



{JORNADA}
PYTHON

AULA 4

PYTHON DEV

```
def json_to_csv(json_file, csv_file):  
    with open(json_file) as f:  
        data = json.load(f)  
  
    # Create a CSV writer  
    csv_writer = csv.writer(csv_file)  
  
    # Write the header  
    csv_writer.writerow(data[0].keys())  
  
    # Write the data  
    csv_writer.writerows(data)
```



Parte 1

Introdução

{JORNADA}
PYTHON
100% GRATUITO E ONLINE

O que Vamos Aprender?

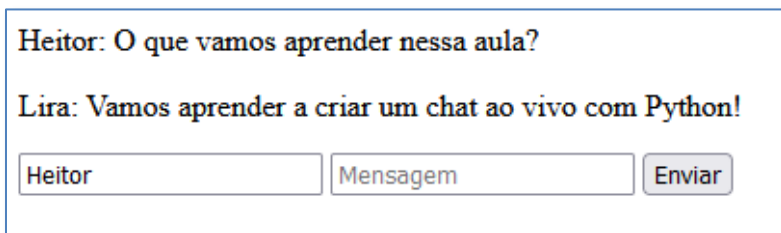
Na aula **Python Dev** nós vamos criar um **Chat ao Vivo**, que será o **HashZap**. Uma espécie de Whatsapp. Vai ser um chat ao vivo, onde as pessoas podem entrar e conversar como se fosse um chat normal.

Só que você não vai conseguir visualizar o histórico das conversas, pois a ideia é que seja de fato um chat ao vivo, então só quem está ali na hora vai visualizar as mensagens, caso saia e entre de novo vai visualizar as mensagens a partir do momento em que entrou no chat.

Para isso nós vamos criar um site e criar o chat dentro do site. Isso quer dizer que vamos ter bibliotecas novas, diferentes das que já utilizamos nas aulas anteriores, mas vamos te mostrar quais são e como instalar cada uma delas para que seu projeto funcione corretamente.

Dá uma olhada como vai ficar o nosso chat. Nós temos duas versões. Uma é a versão “crua”, pois como estamos trabalhando com um site, você deve saber que ele precisa de um visual.

Então nesse primeiro exemplo seria o site sem nenhum elemento visual. Já no segundo exemplo temos um visual totalmente diferente já com cara de chat mesmo!



Heitor: O que vamos aprender nessa aula?

Lira: Vamos aprender a criar um chat ao vivo com Python!

Heitor Mensagem



Hashzap

Welcome to Hashzap!

Heitor: O que vamos aprender nessa aula?

Lira: Vamos aprender a criar um chat ao vivo com Python!

Heitor Irado!

Parte 2

Bibliotecas Necessárias

Instalação das Bibliotecas

Para essa aula nós vamos utilizar algumas bibliotecas diferentes das que já utilizamos até agora, pois vamos fazer a criação de um site e do chat.

Então nós vamos precisar instalar as seguintes bibliotecas:

- **Flask** – pip install flask
- **Socketio** – pip install python-socketio / pip install flask-socketio
- **Simple Websocket** – pip install simple-websocket

Você precisa das bibliotecas instaladas para que consiga fazer esse projeto. Caso contrário vai ter algumas mensagens de erro ao executar o seu código. Então é importante que faça a instalação antes de iniciar o projeto.

Caso tenha alguma dúvida ou queira se aprofundar um pouco nas bibliotecas, vou deixar aqui a documentação de cada uma.

- **Flask** - <https://flask.palletsprojects.com/en/2.3.x/>
- **Flask no Visual Studio Code** - <https://code.visualstudio.com/docs/python/tutorial-flask>
- **Socketio** - <https://socket.io/pt-br/docs/v4/>

Na documentação você tem todas as informações necessárias para utilizar cada biblioteca e ainda tem todas as ferramentas que elas possuem e como usar cada uma delas.

A documentação é muito importante, pois te ajuda a utilizar as bibliotecas da forma correta, então se tiver algum problema pode consultar a biblioteca para verificar a forma correta de executar o código.

Parte 3

Código em Python

{JORNADA}
PYTHON
100% GRATUITO E ONLINE

Código em Python

```
1 from flask import Flask, render_template # estruturas para criar o site
2 from flask_socketio import SocketIO, send # estruturas para criar o chat
3
4 app = Flask(__name__) # cria o site
5 app.config["SECRET"] = "ajuiiahfa78fh9f78shfs768fgs7f6" # chave de segurança, pode ser qualquer coisa, mas escolha algo difícil
6 app.config["DEBUG"] = True # para testarmos o código, no final tiramos
7 socketio = SocketIO(app, cors_allowed_origins="*") # cria a conexão entre diferentes máquinas que estão no mesmo site
8
9 @socketio.on("message") # define que a função abaixo vai ser acionada quando o evento de "message" acontecer
10 def gerenciar_mensagens(mensagem):
11     print(f"Mensagem: {mensagem}")
12     send(mensagem, broadcast=True) # envia a mensagem para todo mundo conectado no site
13
14 @app.route("/") # cria a página do site
15 def home():
16     return render_template("index.html") # essa página vai carregar esse arquivo html que está aqui
17
18 if __name__ == "__main__":
19     socketio.run(app, host='localhost') # define que o app vai rodar no seu servidor local, ou seja, na internet em que o seu computador tá conectado
20
```

Esse é o código em Python que nós vamos utilizar para a criação do site e do chat. Vamos iniciar com a importação das bibliotecas, lembrando que você precisa instalar todas elas para que consiga construir o projeto.

Logo abaixo, vamos fazer a criação do site, vamos inserir uma chave de segurança, o debug para fazer o teste do código e por fim vamos fazer a conexão entre as máquinas que estão no site utilizando o chat.

Código em Python

```
1 from flask import Flask, render_template # estruturas para criar o site
2 from flask_socketio import SocketIO, send # estruturas para criar o chat
3
4 app = Flask(__name__) # cria o site
5 app.config["SECRET"] = "ajuiahfa78fh9f78shfs768fgs7f6" # chave de segurança, pode ser qualquer coisa, mas escolha algo difícil
6 app.config["DEBUG"] = True # para testarmos o código, no final tiramos
7 socketio = SocketIO(app, cors_allowed_origins="*") # cria a conexão entre diferentes máquinas que estão no mesmo site
8
9 @socketio.on("message") # define que a função abaixo vai ser acionada quando o evento de "message" acontecer
10 def gerenciar_mensagens(mensagem):
11     print(f"Mensagem: {mensagem}")
12     send(mensagem, broadcast=True) # envia a mensagem para todo mundo conectado no site
13
14 @app.route("/") # cria a página do site
15 def home():
16     return render_template("index.html") # essa página vai carregar esse arquivo html que está aqui
17
18 if __name__ == "__main__":
19     socketio.run(app, host='localhost') # define que o app vai rodar no seu servidor local, ou seja, na internet em que o seu computador tá conectado
20
```

Aqui nós vamos definir quando a função **gerenciar_mensagens** vai ser executada. Ela vai ser acionada, quando o evento **message** acontecer, que é o evento que temos dentro do nosso código html.

Nessa função nós vamos printar (exibir) a mensagem e fazer o envio da mensagem para todas as pessoas que estão conectadas no site. Então sempre que essa evento acontecer nós vamos executar essa função para o envio das mensagens.

Código em Python

```
1 from flask import Flask, render_template # estruturas para criar o site
2 from flask_socketio import SocketIO, send # estruturas para criar o chat
3
4 app = Flask(__name__) # cria o site
5 app.config["SECRET"] = "ajuiiahfa78fh9f78shfs768fgs7f6" # chave de segurança, pode ser qualquer coisa, mas escolha algo difícil
6 app.config["DEBUG"] = True # para testarmos o código, no final tiramos
7 socketio = SocketIO(app, cors_allowed_origins="*") # cria a conexão entre diferentes máquinas que estão no mesmo site
8
9 @socketio.on("message") # define que a função abaixo vai ser acionada quando o evento de "message" acontecer
10 def gerenciar_mensagens(mensagem):
11     print(f"Mensagem: {mensagem}")
12     send(mensagem, broadcast=True) # envia a mensagem para todo mundo conectado no site
13
14 @app.route("/") # cria a página do site
15 def home():
16     return render_template("index.html") # essa página vai carregar esse arquivo html que está aqui
17
18 if __name__ == "__main__":
19     socketio.run(app, host='localhost') # define que o app vai rodar no seu servidor local, ou seja, na internet em que o seu computador tá conectado
20
```

Nessa parte estamos de fato criando a página, o **Route** é o caminho do nosso “aplicativo”, então temos a página inicial onde vamos utilizar **2 templates** na aula.

Vamos utilizar o **homepage.html** e o **index.html**, que são o mesmo arquivo, mas o último já tem alguns códigos em HTML e CSS para melhorar o visual da página.

Por fim vamos definir que o app vai rodar no servidor local, ou seja, na internet em que o seu computador está conectado

Parte 4

Código em HTML

Código em HTML

Criando o Código Base

Como falei anteriormente, além do código em Python, nós vamos ter um código em HTML que é responsável pela nossa página.

Você pode pegar o arquivo que já está disponível para download, mas pode criar um arquivo **homepage.html** do zero sem problema algum.

Se tiver criado um arquivo do zero, não se preocupe, que você não vai precisar saber nada de **HTML**, vou te mostrar como você pode fazer para facilitar o processo e continuar com o projeto.

Quando criar o arquivo, ele vai estar vazio correto? Então você vai escrever **!** e pressionar a tecla **Tab** do teclado.

Isso vai fazer com o que o Visual Studio já escreva automaticamente um código base de HTML pra você.

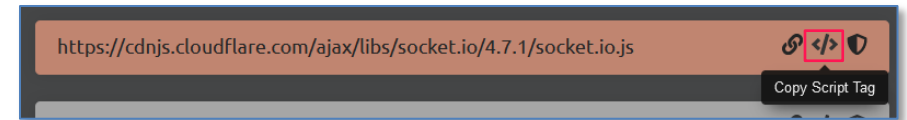
```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport"
6          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7      <meta http-equiv="X-UA-Compatible" content="ie=edge">
8      <title>Document</title>
9  </head>
10 <body>
11
12 </body>
13 </html>
```

Código em HTML

Adicionando os Scripts

Agora que temos o código base, nós vamos precisar acrescentar 2 scripts, que são do socketio e do jquery. Para obter esses scripts você pode acessar o site <https://cdnjs.com/> e procurar por **socketio** e **jquery** (pode clicar nos nomes também que será redirecionado para poder copiar os scripts).

Assim que clicar nas bibliotecas você vai clicar na opção **Copy Script Tag**.



Com os scripts copiados, basta colar os dois logo abaixo do título (**<title>Document</title>**). Não precisa se preocupar com as tags que estão em amarelo, pois no script tem o código completo.

Esses scripts servem para que as nossas bibliotecas funcionem dentro da página, por isso precisamos deles dentro do arquivo html. Lembrando que o código completo também está disponível para download.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7      <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.7.1/socket.io.js" integrity="sha512-Z6C1p1NIexPj5MsVUunW4pg7uMX6/TT3CUVldmjXx2kpi1"
8      <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.0/jquery.min.js" integrity="sha512-3gJwYpMe3QewGELv8k/BX9vcghryRdzRMxVfq6ngyWXwo036F
9  </head>
10 <body>
```

Adicionando Script das Ações

```
</head>
<body>

  <script type="text/javascript">
    $(document).ready(function() {
      var socket = io("localhost:5000")
      socket.on("connect", function() {
        console.log("conectou");
      });

      socket.on("message", function(data) {
        console.log("enviou mensagem");
        $("#lista_mensagens").append($('

').text(data));
      });

      $("#botao").on('click', function() {
        console.log("cliqueu botao");
        socket.send($('#usuario').val() + ": " + $('#mensagem').val());
        $('#mensagem').val('');
      });
    })
  </script>

  <div id="lista_mensagens">

  </div>
  <input type="text" id="usuario" placeholder="Usuario" required>
  <input type="text" id="mensagem" placeholder="Mensagem" required>
  <button id="botao">Enviar</button>

</body>
</html>


```

Vamos adicionar um outro script, só que agora dentro do `<body>`. Esse script vai definir o socket, que vamos alterar para **“localhost:5000”** para que você consiga utilizar na sua rede.

Logo abaixo vamos fazer a conexão e se estiver tudo certo você vai visualizar a mensagem **“conectou”** no console.

Para o envio de mensagens, nós vamos mostrar no console **“enviou mensagem”** e logo em seguida vamos adicionar a **lista_mensagens**, qual foi a mensagem enviada.

Como se fosse o histórico da conversa mesmo, mas só vai ficar esse histórico para quem estiver conectado, se sair vai perder as mensagens.

A outra ação é a de clicar no botão, então assim que você clica no botão de enviar, vamos mostrar isso no console e vamos adicionar o nome do usuário antes da mensagem.

Então você vai visualizar o nome do usuário e a mensagem que foi enviada para saber quem foi que mandou a mensagem, da mesma forma que um chat funciona.

Acrescentando os Elementos no Site

```
<div id="lista_mensagens">  
  
</div>  
<input type="text" id="usuario" placeholder="Usuario" required>  
<input type="text" id="mensagem" placeholder="Mensagem" required>  
<button id="botao">Enviar</button>  
  
</body>  
</html>
```

Vamos ter o espaço onde vão ficar armazenadas as mensagens, que é na **lista_mensagens**.

E por fim vamos ter as caixas onde o usuário vai poder informar seu nome e a mensagem, assim como o botão para poder enviar a mensagem.

Heitor: O que vamos aprender nessa aula?

Lira: Vamos aprender a criar um chat ao vivo com Python!

Com isso já finalizamos a configuração da página e do chat, agora você já vai poder rodar o código para ver como realmente o chat funciona.

OBS: Vale lembrar que temos o arquivo `index.html` também disponível para download, que é um arquivo com códigos em CSS para deixar o visual bem mais agradável.

Hashzap

Welcome to Hashzap!

Heitor: O que vamos aprender nessa aula?

Lira: Vamos aprender a criar um chat ao vivo com Python!

Heitor

Irado!

Enviar

Parte 5

Executando o Código

Executando o Código do Chat

Essa parte de execução do código é um pouco diferente do que fizemos nas últimas aulas. Aqui nós vamos executar o código através do terminal e não simples clicando em **run**.

Dentro do terminal você vai escrever **python -m flask run**, só que aqui temos um pequeno detalhe para ajustar.

Você vai ter que alterar o nome do seu arquivo de **main.py** para **app.py** para que consiga rodar direto sem ter que fazer alguns ajustes dentro do código.

Assim que rodar o código no terminal você vai visualizar algumas mensagens e uma delas é um link. Esse link é onde está a sua página com o chat, então basta clicar nele que o chat vai abrir no seu navegador.

```
PS C:\Users\Heitor\OneDrive\Hashtag Treinamentos\Jornada Python\Jornada Python - Aula 4> python -m flask run
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [21/Jul/2023 16:19:05] "GET /socket.io/?EI0=4&transport=polling&t=0bvv36M HTTP/1.1" 200 -
127.0.0.1 - - [21/Jul/2023 16:19:05] "POST /socket.io/?EI0=4&transport=polling&t=0bvv36W&sid=QhVqPfw2yggjSY36AAAA HTTP/1.1" 200 -
127.0.0.1 - - [21/Jul/2023 16:19:05] "GET /socket.io/?EI0=4&transport=polling&t=0bvv36W.0&sid=QhVqPfw2yggjSY36AAAA HTTP/1.1" 200 -
127.0.0.1 - - [21/Jul/2023 16:19:05] "GET /socket.io/?EI0=4&transport=polling&t=0bvv36k&sid=QhVqPfw2yggjSY36AAAA HTTP/1.1" 200 -
```

Agora basta inserir o seu usuário, a mensagem e clicar em enviar para mandar sua mensagem! Não sei se você reparou, mas no terminal apareceu uma mensagem (Press CTRL+C to quit) que é para pressionar **Ctrl + C** no terminal para parar a execução do código, caso contrário vai continuar executando até fechar o editor.

Executando o Código do Chat

Agora você pode abrir o seu chat e conversar com as pessoas que estão conectadas a mesma rede que você.

Pode inclusive, abrir mais de um chat no mesmo computador para verificar que está tudo funcionando normalmente.

Esse projeto foi um pouco mais complexo que os outros, mas ainda sim conseguiu um resultado incrível com pouco conhecimento.

Outro ponto importante que sempre vai te ajudar são as documentações, elas vão te ajudar tanto na instalação da biblioteca, pois nem todas tem uma instalação parecida ou um nome sugestivo, quanto na utilização das bibliotecas.

Agora você pode tentar refazer o projeto do zero para verificar se conseguiu fixar o conteúdo e pode fazer suas modificações a medida que for entendendo cada parte.

Heitor: O que vamos aprender nessa aula?

Lira: Vamos aprender a criar um chat ao vivo com Python!

Hashzap

Welcome to Hashzap!

Heitor: O que vamos aprender nessa aula?

Lira: Vamos aprender a criar um chat ao vivo com Python!

Parte 6

Conclusão

Conclusão

Conclusão

Nesse projeto tivemos alguns desafios diferente dos outros que fizemos. Utilizamos bibliotecas diferentes, foi necessário utilizar uma biblioteca para criação de sites e para criação de chats.

Tivemos que utilizar um pouco de html para formatar a estrutura do site, pois os sites utilizam essa linguagem de programação. Te mostrei também como fica o visual do site depois de utilizar o CSS para melhorar a parte visual para não ficar um site “cru”.

E o mais importante, você conseguiu criar mais um projeto de Python mesmo sem saber nada, só de assistir a aula, reproduzir o que fizemos e ler a documentação você já vai aprendendo cada vez mais.

Um ponto muito importante em programação é a prática, pois você vai pegando o jeito, aprende como fazer uma ação de formas diferentes, aprende como resolver problemas, como ser mais eficiente, aprender com os erros e ainda aprende com o código das outras pessoas.

A programação, principalmente em Python, te abre muitas portas, pois você pode atuar em diversas áreas construindo diferentes tipos de projetos. Aqui na **Jornada Python** nós fizemos 4 projetos que você pode aplicar normalmente no dia a dia de uma empresa.

Para automatizar processos, para fazer análises de dados mais eficientes, para conseguir diminuir gastos, aumentar os lucros, para criar soluções, entre outras diversas possibilidades.

Não precisa começar pelo mais complexo, começando pelo simples você vai notar que as coisas vão funcionando e depois você vai juntando vários “simples” até chegar em um projeto mais robusto e complexo.

Espero que tenham gostado da **Jornada Python** e espero que continuem praticando e estudando Python!

{JORNADA} PYTHON

100% GRATUITO E ONLINE

Ainda não segue a gente no Instagram e nem é inscrito no nosso canal do Youtube? Então corre lá!



@hashtagprogramacao



youtube.com/hashtag-programacao

