

## **Padrão de projeto Composto *Model-View-Controller* (MVC):**

Essa disciplina trouxe a ideia de que havia apenas três categorias de padrões de projetos:

- Padrões de Criação;
- Padrões Estruturais;
- Padrões Comportamentais.

No entanto, durante implementação de um software, percebemos que as categorias de padrões de projeto não trabalham isoladamente, pois, são usados diretamente ou indiretamente de maneira combinadas para resolver um problema e será por trás desta sistemática que o *Template Method* age como um modelo que estância as classes filhas na distribuição ou criação de projetos utilizando uma base única.

### **Entendendo padrões de projetos compostos:**

Este tipo de padrão combina dois ou mais padrões em uma solução que resolve um problema apresentando-se como um conjunto de padrões que trabalha em conjunto para alcançar uma solução geral para um problema.

### **Entendendo padrões de projeto composto: *Model-View-Controller* ou Padrão Modelo-Visão-Controlado:**

Segundo Giridhar (2016, pg. 149), o “MVC é um padrão de software para implementar interfaces de usuário e uma arquitetura que pode ser facilmente modificada e mantida”.

- Essencialmente, MVC separa a aplicação em três partes: **Modelo**, **visão** e **Controlador**, interconectando os módulos e separando a maneira como a informação é representada e de que forma ela é apresentada:

- Esse modelo representa os dados e a lógica de negócios, ou seja, como a informação é armazenada e consultada:

- A *View* é a maneira como os dados serão apresentados ao usuário;

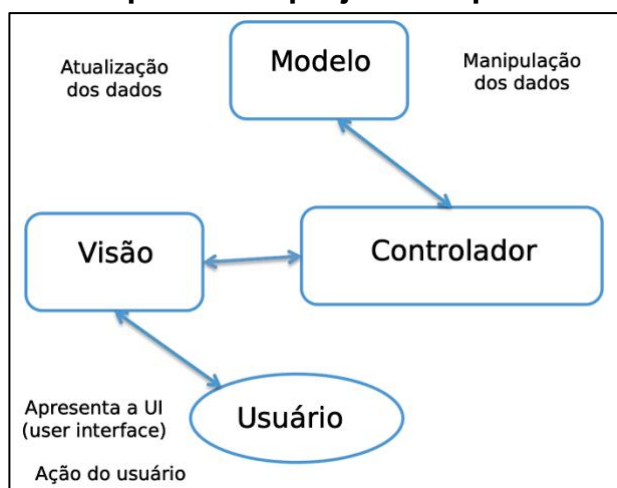
- O *Controller* é a parte que direciona o modelo e a visão definindo como eles devem se comportar de acordo com as necessidades do negócio e do usuário.

É importante observar que a *view* e o *controller* são dependentes do modelo, mas não o contrário, devido ao fato do usuário estar preocupado com os dados. Trabalhar de forma independente é o aspecto essencial do padrão de projeto composto MVC, (GIRIDHAR, 2016).

O desenvolvimento de aplicações para internet é um exemplo clássico para a utilização do padrão de projeto composto MVC.

- Ao analisar o que ocorre em um web site:
  - O usuário clica em um botão, algumas operações são processadas e a tela da aplicação aparece no *browser*;
  - Desta maneira, o usuário interage com a *view*, pois ela é a página apresentada, depois, clica-se nos botões da *view* e ela informa ao controlador o que deve ser feito;
  - Em seguida o *controller* captura a entrada da *view* e a envia ao *model*. O *model* é manipulado de acordo com as ações requisitadas pelo usuário;
  - Os controladores também podem pedir à *view* para mudar conforme a ação recebida do usuário, por exemplo, alterar os botões, apresentar elementos adicionais na interface do usuário etc.;
  - O modelo notifica a alteração de estado à *view*, e isso pode ser feito com base em alterações internas ou por acionamentos externos, como cliques em um botão;
  - A *view* exibe o estado que ela capturou diretamente do modelo. Exemplo, se um usuário fizer login no site, uma visão de painel poderá ser apresentada a ele após o login e todos os detalhes que devem ser preenchidos no painel são fornecidos à *view* pelo *model*.
- Nesse cenário, entende-se que o padrão de projeto composto MVC trabalha com a seguinte estrutura e termos: *Model*, *View*, *Controller* e *Client*, onde:
  - *Model*: declara uma classe para armazenar e manipular os dados;
  - *View*: declara uma classe para construir interfaces de usuário e fazer exibições de dados;
  - *Controller*: declara uma classe que conecta o *model* e a *view*;
  - *Client*: é onde o usuário declara uma classe que solicita determinados resultados com base nas regras de negócio e ações desenvolvidas (GIRIDHAR, 2016, p. 141-142).

### Fluxo do padrão de projeto composto MVC:



Fonte: (Giridhar, 2016, p. 144).

Segundo o autor, para entender o padrão de projeto composto MVC na terminologia do desenvolvimento de software, devemos ficar atentos nas principais classes envolvidas:

- A classe *model* é usada para definir todas as operações que ocorrem nos dados (por exemplo, criar, modificar e apagar), além de oferecer métodos para usar os dados;
- A classe *view* é uma representação da interface de usuário. Ela terá métodos que nos ajudam a construir interfaces *web* ou *GUI* conforme o contexto e as necessidades da aplicação. Ela não deve conter nenhuma lógica própria, mas deve apenas exibir os dados que receber.
- A classe *controller* é usada para receber dados da requisição e enviá-los a outras partes do sistema. Tem métodos usados para encaminhar as requisições Fonte: (Giridhar, 2016, p. 144).

### **Fluxo do padrão de projeto composto MVC:**

- Manter os dados e a sua apresentação separados;
- Facilitar a manutenção das classes e de sua implementação;
- Ter flexibilidade para mudar o modo como os dados são armazenados e exibidos; ambos são independentes e, portanto, têm flexibilidade para mudar;

### **Entendendo em detalhes a *model*, *View* e *Controller*:**

- **Models** - conhecimento da aplicação: segundo Giridhar (2016, p. 144) *apud* Zlobin (2013). A model é a pedra angular de uma aplicação, independe da visão e do controlador. Entenda que a view e o controller são dependentes do modelo.

- A modelo também fornece os dados solicitados pelo cliente. Geralmente nas aplicações, o modelo é representado pelas tabelas do banco de dados que armazenam e devolvem informações. A model tem estados e métodos para alterar esses estados, mas não tem ciência de como os dados serão vistos pelo cliente.

- É crucial que a model permaneça consistente quando houver várias operações; caso contrário, o cliente poderá obter dados corrompidos ou exibir dados desatualizados, o que é extremamente indesejável.

- Como a model é totalmente independente, os desenvolvedores que trabalham nessa parte podem manter o foco na manutenção sem ter as alterações mais recentes da view.

- A **view** é uma representação dos dados na interface vista pelo cliente. A visão pode ser desenvolvida de forma independente, mas não deve conter nenhuma lógica complexa. A lógica deve continuar no controlador ou no modelo.

- No mundo real, as views devem ser suficientemente flexíveis e apropriadas para diversas plataformas, por exemplo, desktop, dispositivos móveis, tablets e vários tamanhos de tela.

- As views devem evitar interagir diretamente com os bancos de dados e devem contar com a model para obter os dados que precisa.

- **Controller - a cola:** controla a interação do usuário na interface. Quando o usuário clicar em determinados elementos da interface, conforme a interação realizada, o *controller* fará uma chamada a *model* que, por sua vez, criará, atualizará ou apagará os dados.

- Os controllers também passam os dados para a visão, que renderiza a informação de modo que o usuário possa visualizá-la na interface.

- O *controller* não deve fazer chamadas ao banco de dados nem se envolver na apresentação dos dados. Ele deve atuar como uma cola entre o modelo e a visão, e deve ser o mais enxuto possível.

- Para dar forma às teorias vamos apresentar uma aplicação exemplo em Python, onde o código implementa o padrão de projeto MVC:

Considere o cenário onde precisamos desenvolver uma aplicação que informe os produtos oferecidos por uma empresa:

**1ª classe** Modelo, cujo objetivo é capturar os produtos e preços, simulando os o que será vendido pela uma empresa:

```
# Primeiro, definimos o Modelo. Este é o componente do MVC que lida
# com os dados do aplicativo – em nosso exemplo é um simples
# dicionário de produtos.
class Modelo:

    def __init__(self):
        self.produtos = {
            'ps5': {'id': 1, 'descricao': 'Playstation 5', 'preco': 4420},
            'xboxx': {'id': 2, 'descricao': 'xbox series x', 'preco': 4349},
            'switch': {'id': 3, 'descricao': 'Nintendo Switch', 'preco': 2176},
        }
```

**2ª classe** Controlador recebe os produtos cadastrados e prepara a listagem:

```
# Classe recebe os produtos cadastrados e os converte para a classe visão
class Controlador:

    def __init__(self):
        # Controlador inicializado com a classe Modelo
        self.modelo = Modelo()

    # O método listar produtos prepara a listagem para impressão
    def listar_produtos(self):
        produtos = self.modelo.produtos.keys()

        print('>>> P R O D U T O S >>>')
        for chave in produtos:
            print(f'ID: {self.modelo.produtos[chave]["id"]}')
            print(f'Descrição: {self.modelo.produtos[chave]["descricao"]}')
            print(f'Preço: {self.modelo.produtos[chave]["preco"]} \n')
```

**3ª classe view**, apresentar as informações ao cliente.

```
# Classe que apresenta os dados ao usuário.
class Visao:

    def __init__(self):
        # Visão é inicializada com a classe Controlador
        self.controlador = Controlador()

    # O método produtos usa a classe Controlador para capturar
    # o layout dos produtos e apresentá-los ao cliente
    def produtos(self):
        self.controlador.listar_produtos()

# Cliente que está utilizando o aplicativo para ver os produtos
if __name__ == '__main__':
    visao = Visao()
    visao.produtos()
```

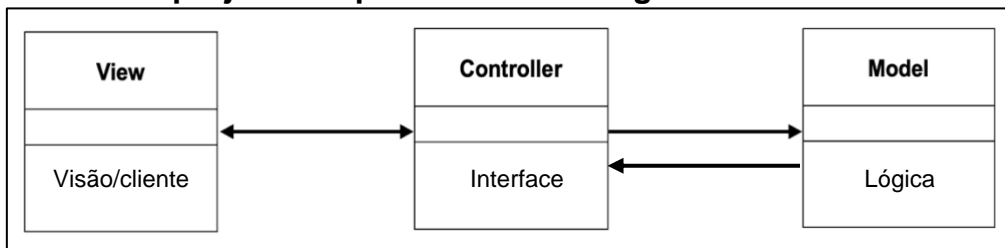
**4º resultado:**

```
>>> P R O D U T O S >>>
ID: 1
Descrição: Playstation 5
Preço: 4420

ID: 2
Descrição: xbox series x
Preço: 4349

ID: 3
Descrição: Nintendo Switch
Preço: 2176
```

**Padrão de projeto composto MVC e o diagrama de classes UML:**



- Analisando o diagrama UML do padrão de projeto composto MVC:

- A view define a visão ou a representação vista pelo cliente;
- O controller é essencialmente uma interface que está entre a view e o model direcionando as ações que o cliente realiza para o modelo e entregando os resultados do modelo para a visão;
- O model define a lógica de negócio ou as operações associadas a determinadas tarefas do cliente.

**Desenvolvendo um projeto no padrão composto MVC para web:** Num cenário onde precisamos desenvolver uma aplicação em Python, Flask e SQLite3, para uma empresa que está iniciando, o cadastro seus produtos (CRUD) para oferece-los futuramente na nuvem – no primeiro momento os usuários são pessoas da empresa que irão testar as funcionalidades da plataforma de cadastro enquanto cadastram os produtos.

### # Defina a estrutura de diretórios:

1º - Crie a pasta principal de seu projeto: **PROJ\_MVC**

2º - Crie a pasta models

3º - Crie a pasta controllers

4º - Crie a pasta templates

5º - Crie a pasta static

- Crie as subpastas

```

  > static
    > css
    > images
    > js
  
```

```

  PROJ_MVC_FLASK
  > __pycache__
  > controllers
  > models
  > static
  > templates
  
```

Nesta etapa utilizaremos apenas a subpasta **css**.

6º - No VsCode abra a subpasta CSS e mantenha-a aberta;

7º - Abra seu *browser* e acesse o endereço:

<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

- Procure o link CSS, selecione e copie a seguinte parte do link

<https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css>

- Abra uma nova guia no *browser* e cole link copiado:

- Na página de códigos que aparece – leve o cursor a qualquer parte e clique com o botão direito do mouse e selecione a opção salvar página como:



- Salve o arquivo dentro da subpasta **css** do projeto com o nome de **bootstrap.min.css** no formato Código-fonte da página;

- Feche o *browser*;

```

  > static
    > css
    # bootstrap.min.css
  
```



**8º** - No VsCode abra a pasta principal de seu projeto:

- crie os arquivos `app.py` e `__init__.py` salve-os vazio. O `app.py` é o arquivo que irá iniciar a aplicação;

- Ainda na pasta principal do projeto, crie o arquivo `database.py` e salve-o vazio, este será o arquivo que instância o SQLAlchemy para uso em outros módulos;

**9º** - No VsCode abra a subpasta **controller**, crie os arquivos com `__init__.py` e **`produto_controller.py`** e salve-os vazio;

**10º** - No VsCode abra a subpasta model, crie e salve o arquivo **`produto_model.py`** – salve-o vazio

**11º** - No VsCode abra a subpasta views e nela crie e salve vazio os seguintes arquivos: `base.html`, `index.html`, `novo.html` e `atualiza.html`

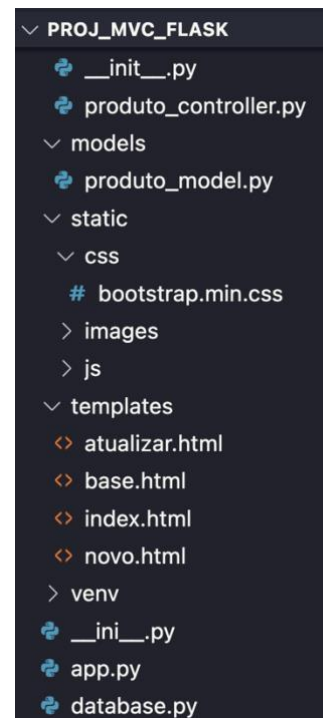
**12º** - A estrutura para desenvolver o projeto está pronta:

**13º** - No VsCode abra a pasta do principal do projeto. Estamos considerando que o ambiente virtual para este projeto já fora criado e está ativo com o nome de `venv`:

**14º** - Com a virtual env ativa, instale o Flask;

**15º** - Após execute o comando: `pip freeze > requirements.txt`

- Observe que no VsCode foi criado o arquivo com mesmo nome, que são os requisitos de nosso projeto.



```
≡ Requirements.txt
1  blinker==1.6.2
2  click==8.1.3
3  Flask==2.3.2
4  Flask-SQLAlchemy==3.0.3
5  greenlet==2.0.2
6  itsdangerous==2.1.2
7  Jinja2==3.1.2
8  MarkupSafe==2.1.3
9  SQLAlchemy==2.0.16
10 typing_extensions==4.6.3
11 Werkzeug==2.3.6
```

**17º** - O primeiro arquivo que iremos trabalhar será o **database.py**

- Com ele iremos criar uma instância do SQLAlchemy para uso em outros

módulos;

- Nesse projeto iremos trabalhar com o banco que já vem integrado ao Python3 que se chama SQLite3. Por isso não haverá necessidade de instalar e nem configurar nada.

**18º** - O segundo arquivo que iremos trabalhar é o `produto_model.py`

▼ models

🔗 produto\_model.py

```
from flask_sqlalchemy import SQLAlchemy
```

```
db = SQLAlchemy() # Cria uma instância do SQLAlchemy para uso em outros módulos
```

```
7 # Importação da instância do SQLAlchemy criada no arquivo database.py
8 from database import db
9
10 # Definição da classe Produto, que representa a tabela 'produtos' no banco de dados
11 class Produto(db.Model):
12     __tablename__ = 'produtos' # Definindo o nome da tabela no banco de dados
13
14     # Definição das colunas da tabela
15     id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Identificador único do produto
16     descricao = db.Column(db.String, nullable=False) # Descrição do produto
17     preco = db.Column(db.Float, nullable=False) # Preço do produto
18     status = db.Column(db.Integer, default=1) # Status do produto: ativo (1) ou inativo (0)
19
20     # Método construtor da classe
21     def __init__(self, descricao, preco, id=None):
22         self.id = id
23         self.descricao = descricao
24         self.preco = preco
25
26     # Método para salvar um produto no banco de dados
27     def salvar(self):
28         db.session.add(self) # Adicionando o produto na sessão do SQLAlchemy
29         db.session.commit() # Salvando as alterações no banco de dados
```



```

30
31     # Método para atualizar um produto no banco de dados
32     def atualizar(self):
33         db.session.commit() # Salvando as alterações no banco de dados
34
35     # Método para deletar um produto no banco de dados
36     def deletar(self):
37         db.session.delete(self) # Removendo o produto da sessão do SQLAlchemy
38         db.session.commit() # Salvando as alterações no banco de dados
39
40     # Método estático para obter todos os produtos do banco de dados
41     @staticmethod
42     def get_produtos():
43         return Produto.query.all() # Consulta que retorna todos os produtos
44
45     # Método estático para obter um produto específico pelo ID
46     @staticmethod
47     def get_produto(id):
48         return Produto.query.get(id) # Consulta que retorna o produto com o ID específico
49

```

**19º** - O terceiro arquivo que iremos trabalhar é o **produto\_controller.py**

PROJ\_MVC  
 controllers  
 produto\_controller.py

controllers > produto\_controller.py > ...

```

1  """
2  Claudinei de Oliveira - pt-BR - 19-06-2023
3  Manipulando o banco de dados sqlite3
4  Adaptado de Giridhar, 2016
5  produto_controller.py
6  """
7
8  # Importando os módulos necessários do Flask
9  from flask import Blueprint, render_template, request, redirect, url_for
10
11 # Importando a classe 'Produto' do arquivo produto_model.py
12 from models.produto_model import Produto
13
14 # Definindo um Blueprint - este é um objeto que permite definir rotas em
15 # um módulo separado. # Um Blueprint, em Flask, é um jeito de organizar
16 # um grupo de rotas relacionadas, funções de visualização e outros recursos de código
17 produto_blueprint = Blueprint('produto', __name__)
18
19 # Definindo a rota raiz "/" e associando a função 'index' a esta rota
20 # Quando um cliente solicita a URL '/', a função 'index' é chamada e a
21 # página retornada é renderizada.
22 @produto_blueprint.route("/")
23 def index():
24     # Consultando todos os produtos do banco de dados
25     produtos = Produto.get_produtos()
26     # Renderizando o template 'index.html', passando a lista de produtos para ele.
27     # O método render_template renderiza um template HTML que está armazenado em
28     # uma pasta de templates predefinida
29     return render_template('index.html', produtos=produtos)

```

```
30 # Definindo a rota "/novo" e associando a função 'novo' a esta rota.
31 # Esta rota aceita os métodos GET e POST
32 @produto_blueprint.route("/novo", methods=['GET', 'POST'])
33 def novo():
34
35     # Verificando se o método da requisição foi POST
36     if request.method == 'POST':
37         # Obtendo os dados do formulário enviado na requisição POST
38         descricao = request.form.get('descricao', None)
39         preco = request.form.get('preco', None)
40
41         # Criando um novo objeto Produto com os dados obtidos do formulário.
42         produto = Produto(descricao=descricao, preco=preco)
43
44         # Salvando o novo produto no banco de dados
45         produto.salvar()
46         # Redirecionando o cliente para a rota raiz
47         return redirect(url_for('index'))
48     else:
49         # Se o método da requisição for GET, apenas renderizamos o template 'novo.html'
50         return render_template('novo.html')
51
52 # Definindo a rota "/atualiza/<int:id>/<int:status>" e associando a função 'atualiza' a esta rota
53 # Esta rota só aceita o método GET
54 @produto_blueprint.route("/atualiza/<int:id>/<int:status>", methods=['GET'])
55 def atualiza(id, status):
56     # Buscando no banco de dados o produto com o id fornecido na URL
57     produto = Produto.get_produto(id)
58
59     # Atualizando o status do produto
60     produto.status = status
61
62     # Salvando a alteração no banco de dados
63     produto.atualizar()
64     # Redirecionando o cliente para a rota raiz
65     return redirect(url_for('index'))
66
67 # Definindo a rota "/deleta/<int:id>" e associando a função 'deleta' a esta rota
68 # Esta rota só aceita o método GET
69 @produto_blueprint.route("/deleta/<int:id>", methods=['GET'])
70 def deleta(id):
71     # Buscando no banco de dados o produto com o id fornecido na URL
72     produto = Produto.get_produto(id)
73
74     # Removendo o produto do banco de dados
75     produto.deletar()
76     # Redirecionando o cliente para a rota raiz
77     return redirect(url_for('index'))
78
79 def init_app(app):
80     # Associando as funções de exibição às respectivas rotas utilizando
81     # a função add_url_rule do objeto app do Flask
82     app.add_url_rule('/', 'index', index)
83     app.add_url_rule('/produto/novo', 'novo', novo, methods=['GET', 'POST'])
84     app.add_url_rule('/produto/atualiza/<int:id>/<int:status>', 'atualiza', atualiza, methods=['GET'])
85     app.add_url_rule('/produto/deleta/<int:id>', 'deleta', deleta, methods=['GET'])
86
```

## 20º - O quarto arquivo que iremos trabalhar está na subpasta **templates** e será o arquivo **base.html**

```

templates > > base.html > html > head
1  <!--
2  Claudinei de Oliveira - pt-BR - 19-06-2023
3  Adaptado de Giridhar, 2016
4  O arquivo base.html -->
5
6
7  <!DOCTYPE html>
8
9  <html lang="pt"> <!-- define o idioma da página como português -->
10
11  <head> <!-- metadados e links para estilos e scripts -->
12
13      <meta charset='utf-8'>
14
15      <meta name='viewport' content='width=device-width, initial-scale=1'>
16      <!-- Define a viewport para tornar a página responsiva,
17           O valor 'width=device-width' define a largura da viewport para a largura do dispositivo.
18           O valor 'initial-scale=1' define o nível de zoom inicial quando a página é carregada pelo navegador -->
19
20      <link href="{{ url_for('static', filename='css/bootstrap.min.css') }}" rel="stylesheet">
21      <!-- Importa o arquivo CSS do Bootstrap, a partir da pasta estática do projeto.
22           O método url_for gera uma URL para uma rota estática, neste caso o arquivo
23           de estilo CSS do Bootstrap -->
24
25      <!-- Importa a biblioteca jQuery, necessária para o funcionamento de vários
26           componentes do Bootstrap e também para a lógica de exibição das mensagens -->
27      <!--<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script> -->
28
29      {% block head %}{% endblock %}
30      <!-- Bloco reservado para que templates filhos possam adicionar
31           conteúdo dentro da tag head – útil para adicionar estilos CSS
32           ou scripts JavaScript específicos para cada página -->
33
34  </head>
35
36  <body>
37
38      <div class='container'>
39          <!-- Abertura de uma div com a classe 'container' do Bootstrap.
40               Esta classe é um componente do Bootstrap que centraliza o
41               conteúdo da página e aplica margens automáticas -->
42
43          {% block body %}{% endblock %}
44          <!-- Bloco reservado para que templates filhos possam
45               adicionar conteúdo dentro da tag body – é onde o
46               conteúdo principal de cada página deve ser inserido. -->
47
48      </div>
49
50  </body>
51
52  </html>
53

```



**21º** - O quinto arquivo que iremos trabalhar está na subpasta **templates** e chama-se **index.html**

templates > <> index.html > table.table

```

1  <!--
2  Claudinei de Oliveira - pt-BR - 19-06-2023
3  Adaptado de Giridhar, 2016
4  O arquivo index.html que estende base.html e usa vários recursos do Jinja2,
5  incluindo herança de templates e interpolação de variáveis
6  do Jinja2 para exibir dados dinâmicos -->
7
8  {% extends 'base.html' %} <!-- baseia-se no template 'base.html' -->
9
10 {% block head %}
11     <title> --- Listagem de Produtos --- </title>
12 {% endblock %}
13
14 {% block body %}
15
16     {% if produtos %} <!-- Verifica se existe uma lista de produtos
17         Se existir, exibe a tabela de produtos -->
18
19         <h3> --- Produtos --- </h3> <!-- Cabeçalho da página -->
20
21         <table class="table"> <!-- Inicia a tabela. A classe 'table'
22             é uma classe do Bootstrap para estilizar tabelas -->
23
24             <!-- Define os cabeçalhos da tabela -->
25             <tr>
26                 <td>ID</td>
27                 <td>Descrição</td>
28                 <td>Preço</td>
29                 <td>Status</td>
30                 <td>Atualizar</td>
31                 <td>Deletar</td>
32             </tr>
33
34             {% for produto in produtos %} <!-- Para cada produto na lista de produtos,
35                 cria uma nova linha na tabela -->
36
37                 <tr> <!-- Inicia a linha para um produto -->
38                     <td>{{ produto[0] }}</td>
39                     <td>{{ produto[1] }}</td>
40                     <td>{{ produto[2] }}</td>
41
42                     {% if produto[3] %} <!-- Se o produto está ativo -->
43                         <td>Ativo</td>
44                     {% else %}
45                         <td>Inativo</td> <!-- Se não, exibe "Inativo" -->
46                     {% endif %}

```

```

47
48     {% if produto[3] %} <!-- Se o produto está ativo -->
49         <!-- Exibe um link para inativar o produto -->
50         <td><a class='btn btn-warning' href='{{ url_for("atualizar_produto", id=produto[0], status=0) }}'>Inativar</a></td>
51     {% else %} <!-- Se o produto está inativo -->
52         <!-- Exibe um link para ativar o produto -->
53         <td><a class='btn btn-success' href='{{ url_for("atualizar_produto", id=produto[0], status=1) }}'>Ativar</a></td>
54     {% endif %}
55
56     <!-- Exibe um link para deletar o produto -->
57     <td><a class='btn btn-danger' href='{{ url_for("deletar_produto", id=produto[0]) }}'>Deletar</a></td>
58 </tr>
59
60 {% endfor %}
61
62 </table>
63
64 {% else %} <!-- Se não existem produtos -->
65     <!-- Exibe a mensagem... -->
66     <h3>Ainda não existem produtos cadastrados...</h3>
67 {% endif %}
68
69 <div>
70     <!-- Exibe um link para a página de criação de um novo produto -->
71     <h3><a class='btn btn-primary' href='{{ url_for("novo_produto") }}'>Adicionar Produto</a></h3>
72 </div>
73
74 {% endblock %}
75

```

## 22 - O sexto arquivo é o arquivo chamado novo.html

```

templates > <> novo.html > ...
1  <!-- Claudineiudinei de Oliveira - pt-BR - 19-06-2023
2  Adaptado de Giridhar, 2016 - novo.html - todas as classes são do bootstrap -->
3
4  <!-- Estende o layout base.html -->
5  {% extends 'base.html' %}
6
7  <!-- Início do bloco head, que substituirá o bloco head em base.html -->
8  {% block head %}
9      <!-- Define o título da página -->
10     <title>Adicionar Novo Produto</title>
11 {% endblock %}
12
13 <!-- Início do bloco body, que substituirá o bloco body em base.html -->
14 {% block body %}
15     <!-- Define o cabeçalho da página -->
16     <h3>Adicionar Novo Produto</h3>
17
18     <!-- Começa o bloco para exibir mensagens flash, se houver -->
19     {% with messages = get_flashed_messages() %}
20         <!-- Verifica se há mensagens para mostrar -->
21         {% if messages %}
22             <!-- Define a estrutura da mensagem, incluindo um botão para fechar a mensagem -->
23             <div class="alert alert-warning alert-dismissible fade show" role="alert">
24                 <!-- Mostra a primeira mensagem flash -->
25                 {{ messages[0] }}
26                 <!-- Define o botão para fechar a mensagem -->
27                 <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
28             </div>
29             <!-- Define um script para fechar a mensagem automaticamente após um certo tempo -->
30             <script>
31                 $(".alert").delay(5000).slideUp(200, function() {
32                     $(this).alert('close');
33                 });
34             </script>
35         {% endif %}
36     {% endwith %}

```

```

37
38 <!-- Define o formulário para adicionar um novo produto -->
39 <form action="{{ url_for('novo_produto') }}" method="POST" autocomplete='off'>
40   <!-- Define o campo para inserir a descrição do produto -->
41   <div class='mb-3'>
42     <label for='descricao' class='form-label'>Descrição</label>
43     <input class='form-control' type='text' name='descricao' id='descricao' placeholder='Informe a descrição do produto' required autofocus />
44   </div>
45
46   <!-- Define o campo para inserir o preço do produto -->
47   <div class='mb-3'>
48     <label for='preco' class='form-label'>Preço</label>
49     <input class='form-control' type='text' name='preco' id='preco' placeholder='Informe o preço do produto' required />
50   </div>
51
52   <!-- Define o botão para enviar o formulário -->
53   <div class='mb-3'>
54     <button type='submit' class='btn btn-primary mb-3'>Cadastrar</button>
55   </div>
56 </form>
57
58 <!-- Define o cabeçalho para a lista de produtos -->
59 <h3>Produtos Cadastrados</h3>
60
61 <!-- Define a tabela para mostrar os produtos cadastrados -->
62 <table class="table">
63   <!-- Define o cabeçalho da tabela -->
64   <thead>
65     <tr>
66       <th>ID</th>
67       <th>Descrição</th>
68       <th>Preço</th>
69       <th>Status</th>
70       <th>Ações</th>
71     </tr>
72   </thead>
73
74   <!-- Define o corpo da tabela -->
75   <tbody>
76     <!-- Loop para cada produto na lista de produtos -->
77     {% for produto in produtos %}
78       <tr>
79         <td>{{ produto.id }}</td>
80         <td>{{ produto.descricao }}</td>
81         <td>{{ produto.preco }}</td>
82         <!-- Verifica se o status do produto é 1 (ativo),
83             caso contrário, o status é 0 (inativo) -->
84         <td>{{ 'Ativo' if produto.status == 1 else 'Desativado' }}</td>
85
86         <td>
87           <!-- Link para a rota atualizar_produto, passando o id e o status do produto -->
88           <a href="{{ url_for('atualizar_produto', id=produto.id, status=1 if produto.status==0 else 0) }}" class="btn btn-warning btn-sm">
89             <!-- Verifica se o status do produto é 0 (inativo), caso contrário, o status é 1 (ativo) -->
90             {{ 'Ativar' if produto.status==0 else 'Desativar' }}
91           </a>
92           <!-- Link para a rota editar_produto, passando o id do produto -->
93           <a href="{{ url_for('editar_produto', id=produto.id) }}" class="btn btn-primary btn-sm">Editar</a>
94           <!-- Link para a rota deletar_produto, passando o id do produto -->
95           <a href="{{ url_for('deletar_produto', id=produto.id) }}" class="btn btn-danger btn-sm">Deletar</a>
96         </td>
97       </tr>
98     {% endfor %}
99   </tbody>
100 </table>
101 {% endblock %}
102

```



## 23 - O sétimo arquivo é o arquivo principal da aplicação chamado atualizar.py

```

templates > <> atualizar.html > ...
1  <!--
2  Claudinei de Oliveira - pt-BR - 19-06-2023
3  Adaptado de Giridhar, 2016
4  0 arquivo atualiza.html -->
5
6  <!-- herda o layout base definido em base.html -->
7  {% extends 'base.html' %}
8
9  {% block head %}
10     <title>Atualizar Produto</title>
11 {% endblock %}
12
13 <!-- conteúdo principal da página -->
14 {% block body %}
15
16     <h3>Atualizar Produto</h3>
17
18     <!-- Início do formulário para atualizar um produto -->
19     <!-- A ação do formulário é definida para apontar para a
20     rota 'editar_produto', passando o 'id' do produto como parâmetro -->
21     <form action="{{ url_for('editar_produto', id=produto.id) }}" method="POST">
22
23         <!-- Início do grupo de input para a descrição -->
24         <div class='mb-3'>
25             <!-- Rótulo para o campo de descrição -->
26             <label for='descricao' class='form-label'>Descrição</label>
27
28             <!-- entrada para a descrição com o valor inicial definido como a descrição do produto -->
29             <input class='form-control' type='text' name='descricao' id='descricao' value '{{ produto.descricao }}' required autofocus />
30         </div>
31         <!-- Fim do grupo de input para a descrição -->
32
33         <!-- Início do grupo de input para o preço -->
34         <div class='mb-3'>
35
36             <label for='preco' class='form-label'>Preço</label>
37             <!-- Campo de entrada para o preço com o valor inicial definido como o preço do produto -->
38             <input class='form-control' type='text' name='preco' id='preco' value '{{ produto.preco }}' required />
39         </div>
40         <!-- Fim do grupo de input para o preço -->
41
42         <!-- Início do grupo de input para o botão de atualizar -->
43         <div class='mb-3'>
44             <!-- Botão para enviar o formulário -->
45             <button type='submit' class='btn btn-primary mb-3'>Atualizar</button>
46         </div>
47         <!-- Fim do grupo de input para o botão de atualizar -->
48
49     </form>
50
51 {% endblock %}
52
  
```

## 24 - O oitavo arquivo está na pasta principal do projeto e chama-se \_\_init\_\_.py

```

6  # Importamos o módulo Flask
7  from flask import Flask
8
9  # Cria uma nova instância da aplicação Flask
10 # A variável __name__ é uma variável especial do Python,
11 # que retorna o nome do módulo # Neste caso, será '__init__',
12 # pois esse script será o ponto de entrada do programa
13 app = Flask(__name__)
14
  
```

## 25 - O nono arquivo está na pasta principal do projeto e chama-se app.py

```
5 # Importação das bibliotecas e módulos necessários
6 from flask import Flask, render_template, request, redirect, url_for, flash
7 from models.produto_model import Produto # Importa a classe Produto do modelo
8 from sqlalchemy import create_engine, inspect
9 from database import db # Importa a instância do SQLAlchemy em database.py
10
11 # Função para criar uma instância do aplicativo Flask
12 def create_app():
13     app = Flask(__name__) # Cria a instância do aplicativo Flask
14     # Configura o URI do banco de dados
15
16     app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///Users/proj_mvc_flask/ifro2023'
17     # Adiciona uma chave secreta para permitir flash messages
18     app.secret_key = 'seu segredo'
19
20     # Inicializa o SQLAlchemy com o aplicativo Flask
21     db.init_app(app)
22     return app
23
24
25 # Função para criar as tabelas no banco de dados
26 # Cria um contexto para o aplicativo Flask
27 def create_tables(app):
28     # Cria o engine do banco de dados
29     with app.app_context():
30         engine = create_engine('sqlite:///Users/proj_mvc_flask/ifro2023', echo=True)
31
32         # Cria um inspetor para o banco de dados
33         inspector = inspect(engine)
34
35         # Captura os nomes das tabelas do banco de dados
36         tables = inspector.get_table_names()
37
38         # Verifica se a tabela 'produto' existe no banco de dados
39         if 'produto' not in tables:
40             # Cria todas as tabelas no banco de dados
41             db.create_all()
42             print('Tabelas criadas com sucesso!')
43         else:
44             print('As tabelas já existem.')
```



```
45
46 # Cria a instância do aplicativo Flask
47 app = create_app()
48
49 # Rota principal do aplicativo que exibe todos os produtos
50 @app.route('/')
51 def index():
52     produtos = Produto.get_produtos()
53
54     # Renderiza o template novo.html com os produtos
55     return render_template('novo.html', produtos=produtos)
56
57 # Rota para adicionar um novo produto
58 @app.route('/produto/novo', methods=['GET', 'POST'])
59 def novo_produto():
60
61     # Se o método da requisição é POST
62     if request.method == 'POST':
63         descricao = request.form.get('descricao', None)
64         preco = request.form.get('preco', None)
65
66         # Verifica se o preço é um número válido
67         try:
68             preco = float(preco)
69         except ValueError:
70
71             # Exibe uma mensagem de erro
72             flash('O preço deve ser um número válido.')
73             produtos = Produto.get_produtos()
74             return render_template('novo.html', produtos=produtos)
75
76         # Cria uma nova instância do produto
77         produto = Produto(descricao=descricao, preco=preco)
78         produto.salvar()
79
80     # captura todos os produtos do banco de dados
81     produtos = Produto.get_produtos()
82     return render_template('novo.html', produtos=produtos)
83
84 # Rota para atualizar um produto
85 @app.route('/produto/editar/<float:id>', methods=['GET', 'POST'])
86 def editar_produto(id):
87     produto = Produto.get_produto(id)
88     if request.method == 'POST':
89         descricao = request.form.get('descricao')
90         preco = request.form.get('preco')
91
92         # Verifica se o preço é um número válido
93         try:
94             preco = float(preco)
95         except ValueError:
96             flash('O preço deve ser um número válido.')
97             return render_template('atualizar.html', produto=produto)
98
99         produto.descricao = descricao # Atualiza a descrição do produto
100         produto.preco = preco
101         produto.atualizar()
102         return redirect(url_for('index')) # Redireciona para a rota '/'
103     return render_template('atualizar.html', produto=produto)
```

```

104
105 # Rota para atualizar o status de um produto
106
107 # Altera o 'status' adicionado como parâmetro
108 @app.route('/produto/atualizar/<int:id>/<int:status>', methods=['GET', 'POST'])
109 def atualizar_produto(id, status):
110     produto = Produto.get_produto(id)
111     produto.status = status # Atualiza o status do produto
112     produto.atualizar() # Atualiza o produto no banco de dados
113     return redirect(url_for('index')) # Redireciona para a rota '/'
114
115 # Rota para deletar um produto
116 @app.route('/produto/deletar/<id>', methods=['GET'])
117 def deletar_produto(id):
118     produto = Produto.get_produto(id) # Obtém o produto pelo ID
119     produto.deletar() # Deleta o produto do banco de dados
120     return redirect(url_for('index')) # Redireciona para a rota '/'
121
122 # Verifica se o script está sendo executado diretamente e não importado
123 if __name__ == '__main__':
124     create_tables(app) # Cria as tabelas no banco de dados
125     app.run(debug=True) # Executa o aplicativo Flask em modo de debug
126

```

**26 –** Abra o terminal do VsCode, verifique se a virtual env está ativa e execute o comando:  
**Python app.py**

**27 –** Aparecerá as seguintes mensagens:

```

2023-06-19 12:34:22,973 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2023-06-19 12:34:22,973 INFO sqlalchemy.engine.Engine SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite~_
%' ESCAPE '~' ORDER BY name
2023-06-19 12:34:22,973 INFO sqlalchemy.engine.Engine [raw sql] ()
2023-06-19 12:34:22,981 INFO sqlalchemy.engine.Engine ROLLBACK
Tabelas criadas com sucesso!
* 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
2023-06-19 12:34:23,418 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2023-06-19 12:34:23,418 INFO sqlalchemy.engine.Engine SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite~_
%' ESCAPE '~' ORDER BY name
2023-06-19 12:34:23,418 INFO sqlalchemy.engine.Engine [raw sql] ()
2023-06-19 12:34:23,419 INFO sqlalchemy.engine.Engine ROLLBACK
Tabelas criadas com sucesso!
* Debugger is active!
* Debugger PIN: 248-113-334

```

**Note:** que não precisamos executar os comandos `export FLASK_APP=app.py` e `flask run` devido ao bloco de código no final do arquivo `app.py`:

```

# Verifica se o script está sendo executado diretamente e não importado
if __name__ == '__main__':
    create_tables(app) # Cria as tabelas no banco de dados
    app.run(debug=True) # Executa o aplicativo Flask em modo de debug

```

Esse bloco, verifica se o *script* está sendo executado diretamente, e não importado. Isso ocorre quando o Python executa a linha o valor de `__name__` é `'__main__'`.

- Se a condicional for verdadeira, o Python chama a função `create_tables(app)` para criar as tabelas no banco de dados e `app.run(debug=True)` para executar o aplicativo Flask;



- O **app.run(debug=True)** equivale ao comando flask run, mesmo se seu argumento for **False**, por isso, o Flask reinicia automaticamente o **servidor** sempre que detectar uma alteração no código;

- Devido a essa característica, o aplicativo Flask roda a aplicação simplesmente executando o *script app.py* no terminal do VsCode;

- Vale lembrar que essa abordagem funciona somente para desenvolvimento local, pois, para ambientes de produção, é mais comum usar um servidor WSGI, como Gunicorn ou uWSGI, em vez de app.run().

### Sobre as linhas do terminal VsCode

- **Python app.py**: linha inicia a execução do seu aplicativo Flask que está contido no arquivo app.py;

- **INFO sqlalchemy.engine.Engine BEGIN (implicit)**: SQLAlchemy está iniciando uma transação implícita. Isso significa que as operações do banco de dados estão prestes a começar;

- **INFO sqlalchemy.engine.Engine SELECT name FROM sqlite\_master WHERE type='table' AND name NOT LIKE 'sqlite~\_%' ESCAPE '~' ORDER BY name**: SQLAlchemy está executando uma consulta SQL bruta que busca todos os nomes de tabelas em seu banco de dados SQLite;

- **INFO sqlalchemy.engine.Engine [raw sql] ()**: SQLAlchemy está apenas informando que a consulta SQL bruta foi executada;

- **INFO sqlalchemy.engine.Engine ROLLBACK**: SQLAlchemy está finalizando a transação com um ROLLBACK, que é usado para desfazer as transações que ainda não foram salvas no banco de dados;

- **Tabelas criadas com sucesso!**: Uma mensagem de confirmação indicando que as tabelas foram criadas com sucesso no banco de dados;

- **\* Serving Flask app 'app'**: O servidor Flask está iniciando e servindo seu aplicativo, que está contido no módulo 'app';

- **\* Debug mode: on**: indica que o seu aplicativo Flask está sendo executado em modo de depuração e que o servidor irá reiniciar automaticamente em caso de mudanças no código e fornecerá informações de depuração detalhadas caso algo dê errado;

- **WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead**: é uma advertência informando que o servidor de desenvolvimento Flask não é seguro para uso em ambientes de produção;

- **\* Running on http://127.0.0.1:5000**: o servidor Flask está rodando localmente (na sua máquina) no endereço 'http://127.0.0.1';

- **Press CTRL+C to quit**: Isso é apenas um lembrete de que, se você quiser parar o servidor Flask, você pode fazer isso pressionando CTRL+C no seu teclado;

- \* **Restarting with stat:** o servidor Flask está reiniciando com 'stat'. Isso faz parte da funcionalidade de recarregamento automático do Flask quando está em modo de depuração;

- \* **Debugger is active!:** indica que o depurador do Flask foi ativado. Isso permite que você veja informações detalhadas sobre qualquer erro que ocorra durante a execução do seu aplicativo;

- \* **Debugger PIN: 248-113-334:** Este é o PIN para o depurador interativo. Você precisará dele se quiser usar o depurador;

- **127.0.0.1 - - [19/Jun/2023 12:50:16] "GET / HTTP/1.1" 200 - e 127.0.0.1 - - [19/Jun/2023 12:50:16] "GET /static/css/bootstrap.min.css HTTP/1.1" 304:** essas são as solicitações HTTP feitas ao servidor Flask. A primeira é uma solicitação GET para a página inicial do seu site, que retornou um código de status HTTP 200, o que significa que a solicitação foi bem-sucedida. A segunda é uma solicitação GET para o arquivo CSS do Bootstrap, que retornou um código de status HTTP 304, indicando que o arquivo não foi modificado desde a última solicitação e, portanto, o navegador pode continuar a usar a versão em cache;

### Sobre as linhas o SQLAlchemy:

Segundo Grinberg (2018). O SQLAlchemy é uma biblioteca de mapeamento objeto-relacional (ORM) para Python que fornece um conjunto abrangente e consistente de ferramentas para interagir com bancos de dados;

- O objetivo da biblioteca é proporcionar aos desenvolvedores um meio de trabalhar com bancos de dados de maneira orientada a objetos, permitindo a manipulação de registros de banco de dados como se fossem objetos Python;

- SQLAlchemy tem duas formas de uso: o ORM e o Core. Estamos trabalhando com a primeira forma, pois, ela permite que os desenvolvedores trabalhem com o banco de dados de uma perspectiva orientada a objetos, enquanto a segunda é uma maneira mais direta de interagir com o banco de dados através de SQL;

- Um dos principais recursos do **SQLAlchemy** é a sua habilidade de criar um modelo de dados a partir de classes Python, assim como a criação automática de tabelas a partir desses modelos. Ele também oferece consultas de alto nível e suporte a transações;

- Ao usar o SQLAlchemy, os desenvolvedores podem se concentrar na lógica dos aplicativos, enquanto o SQLAlchemy cuida da interação com o banco de dados, tornando o desenvolvimento mais rápido e mais eficiente.

### Quanto as siglas ORM e CORE:

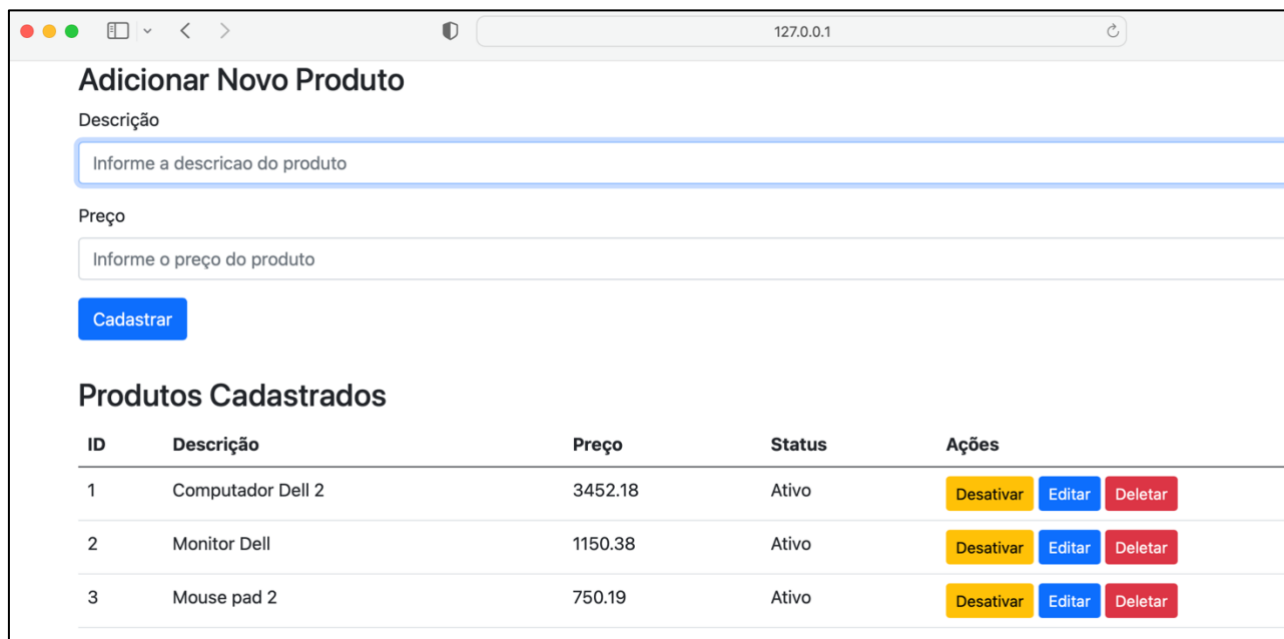
- Segundo o autor: ORM: significa **Object-Relational Mapping** (Mapeamento Objeto-Relacional), que é uma técnica que permite aos desenvolvedores interagir com seu banco de dados, como se fossem objetos Python:



- Com ORM, você pode criar classes em Python que correspondem às tabelas em seu banco de dados, e instâncias dessas classes correspondem a linhas nas tabelas. ORM permite que você escreva consultas usando Python, em vez de SQL, o que pode facilitar o desenvolvimento para quem está familiarizado com Python e não com SQL;

- Core: O SQLAlchemy Core é uma parte da biblioteca SQLAlchemy que lida com as operações de banco de dados no nível do SQL, pois, ele apresenta **interface SQL-*abstrata***, que permite ao desenvolvedor escrever consultas SQL de maneira pythonica, no modo interpretador, exigindo dessa maneira um conhecimento mais aprofundado da SQL.

**25** – Com o servidor rodando, abra seu *browser* e digite o endereço: <http://127.0.0.1:5000>



ID	Descrição	Preço	Status	Ações
1	Computador Dell 2	3452.18	Ativo	Desativar Editar Deletar
2	Monitor Dell	1150.38	Ativo	Desativar Editar Deletar
3	Mouse pad 2	750.19	Ativo	Desativar Editar Deletar

- Faça os testes de cadastrar produtos, desativar, editar e deletar.

**Exercício:** Desenvolver em Python e Flask um único arquivo.py todos os exemplos apresentados no material 11-09-2024 disponibilizado no ambiente virtual.ifro.edu.br. Após postar em um único arquivo.py no ambiente virtual.ifro.edu.br na tarefa: “Poste aqui o arquivo.py contendo os códigos exemplos disponibilizados no material – 04-11-2024 menos a subpasta env e \_\_apache - até o dia 18-09-2024 às 18:30 – vale até 10 pontos”.

## REFERENCIAS BIBLIOGRÁFICAS

F. V. C. Santos, Rafael. Python: Guia prático do básico ao avançado (Cientista de dados Livro 2) (p. 180). rafaelfvcs. Edição do Kindle.

FRAGOSO, Wallace. 2018. **Guia Prático Python & Flask: Aprenda a criar aplicações web usando o mini framework mais usado na atualidade**. Edição do Kindle.

GIRIDHAR, Chetan. **Learning Python Design Patterns - Second Edition**. 2016. Birmingham, Reino Unido.

GRINBERG, Miguel. **Desenvolvimento web com Flask: Desenvolvendo Aplicações Web com Python**. Tradução em português autorizada da edição em inglês da obra Flask Web Development ISBN: 9781491991732. Novatec Editora Ltda. 2018. São Paulo, SP – Brasil.

MUELLER, John Paul. **Programação Funcional Para Leigos**. Alta Books. Edição do Kindle, 2020.

MENEZES, Nilo Ney Coutinho. **Introdução à programação com Python**. 3ª Edição. 2019. Editora Novatec - São Paulo - SP - Brasil.

ZLOBIN, Gennadiy. **Learning Python Design Patterns**. Editora: Packt Publishing. 2013. Idioma: Inglês. ISBN-10: 1783283378 - ISBN-13: 978-1783283378

### **SOBRE o SQLITE:**

SQLite é um sistema de banco de dados relacional. Ele implementa uma grande parte do padrão SQL-92 e fornece muitas das características e funcionalidades que você esperaria de um sistema de banco de dados relacional.

Uma característica única do SQLite, no entanto, é que é um banco de dados baseado em arquivo, significando que todo o banco de dados é armazenado em um único arquivo no disco. Isso torna o SQLite extremamente portátil e fácil de integrar em uma variedade de aplicações, incluindo aplicativos de desktop, aplicativos para dispositivos móveis e até mesmo certos tipos de aplicações web.

Além disso, o SQLite é um software de código aberto e é projetado para ser muito leve e eficiente, com um tamanho de biblioteca binária muito pequeno comparado a outros sistemas de banco de dados. Ele não requer um processo de servidor separado (é incorporado diretamente no aplicativo) e não requer configuração.

Apesar de sua simplicidade e leveza, o SQLite oferece muitos recursos avançados, incluindo suporte para transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), vistas, gatilhos e muitas outras funcionalidades que você esperaria de um sistema de banco de dados relacional completo.

### **Relacionamento de tabelas no SQLITE3**

O relacionamento entre as tabelas no SQLite é feito através de chaves estrangeiras, assim como em qualquer outro sistema de banco de dados relacional.

Uma chave estrangeira é um campo (ou conjunto de campos) em uma tabela que é usado para vincular essa tabela a uma chave primária na outra tabela. As chaves estrangeiras permitem que você estabeleça relações entre as tabelas, que podem ser de um para um, um para muitos, ou muitos para muitos.

Exemplo de como criar tabelas com uma relação de chave estrangeira no SQLite:

```
CREATE TABLE Autores (  
    AutorId INTEGER PRIMARY KEY,  
    Nome TEXT NOT NULL  
);  
  
CREATE TABLE Livros (  
    LivroId INTEGER PRIMARY KEY,  
    Titulo TEXT NOT NULL,  
    AutorId INTEGER,  
    FOREIGN KEY (AutorId) REFERENCES Autores(AutorId)  
);
```

Neste exemplo, a tabela Autores tem uma chave primária AutorId, e a tabela Livros tem uma chave estrangeira AutorId que referencia a chave primária na tabela Autores. Isso cria um relacionamento entre as duas tabelas, onde cada livro está associado a um autor.

Por padrão, o SQLite permite a inserção de valores que não existem na tabela referenciada (Autores, neste caso), o que pode resultar em registros "órfãos". Se quiser reforçar a integridade referencial, você deve ativá-la explicitamente com o seguinte comando, logo após abrir a conexão:

```
PRAGMA foreign_keys = ON;
```

Após essa ativação, o SQLite verificará a existência do valor referenciado antes de permitir a inserção ou atualização de um registro na tabela Livros.