Server-Side – Database

Católica de Santa Catarina – Centro Universitário
Jaraguá do Sul - SC, Brasil

Professor: Ph.D. Andrei Carniel (Prof. Andrei)
Contato: andrei.carniel@gmail.com
linktr.ee/andrei.carniel

SQL Alchemy

SQL Alchemy

- É um framework para acessar e manipular banco de dados e um Object Relational Mapper (ORM, Mapeamento de Objeto Relacional).
  - Flexibilizar o uso do SQL no código.
  - Versão 2.0.
  - MySQL, SQL server, Oracle...

- ORM é um Framework que visa auxiliar na redução da impedância, realizando todas as conversões necessárias entre o modelo relacional e o modelo orientado a objetos de maneira automática, geralmente da seguinte forma:
  - Cada classe é interpretada com uma tabela;
  - Cada linha de uma determinada tabela, junto com seu relacionamento é tratada como instância do objeto relacionado à tabela em questão.

https://www.sqlalchemy.org

https://pypi.org/project/SQLAlchemy/

Python - Pycharm

```
pip install flask-sqlalchemy
```

Problemas ao instalar o sqlalchemy.

- Não instalou na minha virtual environment...
  - Copiei as libs para a pasta.
  - Ou **`Ctrl+Alt+S`** > **`Project:<nome do projeto>`** > **`Phyton Interpreter`** e adicionei manualmente o **`flask_slqalchemy`** e o **`slqalchemy`** (no meu caso precisou os dois...).
- Depois disso ocorreu o seguinte erro: This typically means that you attempted to use functionality that needed the current application. To solve this, set up an application context with app.app_context(). See the documentation for more information.
- Resolvi com: `app.app_context().push()` antes do `db.create.all()`.

Telas de HTML

Teremos 3 telas nessa atividade.

- Exibir todos os estudantes cadastrados (show_all.html).
- Adicionar novo estudante (new.html).
- Atualizar um estudante existente (update.html).

Disponível no Teams.

Telas de HTML

`show_all.html`

```
<!DOCTYPE html>
<html lang = "en">
    <head></head>
    <body>
        <h3>
            <a href = "{{ url_for('show_all') }}">Comentários - Exemplo com Flask SQLAlchemy</a>
        </h3>
        <hr/>
        {%- for mensagem in get_flashed_messages() %}
            {{ mensagem }}
        {%- endfor %}
        <h3>Estudante (<a href = "{{ url_for('new') }}">Adicionar estudante</a>)</h3>
        <table>
            <thead>
                <tr>
                    <th>Nome</th>
                    <th>Cidade</th>
                    <th>Email</th>
                    <th>Pin</th>
                    <th>Ações</th>
                </tr>
            </thead>
            <tbody>
                {% for estudante in estudantes %}
                    <tr>
                        <td>{{ estudante.nome }}</td>
                        <td>{{ estudante.cidade }}</td>
                        <td>{{ estudante.email }}</td>
                        <td>{{ estudante.pin }}</td>
                        <td>
                            <a href="/update/{{estudante.id}}">Editar</a>
                            <a href="/delete/{{estudante.id}}">Deletar</a>
                        </td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </body>
</html>
```

Telas de HTML

new.html

```
<!DOCTYPE html>
<html>
   <body>

      <h3>Estudantes - Exemplo de Flask SQLAlchemy</h3>
      <hr/>

      {%- for category, message in get_flashed_messages(with_categories = true) %}
         <div class = "alert alert-danger">
            {{ message }}
         </div>
      {%- endfor %}

      <form action = "{{ request.path }}" method = "post">
         <label for = "nome">Nome</label><br>
         <input type = "text" name = "nome" placeholder = "Nome" /><br>
         <label for = "cidade">Cidade</label><br>
         <input type = "text" name = "cidade" placeholder = "Cidade" /><br>
         <label for = "email">Email</label><br>
         <textarea name = "email" placeholder = "Email"></textarea><br>
         <label for = "pin">Pin</label><br>
         <input type = "text" name = "pin" placeholder = "pin" /><br>
         <input type = "submit" value = "Submit" />
      </form>

   </body>
</html>
```

Telas de HTML

update.html

```
<!DOCTYPE html>
<html>
   <body>

      <h3>Estudantes - Exemplo de UPDATE com Flask SQLAlchemy</h3>
      <hr/>

      {%- for category, message in get_flashed_messages(with_categories = true) %}
         <div class = "alert alert-danger">
            {{ message }}
         </div>
      {%- endfor %}

      <form action = "/update/{{ estudante.id }}" class="container" method = "post">
         <label for = "nome">Nome</label><br>
         <input type = "text" name = "nome" placeholder = "Nome" value="{{ estudante.nome }}"/><br>
         <label for = "cidade">Cidade</label><br>
         <input type = "text" name = "cidade" placeholder = "Cidade" value="{{ estudante.cidade }}"/><br>
         <label for = "email">Email</label><br>
         <textarea name = "email" placeholder = "Email" >{{ estudante.email }}</textarea><br>
         <label for = "pin">Pin</label><br>
         <input type = "text" name = "pin" placeholder = "pin" value="{{ estudante.pin }}"/><br>
         <input type = "submit" value = "Submit" />
      </form>

   </body>
</html>
```

Listar dados

Estrutura básica do software para utilizarmos o banco de dados

```python
# libs que iremos utilizar neste projeto
from flask import Flask, render_template, request, flash, url_for, redirect
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__, template_folder='templates')
app.config ['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///estudantes.db' # conexao com o banco de dados
app.config['SECRET_KEY'] = "random string" # para criptografar as sessões


# desde que você tem as classes relaciandas ao banco, o ORM faz o processo de criar, recuperar e converter
db = SQLAlchemy(app)
class Estudantes(db.Model):
    # definição dos tipos de dados do banco
    id = db.Column('student_id', db.Integer, primary_key = True)
    nome = db.Column(db.String(100))
    cidade = db.Column(db.String(50))
    email = db.Column(db.String(200))
    pin = db.Column(db.String(10))

    # objeto
    def __init__(self, nome, cidade, email, pin):
        self.nome = nome
        self.cidade = cidade
        self.email = email
        self.pin = pin


# lista todos os estudantes
@app.route('/')
def show_all():
    # renderiza os dados em tela
    return render_template('show_all.html', estudantes = Estudantes.query.all())


if __name__ == '__main__':
    app.app_context().push() # recupera o contexto da aplicação que está usando o banco
    db.create_all() # cria todas as instâncias da database se não existirem
    app.run(host="0.0.0.0", port="5000", debug=True)
```

Inserir

```python
# adiciona novo estudante
# GET para quando você errar o preenchimento e precisa informar ao usuário
# POST para recuperar e salvar os dados
@app.route('/new', methods=['GET', 'POST'])
def new():
    if request.method == 'POST':
        # verifica os dados que foram preenchidos
        if not request.form['nome'] or not request.form['cidade'] or not request.form['email'] or not request.form['pin']:
            # caso algum deles não for preenchido, avisa ao usuário
            flash('Preencha todo os campos', 'Erro')
        else:
            # Recupera os dados preenchidos e coloca em um objeto
            estudante = Estudantes(request.form['nome'], request.form['cidade'], request.form['email'], request.form['pin'])
            db.session.add(estudante) # salva no banco
            db.session.commit() # commit da transação
            flash('Registro salvo com sucesso') # coloca uma mensagem para a próxima página
            return redirect(url_for('show_all')) # redireciona para a próxima página
    return render_template('new.html') # renderiza os dados em tela
```

Enquanto tiver erro ao salvar, permanece na página.

Se tiver sucesso, exibe todos os estudantes salvos.

Inserir - Erro

- Erro ao inserir: `RuntimeError: The session is unavailable because no secret key was set. Set the secret_key on the application to something unique and secret.`

- Resolvi com: `app.config['SECRET_KEY'] = "random string“`
  - Após a configuração do banco.
  - É uma chave secreta para que você use para encriptar dados sensíveis como *senhas* e *cookies*.
  - Deve ser uma chave randômica.

Deletar

```python
# deleta um estudante
@app.route('/delete/<int:id>')
def delete(id):
    estudante = Estudantes.query.get(id) # recupera o estudando pelo Id
    db.session.delete(estudante) # deleta o estudante
    db.session.commit() # commit da operação
    flash('Estudate: ' + str(id) + ' foi deletado') # retorna a mensagem de sucesso ao deletar para a próxima página
    return redirect(url_for('show_all')) # redireciona para a página que exibe os estudantes
```

É necessário buscar o estudante pelo Id, e mandar o objeto para ser deletado.
Lembre-se, estamos utilizando ORM.

Update

```python
# atualiza um estudante
@app.route('/update/<int:id>', methods=['GET', 'POST'])
def update(id):
    estudante = Estudantes.query.get(id)  # recupera o estudando pelo Id

    # verifica os dados que foram preenchidos
    if request.method == 'POST':
        if not request.form['nome'] or not request.form['cidade'] or not request.form['email'] or not request.form['pin']:
            # caso algum deles não for preenchido, avisa ao usuário
            flash('Preencha todo os campos', 'Erro')
        else:
            # atualiza o objeto os dados do objeto
            estudante.nome = request.form['nome']
            estudante.cidade = request.form['cidade']
            estudante.email = request.form['email']
            estudante.pin = request.form['pin']
            db.session.commit() # commit das atualizações
            flash('Registro salvo com sucesso')  # coloca uma mensagem para a próxima página
            return redirect(url_for('show_all')) # redireciona para a página que exibe os estudantes
    return render_template('update.html', estudante=estudante) # renderiza os dados em tela com os dados originais
```

Outras explicações

```
model.query.all()
```

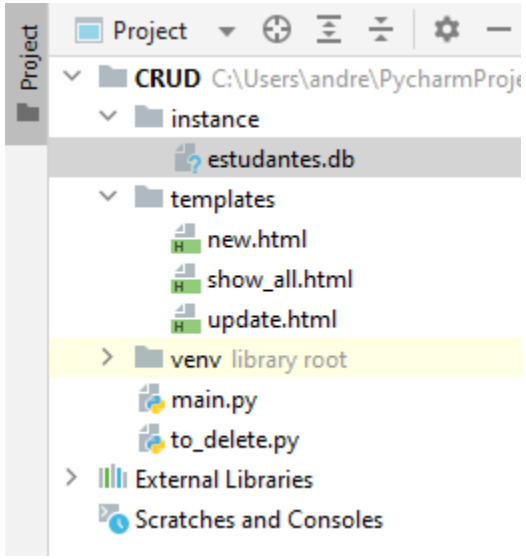
Recupera todos os registros de um determinado objeto/tabela.

```
estudantes.query.filter_by(city = 'Jaragua').all()
```

Recupera todos os registros de um determinado objeto/tabela, que residem em Jaragua.

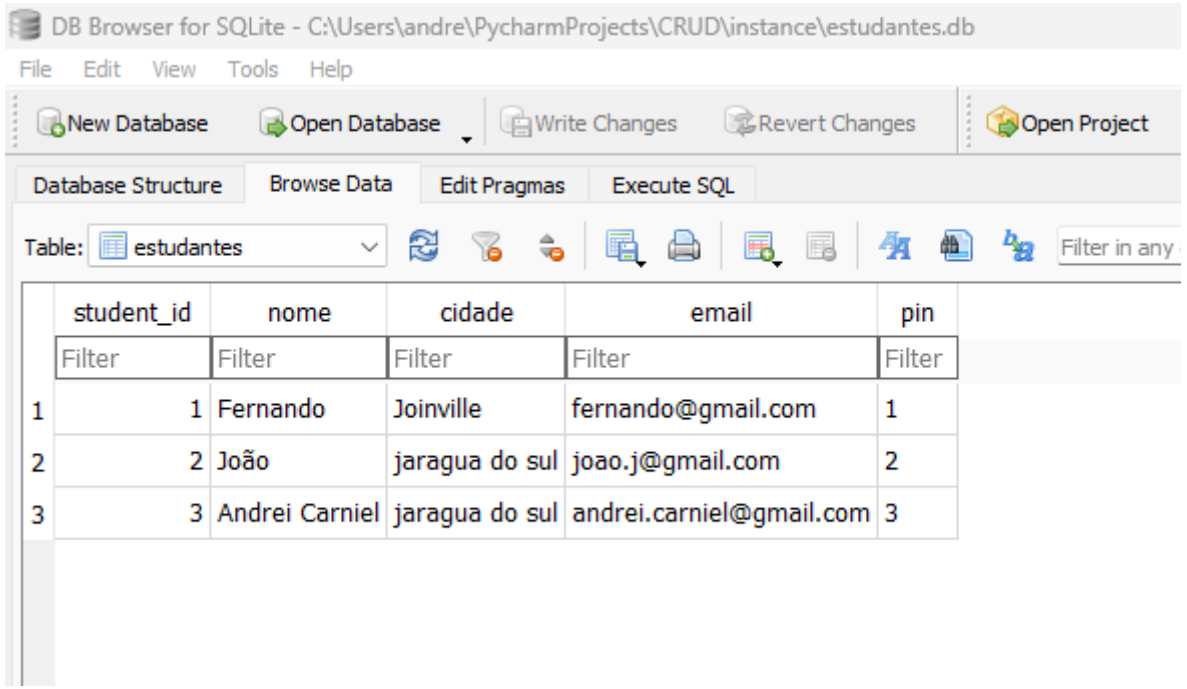
Outras explicações

O banco de dados fica salvo no diretório *instance* (criado automaticamente).



Outras explicações

Pode ser acessado utilizando o BB Browser dor SQLite.



	student_id	nome	cidade	email	pin
	Filter	Filter	Filter	Filter	Filter
1	1	Fernando	Joinville	fernando@gmail.com	1
2	2	João	jaragua do sul	joao.j@gmail.com	2
3	3	Andrei Carniel	jaragua do sul	andrei.carniel@gmail.com	3

https://sqlitebrowser.org/dl/

Avisos

- O arquivo show_all.html tem um *redirect* para o new.html. Se tentar iniciar o app sem o new.html vai ocorrer um erro.

- A lib está do Flask em constante atualização, pode ser necessário adaptar esse código no futuro devido a padronização de segurança.

- Estamos usando uma lib para acessar o banco de dados, pode ser necessário atualizar o código para atender as necessidades da lib.

- Pode ter casos onde você recebe o banco de dados pronto, e outros que você terá de criar. Utilize o tratamento de erros.

Atividades

- Reproduzir esse código em seu computador.
- Tentar melhorar o update para ele não apagar os dados já alterados.
- Implementar uma função de busca por nome.

Na próxima aula iremos continuar melhorando esse projeto. Recomendo que criem uma cópia.