Engenharia - Tecnologias Web
Profs: Luciano Soares <lpsoares@insper.edu.br> e Camila Achutti <camilafa1@insper.edu.br>

HANDOUT AULA 2

Os objetivos de aprendizado dessa aula são:

- Acessar um banco de dados por uma aplicação web.
- Criar um CRUD (Create, Read, Update e Delete) web.

Resumo:

Nesta aula estaremos entrando no mundo dos bancos de dados SQL (Structured Query Language). O SQL pode ser considerada uma linguagem padrão de mercado para banco de dados relacionais. Neste primeiro contato não estaremos usando os comandos SQL diretamente, mas sim por uma técnica conhecida como ORM (Object-Relational Mapping), onde as registros dos dados serão mapeadas diretamente em objetos do programa em Python pela biblioteca Flask-Sqlalchemy.

1. CRIANDO BANCO DE DADOS

Os bancos de dados SQL precisam de algum tipo de serviço para funcionar. A vantagem de usar Python é que ele já vem com um suporte a SQL, no caso o servidor sqlite. Antes de tudo devemos definir onde estarão sendo armazenados os dados (no nosso exemplo estaremos usando um arquivo local mesmo) e associar com a aplicação Flask que ira usar os recursos.

Crie um arquivo [user.py] e insira as rotinas de inicialização do banco de dados. Perceba que o endereço informado no config, define o local e o nome do arquivo do banco de dados, que nesse caso será: database.db.

Não esqueçam de instalar o módulo SQLAlchemy:$ pip install flask-sqlalchemy

```
# -*- coding: utf-8 -*-
from flask import Flask, request, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)
```

Profs: Luciano Soares <lpsoares@insper.edu.br> e Camila Achutti <camilafa1@insper.edu.br>

2. Classes Mapeadas

Os bancos de dados em SQL são muito similares a tabelas. As colunas são os campos e as linhas são os dados. Essa nomenclatura de fato é usada. Para conseguirmos usar o banco de dados com os recursos de ORM, temos de criar as classes e seus atributos, que vão virar automaticamente os campos da tabela. Um ponto importante é criar uma chave primaria, que é uma coluna que será usada para uma identificação única do registro, ou seja, não pode repetir e nem ser vazio.

Insira no seu código uma classe chamada de *User* que servirá para gerenciar uma base de dados de usuários. Para ficar interessante vamos criar o campo de nome e e-mail por enquanto:

```
class User(db.Model):
      id = db.Column(db.Integer, primary_key=True)
      username = db.Column(db.String(80), unique=True)
      email = db.Column(db.String(120), unique=True)

      def __init__(self, username, email):
         self.username = username
         self.email = email
```

Para finalizar, as tabelas precisam ser criadas pelo sqlite. Assim chame a rotina para criar ela, a create_all(). Aproveitando, aqui está também a rotina para executar o Flask diretamente pelo Python:

```
db.create_all()

if __name__ == "__main__":
      app.run()
```

Lembre-se essas chamadas devem estar no final do seu programa.

Detalhes em: http://flask-sqlalchemy.pocoo.org/2.1/quickstart/

Profs: Luciano Soares <lpsoares@insper.edu.br> e Camila Achutti <camilafa1@insper.edu.br>

3. Criando dados

Já temos um programa, mas até o momento ele não faz muita coisa. Por exemplo se chamarmos o endereço de sempre: http://127.0.0.1:5000/ temos um erro! Para começar vamos fazer uma página para criar dados no nosso banco de dados.

Insira o código abaixo para criarmos um endereço que permita tal inclusão:

```
@app.route('/create', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        username = request.form["username"]
        email = request.form["email"]
        user = User(username=username, email=email)
        db.session.add(user)
        db.session.commit()
        return " dado inserido"
    return '''
        <form action="" method="post">
              <p>usuário: <input type=text name=username>
              <p>email: <input type=text name=email>
              <p><input type=submit value=Inserir>
        </form>
        '''
```

Mas e agora? Como saber se funcionou ou não? Uma dica aqui é usar um programa que permita bisbilhotar o que tem dentro do nosso arquivo de banco de dados. Existem inúmeras opções, aqui estão algumas delas:

- http://sqlitestudio.pl/
- http://sqlitebrowser.org/

  Para o sqlitebrowser para LINUX:

  sudo add-apt-repository -y ppa:linuxgndu/sqlitebrowser

  sudo apt update

sudo apt install sqlitebrowser

Tente agora colocar mais campos de dados para o usuário. Coloque campos de idade, altura, gênero.

Detalhes em: http://flask-sqlalchemy.pocoo.org/2.1/models/

4. Lendo Dados

Para ler os dados, precisamos primeiro encontrar eles no nosso banco de dados. Para isso existem as chamadas "queries". Existem diversas rotinas de busca, com os mais variados filtros, vamos usar um bem simples por enquanto. Insira o seguinte trecho no seu código:

```
@app.route('/read/<username>')
def read_user(username):
      user = User.query.filter_by(username=username).first()
      if(user):
        return user.username + " e-mail: &lt;" + user.email + "&gt;"
      else:
        return "Usuário não encontrado",404
```

Mas que diabos é esse < e >? Bem, experimente colocar os sinais < e > direto no seu código. O que acontece? Por que?

E esse 404 depois no return? Bem, se lembra dos códigos de erro? Então, esse 404 é o "Not found". Melhor avisar direitinho o navegador com esse segundo parâmetro que não deu muito certo a requisição do usuário.

Va lá, agora inclua a altura, idade e os outros campos que você criou. Quer deixar mais bonitinho? Use um template e um css.

5. Atualizando Dados

A rotina para atualizar dados é uma mistura das duas últimas. Temos de encontrar o dado e depois trocar os valores. A boa notícia é que com o ORM,

basta trocar o dado e fazer a chamada de commit e tudo deve funcionar sem maiores problemas. Insira o código abaixo:

```
@app.route('/update/<username>', methods=['GET', 'POST'])
def update_user(username):
   if request.method == 'POST':
      username = request.form["username"]
      email = request.form["email"]
      user = User.query.filter_by(username=username).first()
      user.email = email
      db.session.commit()
      return "Bem vindo"
   user = User.query.filter_by(username=username).first()
   return '''
      <form action="" method="post">
            <p>usuário: <input type=text name=username value={0}>
            <p>email: <input type=text name=email value={1}>
             <p><input type=submit value=Login>
      </form>
      '''.format(username,user.email)
```

Isso não esta ficando fácil. Colocar esse monte de código HTML dentro do programa Python está meio confuso. Remova o código HTML do Python e use como um template.

6. Removendo Dados

A rotina de remoção é muito parecida com a de update . O dado é encontrado, e então a rotina de exclusão é invocada. Insira o código abaixo e teste as funcionalidades.

```
@app.route('/delete/<username>')
def delete_user(username):
      user = User.query.filter_by(username=username).first()
      db.session.delete(user)
      db.session.commit()
      return "usuário removido"
```

Profs: Luciano Soares <lpsoares@insper.edu.br> e Camila Achutti <camilafa1@insper.edu.br>

Mais um ponto, esse código não faz nenhum tratamento para situações dos casos onde um erro pode acontecer. Faça um tratamento na rotina para que ela não corrompa o servidor.

7. Listando dados

As operações funcionam, porém muitas vezes não é prático sempre ficar lembrando de todos os usuários. Abaixo esta uma rotina que lista todos os usuários do banco de dados. Insira ela em seu código:

```
@app.route('/list')
def list_user():
      users = User.query.all()
      return render_template('list.html',users=users)
```

Como você pode perceber, ainda é necessário um template. Crie o diretório de templates se ainda não fez, e insira o arquivo html de template abaixo.

```
<style>
table, th, td {
      border: 1px solid black;
}
</style>
<table>
<tbody>
{% for row in users %}
      <tr>
      <td>{{ row.username }}</td>
      <td>{{ row.email }}</td>
      </tr>
{% endfor %}
</tbody>
</table>
```

Profs: Luciano Soares <lpsoares@insper.edu.br> e Camila Achutti <camilafa1@insper.edu.br>

8. Mais Funcionalidades

O sistema está funcionando. Inclua agora as seguintes funcionalidades:

- Adicione uma coluna na lista para remover diretamente um registro.
- Crie um campo de busca para encontrar um determinado registro.
- Calcule a idade média e altura média das pessoas da base de dados.
- Use as chamada REST para a PATCH e DELETE pra a atualização e remoção de registros.

9. Projeto

Comece a rascunhar como seu projeto pode ser implementado. Pense nas páginas que serão necessárias, como as informações estarão dispostas. Crie um backlog e verifique com os professores as funcionalidades desejadas para o sistema. Defina um sprint backlog e comece a implementar, nesse momento ainda estamos em só um aquecimento, assim não se prenda muito no que esta fazendo, se realmente quer reusar o que está fazendo agora, não esqueça de minimamente documentar o seu código. Senão vai acabar esquecendo do que fez.