.
  + Ela retorna o **HTML** que será renderizado no navegador do usuário, incluindo o título da página, o formulário para adicionar mensagens e a lista de mensagens existentes.

Em resumo, a rota principal garante que, ao acessar a página inicial, o usuário veja o formulário para deixar sua mensagem e a lista de mensagens já publicadas. 🪄

2. Rota de Envio de Mensagens (/submit\_message) - Processando Novas Mensagens

@app.post("/submit\_message")

async def submit\_message(request: Request):

# ... (código da função)

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Python

* **@app.post("/submit\_message")**: Este decorador conecta a função submit\_message() à rota /submit\_message.
  + @app.post(...): Indica que esta rota responde a requisições **POST**, o método HTTP usado para **enviar dados** ao servidor (como ao submeter um formulário).
  + "/submit\_message": Define a rota específica para lidar com o envio de novas mensagens.
* **async def submit\_message(request: Request):**: Define a função **assíncrona** submit\_message(), que recebe a requisição (request) como argumento. O uso de async permite que a função lide com operações assíncronas, como interagir com o banco de dados, de forma mais eficiente.
  + form\_data = await request.form(): Extrai os dados do formulário enviado pelo usuário. A palavra-chave await é usada aqui porque request.form() é uma operação assíncrona.
  + name, message: Obtém os valores dos campos "name" e "message" do formulário.
  + add\_message(name, message): Chama a função auxiliar add\_message() para inserir a nova mensagem no banco de dados.
  + return render\_message\_list(get\_messages()): Retorna a lista de mensagens atualizada para ser exibida na página, garantindo que o usuário veja sua mensagem publicada imediatamente após o envio.

Em resumo, a rota /submit\_message é responsável por processar os dados enviados pelo formulário, salvar a nova mensagem no banco de dados e atualizar a página com a lista completa de mensagens. 📨

Entendendo a Magia: A Interação entre as Rotas ✨

A beleza da estrutura de rotas do FastHTML está na sua simplicidade e organização. Cada rota tem uma função clara e definida, o que torna o código mais fácil de entender, manter e escalar.

No nosso Livro de Visitas, a rota principal exibe o formulário e a lista de mensagens, enquanto a rota /submit\_message lida exclusivamente com o processamento de novas mensagens. Essa separação de responsabilidades torna o código mais modular e facilita a implementação de novas funcionalidades no futuro. 🚀

Espero que esta análise detalhada das rotas tenha tornado o código mais claro para você! 😄 Continue explorando e experimentando com o FastHTML para construir suas próprias aplicações web incríveis! 🎉