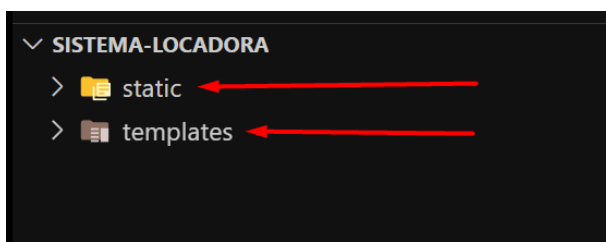


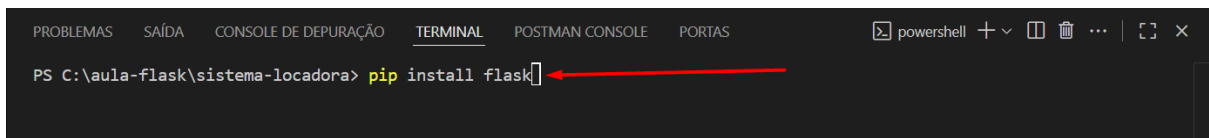
# Projeto Sistema de Gestão da Locadora "Cineflix" Prof.: Caio Malheiros

## Criando o projeto e listando todos os clientes

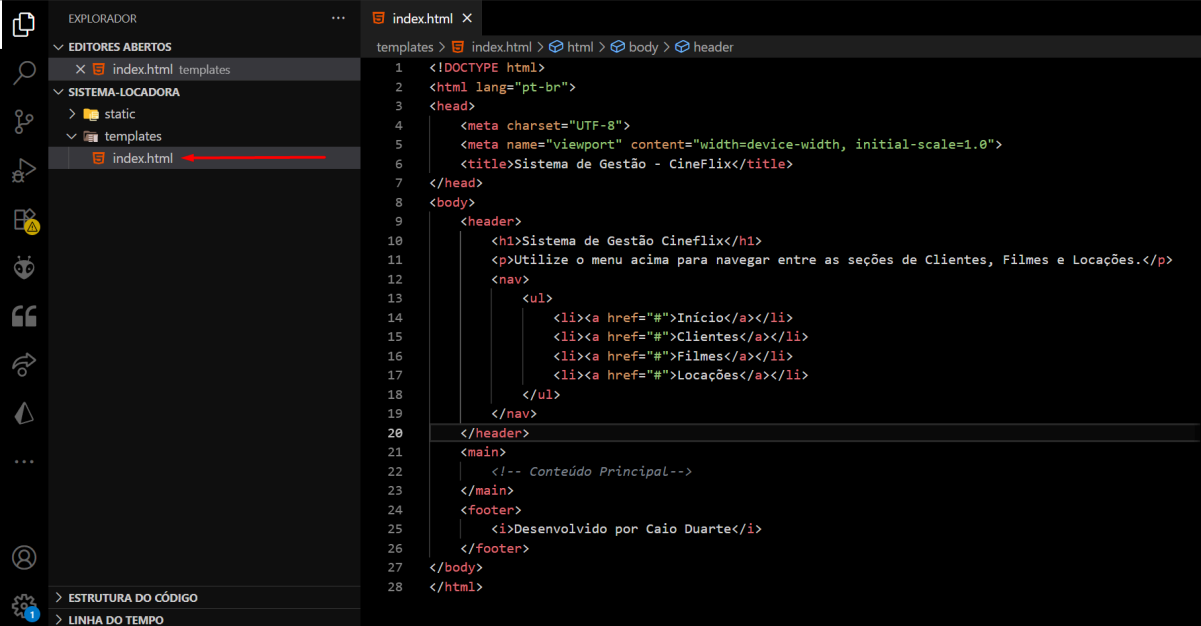
1 - Crie uma pasta chamada “sistema-locadora”, abra no visual studio code e em seguida crie a seguinte estrutura:



2- Abra o terminal do vscode, e utilize o comando abaixo para instalar o **flask** no projeto:



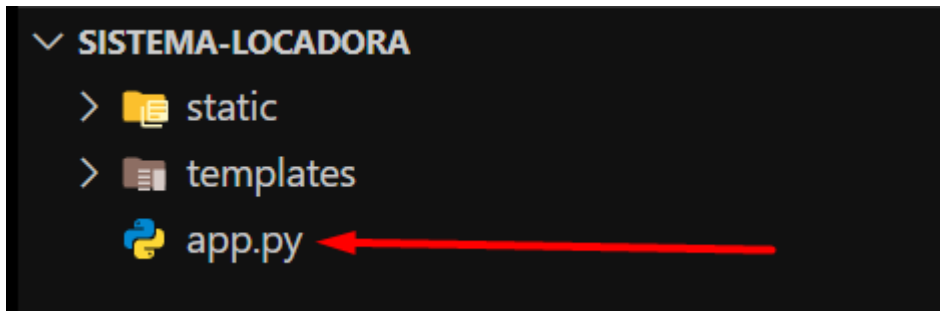
3 - Agora dentro da pasta templates, crie um arquivo chamado **index.html**:



The screenshot shows a code editor interface with a dark theme. On the left, the 'EXPLORADOR' (Explorer) sidebar is open, showing the project structure. Under 'SISTEMA-LOCADORA', there is a 'static' folder and a 'templates' folder. The 'index.html' file is highlighted within the 'templates' folder, with a red arrow pointing to it. The main editor area shows the content of 'index.html', which is an HTML document for a system called 'Sistema de Gestão CineFlix'. The code includes a DOCTYPE declaration, language and charset attributes, a viewport meta tag, a title, a header section with a main heading and a navigation menu, a main content area with a placeholder comment, and a footer with the developer's name.

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Sistema de Gestão - CineFlix</title>
7 </head>
8 <body>
9   <header>
10    <h1>Sistema de Gestão CineFlix</h1>
11    <p>Utilize o menu acima para navegar entre as seções de Clientes, Filmes e Locações.</p>
12    <nav>
13      <ul>
14        <li><a href="#">Início</a></li>
15        <li><a href="#">Clientes</a></li>
16        <li><a href="#">Filmes</a></li>
17        <li><a href="#">Locações</a></li>
18      </ul>
19    </nav>
20  </header>
21  <main>
22    <!-- Conteúdo Principal -->
23  </main>
24  <footer>
25    <i>Desenvolvido por Caio Duarte</i>
26  </footer>
27 </body>
28 </html>
```

4 - Agora vamos criar o arquivo principal da aplicação, sendo assim crie um novo arquivo chamado **app.py** dentro da pasta principal do projeto:



5 - Agora vamos criar a codificação do arquivo **app.py**:

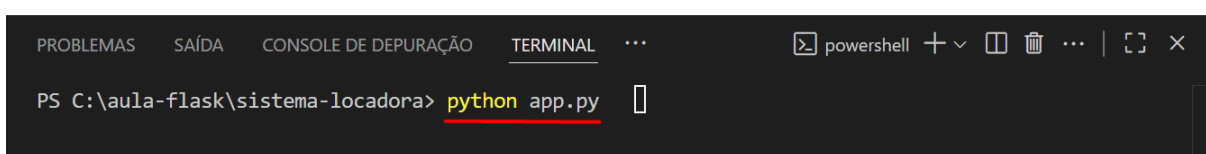
```
# Importa as funções necessárias do Flask
from flask import Flask, render_template

# Cria a aplicação Flask
app = Flask(__name__)

# -----
# Página inicial (rota "/")
# -----
@app.route('/')
def index():
    # Renderiza o arquivo "index.html" dentro da pasta templates
    return render_template("index.html")

# -----
# Inicia o servidor Flask se rodar o arquivo direto
# -----
if __name__ == "__main__":
    app.run(debug=True) # debug=True atualiza automaticamente a cada alteração
```

6 - Para testar aplicação salve todos os arquivos, abra o terminal e digite o seguinte comando:

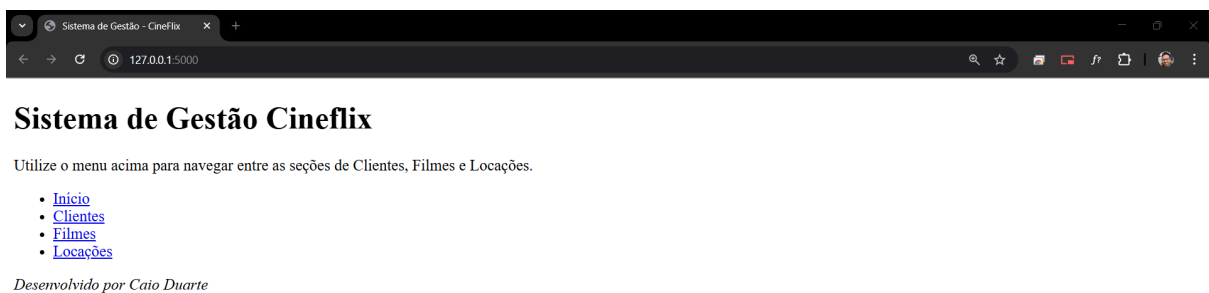


Perceba que nossa aplicação flask está **rodando** nesse **endereço** `http://127.0.0.1:5000`


```
PROBLEMAS SAÍDA CONSOLE DE DEURAÇÃO TERMINAL ... python + - [ ] [X] [Y] [Z]
```

```
* Debug mode: on  
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  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 577-654-887
```

Copie este endereço e cole no navegador conforme abaixo:



7 - Agora vamos criar uma rota para a página responsável por exibir todos os clientes cadastrados, porém antes teremos que configurar a conexão com o banco de dados mysql. Sendo assim, abra o terminal e utilize o comando abaixo para instalar a biblioteca responsável por conectar o banco de dados:

PROBLEMAS   SAÍDA   CONSOLE DE DEPURAÇÃO   TERMINAL   ...    powershell   +   ▾

```
PS C:\aula-flask\sistema-locadora> pip install mysql-connector-python
```

Resultado esperado:

```

PROBLEMAS   SAÍDA   CONSOLE DE DEPURAÇÃO   TERMINAL   ...
Download mysql_connector_python-9.4.0-cp312-cp312-win_amd64.whl.metadata (7.7 kB)
Downloading mysql_connector_python-9.4.0-cp312-cp312-win_amd64.whl (16.4 MB)
16.4/16.4 MB 13.9 MB/s eta 0:00:00
Installing collected packages: mysql-conector-python
Successfully installed mysql-conector-python-9.4.0

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\aula-flask\sistema-locadora> 

```

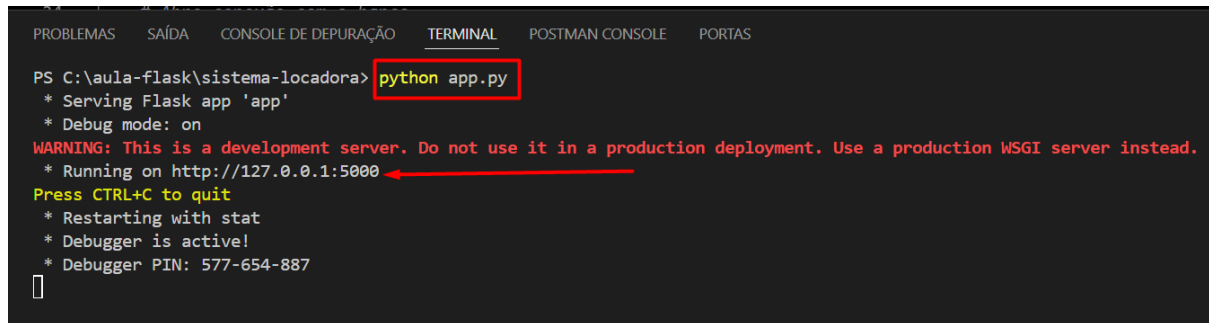
8 - Com a biblioteca do MySQL instalada, vamos configurar a conexão com o banco de dados. Sendo assim dentro do arquivo `app.py`, adicione o trecho de código abaixo:

```
app.py X
app.py > ...
1
2 # Importa as funções necessárias do Flask
3 from flask import Flask, render_template
4
5 # Importando a biblioteca do mysql
6 import mysql.connector
7
8 # Cria a aplicação Flask
9 app = Flask(__name__)
10
11 # Configurando a conexão do banco de dados MySQL
12 def get_connection():
13     return mysql.connector.connect(
14         host="localhost",      # endereço do servidor MySQL
15         user="root",          # troque pelo seu usuário
16         password="root",      # troque pela sua senha
17         database="bdlocadora" # nome do seu banco de dados
18     )
19
```

9 - Agora vamos criar uma rota responsável por executar o comando sql necessário, e renderizar uma página com os clientes encontrados no banco de dados:

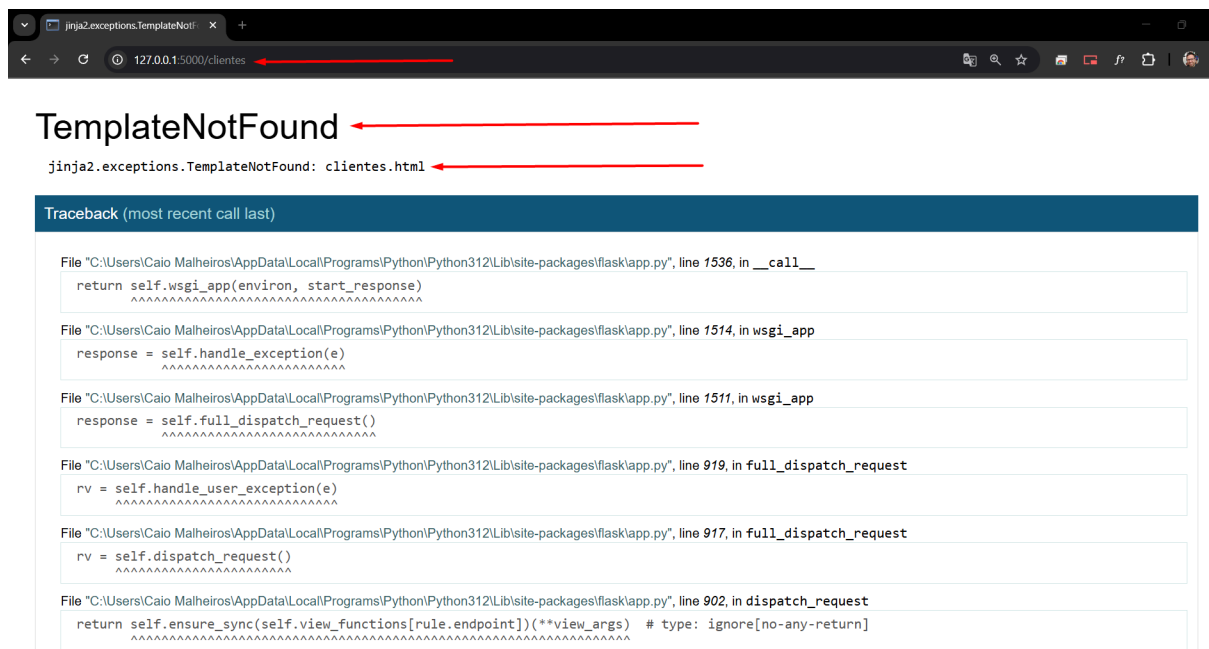
```
app.py X
app.py > ...
21
22 # -----
23 # Página inicial (rota "/")
24 # -----
25 @app.route('/')
26 def index():
27     # Renderiza o arquivo "index.html" dentro da pasta templates
28     return render_template("index.html")
29
30 # -----
31 # Rota para listar todos os clientes
32 # -----
33 @app.route('/clientes')
34 def listar_clientes():
35     # Abre conexão com o banco
36     conn = get_connection()
37     # Cria cursor para executar comandos SQL
38     cursor = conn.cursor(dictionary=True) # dictionary=True faz o retorno ser em dicionário
39     # Executa comando SQL para buscar todos os clientes
40     cursor.execute("SELECT * FROM tblcliente")
41     # Pega todos os resultados da consulta
42     clientes = cursor.fetchall()
43     # Fecha a conexão
44     conn.close()
45     # Envia os dados para o template "clientes.html"
46     return render_template("clientes.html", clientes=clientes)
47
```

Salve todos os arquivos, execute o servidor pelo terminal e teste no navegador:



```
PS C:\aula-flask\sistema-locadora> python app.py
* Serving Flask app 'app'
* Debug mode: on
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
* Restarting with stat
* Debugger is active!
* Debugger PIN: 577-654-887
```

Resultado esperado:



TemplateNotFound

jinja2.exceptions.TemplateNotFound: clientes.html

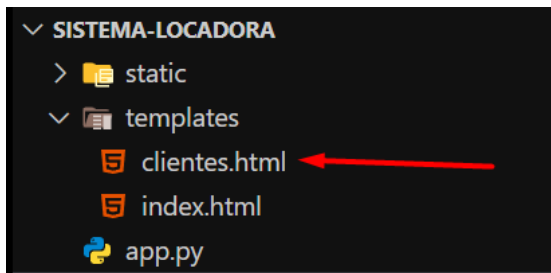
Traceback (most recent call last)

```
File "C:\Users\Caio Malheiros\AppData\Local\Programs\Python\Python312\Lib\site-packages\flask\app.py", line 1536, in __call__
    return self.wsgi_app(environ, start_response)
File "C:\Users\Caio Malheiros\AppData\Local\Programs\Python\Python312\Lib\site-packages\flask\app.py", line 1514, in wsgi_app
    response = self.handle_exception(e)
File "C:\Users\Caio Malheiros\AppData\Local\Programs\Python\Python312\Lib\site-packages\flask\app.py", line 1511, in wsgi_app
    response = self.full_dispatch_request()
File "C:\Users\Caio Malheiros\AppData\Local\Programs\Python\Python312\Lib\site-packages\flask\app.py", line 919, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "C:\Users\Caio Malheiros\AppData\Local\Programs\Python\Python312\Lib\site-packages\flask\app.py", line 917, in full_dispatch_request
    rv = self.dispatch_request()
File "C:\Users\Caio Malheiros\AppData\Local\Programs\Python\Python312\Lib\site-packages\flask\app.py", line 902, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)  # type: ignore[no-any-return]
```

Perguntas norteadoras:

- O que aconteceu? Deu certo?
- Qual é o problema da aplicação?
- O que é preciso fazer para consertar?

10 - Crie um arquivo chamado **clientes.html** dentro da pasta **templates**:

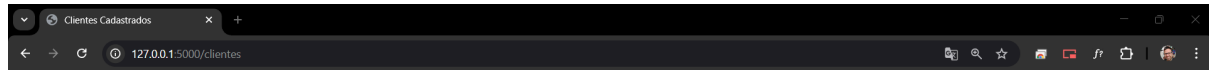


11 - Codifique o arquivo **clientes.html** conforme abaixo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Clientes Cadastrados</title>
</head>
<body>
  <h1>Clientes Cadastrados</h1>
  <table border="1">
    <tr>
      <th>Código</th>
      <th>RG</th>
      <th>Nome</th>
      <th>Endereço</th>
      <th>Bairro</th>
      <th>Cidade</th>
      <th>Estado</th>
      <th>Telefone</th>
      <th>Email</th>
      <th>Data Nascimento</th>
      <th>Estado Civil</th>
    </tr>
    <tr>
      <td>1</td>
      <td>123456789</td>
      <td>Márcio Silva</td>
      <td>Rua A, 123</td>
      <td>Centro</td>
      <td>São Paulo</td>
      <td>SP</td>
      <td>(11) 98765-4321</td>
      <td>marcio@gmail.com</td>
      <td>01/01/1990</td>
      <td>Solteiro</td>
    </tr>
  </table>
</body>
</html>
```

Salve todos os arquivos, execute a aplicação pelo terminal e em seguida teste no navegador:

### Resultado esperado :



#### Clientes Cadastrados

Código	RG	Nome	Endereço	Bairro	Cidade	Estado	Telefone	Email	Data Nascimento	Estado Civil
1	123456789	Márcio Silva	Rua A, 123	Centro	São Paulo	SP	(11) 98765-4321	marcio@gmail.com	01/01/1990	Solteiro

### Perguntas norteadoras:

- Qual foi o resultado esperado?
- A funcionalidade funcionou com sucesso?
- Quais são os próximos passos?

12 - Agora vamos utilizar parte da **sintaxe do Jinja2**, que é o **motor de templates** usado pelo Flask para gerar **HTML dinâmico**.

- **{{ ... }}** → são chamadas **expressões de interpolação**.
  - Servem para exibir **valores de variáveis** dentro do HTML.
- **{% ... %}** → são chamadas **tags de controle**.
  - Servem para **estruturas de controle**, como **for, if, else, endfor**, etc.

Resumindo:

- **{{ ... }}** → mostra valores.
- **{% ... %}** → controla a lógica (condições, laços, blocos).



Sabendo como o jinja2 funciona, substitua os <td> com dados estáticos pelo trecho de código abaixo:

```
{% for cliente in clientes %}
<tr>
  <td>{{ cliente.COD_CLIENTES }}</td>
  <td>{{ cliente.RG }}</td>
  <td>{{ cliente.NOME }}</td>
  <td>{{ cliente.ENDERECO }}</td>
  <td>{{ cliente.BAIRRO }}</td>
  <td>{{ cliente.CIDADE }}</td>
  <td>{{ cliente.ESTADO }}</td>
  <td>{{ cliente.TELEFONE }}</td>
  <td>{{ cliente.EMAIL }}</td>
  <td>{{ cliente.DATA_NASCIMENTO }}</td>
  <td>{{ cliente.ESTADO_CIVIL }}</td>
</tr> {% else %} <tr>
  <td colspan="3">Nenhum cliente cadastrado.</td>
</tr> {% endfor %}
```

Salve todos os arquivos, restart a aplicação pelo terminal e teste no navegador:

Resultado esperado: