**Conexão com o banco e leitura de dados**

**Nessa aula conectaremos nossa aplicação com um banco de dados SQLite.**

A SQLite é um SGDB (Sistema de Gerenciamento de Base de Dados), assim como a MySQL e tantos outros.

Um SGDB é um software que interage com uma base de dados. Ele facilita diversas operações que as bases de dados gerenciam.

Tanto a SQLite como MySQL utilizam a linguagem SQL (Structured Query Language ou Linguagem de Consulta Estruturada).

**Diferenças entre o SQLite e o MySQL:**

A SQLite é uma base de dados independente que opera sem a necessidade de um servidor. Isso também é conhecido como uma base de dados embutida, o que significa que a engine da DB roda como parte do aplicativo. Diferente da MySQL que exige o uso de um servidor para rodar.

A SQLite armazena as informações diretamente num arquivo único, tornando fácil a sua cópia.

A SQLite não tem qualquer funcionalidade específica de gerenciamento de usuários — portanto, não é adequada para uso com múltiplos acessos. Sendo assim, a SQLite é mais indicada para bases de dados menores.

Existem certos momentos em que a SQLite pode ser mais efetiva do que usar a MySQL. Alguns desses cenários são:

Quando se desenvolve pequenos aplicativos independentes

Projetos menores que não exigem muita escalabilidade

Quando você tem a exigência de ler e escrever direto da unidade de armazenamento

Desenvolvimento básico e testes

Para utilizarmos a SQLite em nossa aplicação o primeiro passo é instalar a biblioteca **SQL Alchemy**

Para instalar, execute o seguinte comando no terminal:

**pip install -U Flask-SQLAlchemy**

Feito isso, seguindo a arquitetura MVC criaremos nossa pasta de Models que será a camada responsável pela manipulação de dados.

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Dentro da pasta “models” crie um arquivo com o nome “database.py”.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Após isso, iremos importar a biblioteca SQL Alchemy, e faremos o seu carregamento na variável “db”, conforme código abaixo:

*from* flask\_sqlalchemy *import* SQLAlchemy

db = SQLAlchemy()

Logo abaixo dessa linha, criaremos a classe “Game”, que será a classe responsável por criar a entidade “Game” no banco de dados com seus respectivos atributos:

class Game(db.Model):

    id = db.Column(db.Integer, primary\_key=True)

    titulo = db.Column(db.String(150))

    ano = db.Column(db.Integer)

    categoria = db.Column(db.String(150))

    plataforma = db.Column(db.String(150))

    preco = db.Column(db.Float)

    quantidade = db.Column(db.Integer)

    def \_\_init\_\_(self, titulo, ano, categoria, plataforma, preco, quantidade):

        self.titulo = titulo

        self.ano = ano

        self.categoria = categoria

        self.plataforma = plataforma

        self.preco = preco

        self.quantidade = quantidade

Sendo assim, no banco será criado a entidade “Game” com os atributos: id, titulo, ano, categoria, plataforma, preco e quantidade. Essa tabela será responsável por fazer um controle de estoque de games.

Agora no arquivo “app.py” faremos a importação do nosso model.

Além do model importaremos também a biblioteca OS. A OS é uma biblioteca de comandos do sistema operacional, que permite realizar várias operações, como por exemplo, ler um determinado diretório do seu computador no Python.

*from* flask *import* Flask, *render\_template*

*from* controllers *import* routes

*from* models.database *import* db

*import* os

Agora logo abaixo do método “routes.init\_app(app)” do arquivo “app.py” incluiremos o seguinte código:

app = Flask(\_\_name\_\_, template\_folder='views')

routes.*init\_app*(app)

# *Permite ler o diretório de um determinado arquivo*

dir = os.path.*abspath*(os.path.*dirname*(\_\_file\_\_))

# *Passamos o diretório ao SQLAlchemy*

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///' + os.path.*join*(dir, 'models/games.sqlite3')

Assim será criado o banco “games.sqlite3” dentro do diretório “models” que por sua vez, está no diretório de nossa aplicação.

Após isso, dentro da condição if \_\_name\_\_ == ‘\_\_main\_\_’, incluiremos o código responsável por criar de fato o banco de nossa aplicação, conforme o a seguir:

*if* \_\_name\_\_ *==* '\_\_main\_\_':

    # *Verifica no início da aplicação se o BD já existe. Caso contrário ele criará o BD.*

    db.*init\_app*(app=app)

*with* app.*test\_request\_context*():

        db.*create\_all*()

    app.*run*(host='localhost', port=5000, debug=True)

Nesse caso, o banco só será criado caso ainda não exista.

Feito essas configurações, rode a aplicação com o comando “python app.py” no terminal e verifique se o banco de dados é criado com sucesso.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Para manipularmos nosso banco faremos a instalação da ferramenta SQLiteStudio, disponível para download em:

<https://github.com/pawelsalawa/sqlitestudio/releases/download/3.4.4/SQLiteStudio-3.4.4-windows-x64-installer.exe>

Após instalar o SQLiteStudio, Indo até a pasta “models” do nosso projeto, vemos que o banco agora já possui o ícone do aplicativo. Para abrir nosso banco no SQLiteStudio basta dar dois cliques no arquivo:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Abrindo o arquivo, será exibida a tabela “game” do lado esquerdo do aplicativo, para abri-la basta dar dois cliques em cima da tabela.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Agora iremos inserir de forma manual os primeiros dados em nossa tabela, para isso vá até a guia “Dados”. Depois, insira alguns dados na tabela e pressione o botão “Commit”:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Feito isso, iremos criar a rota “/estoque” que será responsável por renderizar a página de estoque e exibir os games cadastrados no banco de dados.

Nessa rota também faremos uma consulta no banco de dados para selecionar todos os games e passar para essa págna. Para isso, usaremos o método query.all() do SQL Alchemy:

    @app.*route*('/estoque')

    def *estoque*():

        gamesestoque = Game.query.*all*()

*return* *render\_template*('estoque.html', gamesestoque=gamesestoque)

Agora iremos criar a view “estoque.html” que será a página responsável por gerir o nosso controle de estoque de games. Nela, futuramente poderemos exibir, cadastrar, editar e deletar games.

Essa página terá a seguinte estrutura:

{% extends 'base.html' %}

{% block content %}

    <h2>Controle de estoque</h2><br>

    <table *class*="table table-bordered table-hover">

        <thead *class*="table-dark">

            <tr>

                <th>Título:</th>

                <th>Ano:</th>

                <th>Categoria:</th>

                <th>Plataforma:</th>

                <th>Preço:</th>

                <th>Quantidade:</th>

            </tr>

        </thead>

        <tbody>

        {% for g in gamesestoque %}

            <tr>

                <td>{{g.titulo}}</td>

                <td>{{g.ano}}</td>

                <td>{{g.categoria}}</td>

                <td>{{g.plataforma}}</td>

                <td>{{g.preco}}</td>

                <td>{{g.quantidade}}</td>

            </tr>

        {% endfor %}

        </tbody>

    </table>

{% endblock content %}

Acessando agora a rota “localhost:500/estoque” temos o seguinte resultado:

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

Note que os jogos cadastrados no banco são exibidos dentro de uma tabela.

Aprendemos assim como criar um banco de dados SQLite no Flask e fazer a consulta de dados. No próximo tópico aprenderemos como cadastrar dados no banco.