

Aula 11

Programação Back-end | Formulários e Banco de Dados

Programação IV

Prof. Sandino Jardim | CC-UFMT-CUA



Formulários no Flask

- Extensão Flask-WTF
 - Permite a definição de formulários semelhante à definição de classes
- Instalação

(venv) \$ pip install flask-wtf



Configuração do Flask-WTF

- Aplicação deve definir uma chave secreta
 - Prevenção contra interferências externas

```
app = Flask(__name__)  
app.config['SECRET_KEY'] = 'hard to guess string'
```

- Definição de Formulário

```
from flask_wtf import FlaskForm  
from wtforms import StringField, SubmitField  
from wtforms.validators import DataRequired  
  
class NameForm(FlaskForm):  
    name = StringField('What is your name?', validators=[DataRequired()])  
    submit = SubmitField('Submit')
```



Tipos de Campos

Tipo de Campo	Descrição
BooleanField	Caixa de seleção com valores Verdadeiro e Falso
DateField	Campo de texto que aceita um valor datetime.date em um formato específico
DateTimeField	Campo de texto que aceita um valor datetime.datetime em um formato específico
DecimalField	Campo de texto que aceita um valor decimal.Decimal
FileField	Campo de upload de arquivo
HiddenField	Campo de texto oculto
MultipleFileField	Campo de upload de múltiplos arquivos
FieldList	Lista de campos de um tipo específico



Tipos de campos (cont.)

Tipo de Campo	Descrição
FloatField	Campo de texto que aceita um valor de ponto flutuante
FormField	Formulário incorporado como um campo em um formulário contêiner
IntegerField	Campo de texto que aceita um valor inteiro
PasswordField	Campo de texto para senha
RadioField	Lista de botões de rádio
SelectField	Lista suspensa de opções
SelectMultipleField	Lista suspensa de opções com seleção múltipla
SubmitField	Botão de submissão de formulário
StringField	Campo de texto
TextAreaField	Campo de texto de múltiplas linhas

Validadores

Validador	Descrição
DataRequired	Valida que o campo contém dados após a conversão de tipo
Email	Valida um endereço de email
EqualTo	Compara os valores de dois campos; útil ao solicitar que uma senha seja inserida duas vezes para confirmação
InputRequired	Valida que o campo contém dados antes da conversão de tipo
IPAddress	Valida um endereço de rede IPv4
Length	Valida o comprimento da string inserida
MacAddress	Valida um endereço MAC
NumberRange	Valida que o valor inserido está dentro de um intervalo numérico
Optional	Permite entrada vazia no campo, pulando validadores adicionais
Regex	Valida a entrada com base em uma expressão regular
URL	Valida uma URL
UUID	Valida um UUID
AnyOf	Valida que a entrada é uma entre uma lista de valores possíveis
NoneOf	Valida que a entrada não corresponde a nenhuma das opções em uma lista de valores

View function com formulário

```
@app.route('/', methods=['GET', 'POST'])
def index():
    name = None
    form = NameForm()
    if form.validate_on_submit():
        name = form.name.data
        form.name.data = ''
    return render_template('index.html', form=form, name=name)
```



Renderização do Formulário

```
{% extends "base.html" %}
{% import "bootstrap/wtf.html" as wtf %}

{% block title %}Flasky{% endblock %}

{% block page_content %}
<div class="page-header">
    <h1>Hello, {% if name %}{{ name }}{% else %}Stranger{% endif %}!</h1>
</div>
{{ wtf.quick_form(form) }}
{% endblock %}
```



Renderização do Formulário

```
{% extends "base.html"%}
{% import "bootstrap5/form.html" as wtf %}
{%block title%}Flask Formulários {%endblock%}
{%block page_content%}
<h1>Hello, {%if name %} {{name}} {%else%} Stranger {%endif%}</h1>
{{wtf.render_form(form)}}
{%endblock%}
```



Redirect

- Nosso exemplo solicita o reenvio dos dados submetidos a cada nova atualização da página
 - Pode ser prejudicial esse comportamento
- Solução:
 - Guardar dados enviados em cookie e evitar o reenvio e ressubmissão dos dados

```
from flask import Flask, render_template, session, redirect, url_for

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        session['name'] = form.name.data
        return redirect(url_for('index'))
    return render_template('index.html', form=form, name=session.get('name'))
```



Flash messages

```
from flask import Flask, render_template, session, redirect, url_for, flash

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        old_name = session.get('name')
        if old_name is not None and old_name != form.name.data:
            flash('Looks like you have changed your name!')
        session['name'] = form.name.data
        return redirect(url_for('index'))
    return render_template('index.html',
        form = form, name = session.get('name'))
```

Renderizando Flash Messages

```
{% block content %}
<div class="container">
  {% for message in get_flashed_messages() %}
    <div class="alert alert-warning">
      <button type="button" class="close" data-dismiss="alert">&times;</button>
      {{ message }}
    </div>
  {% endfor %}

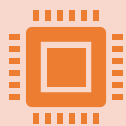
  {% block page_content %}{% endblock %}
</div>
{% endblock %}
```

Renderizando Flash Messages

```
{% for message in get_flashed_messages() %}
<div class="alert alert-primary d-flex align-items-center alert-dismissible fade
show" role="alert">
  {{ message }}
  <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
  label="Close"></button>
</div>
{% endfor %}
```



Frameworks Python para BD



Python possui bibliotecas para a maioria dos bancos de dados, seja código aberto ou comerciais



Flask não coloca restrições na escolha do BD a ser utilizado



Também permite acessar bibliotecas com abstração de bancos de dados que eliminam necessidade de conhecimento de SQL

Flask-SQLAlchemy

- SQLAlchemy
 - Framework de banco de dados relacional que suporta vários tipos de BD
- Flask-SQLAlchemy realiza a integração do Flask com essa ferramenta
- Instalação

```
(venv) $ pip install flask-sqlalchemy
```



Configuração

```
import os
from flask_sqlalchemy import SQLAlchemy

basedir = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:/// ' + os.path.join(basedir, 'data.sqlite')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
```



Definição do Modelo

- A biblioteca Flask-SQLAlchemy provê uma classe base para definição de modelos

```
class Role(db.Model):
    __tablename__ = 'roles'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)
    users = db.relationship('User', backref='role')

    def __repr__(self):
        return '<Role %r>' % self.name

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True, index=True)
    role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))

    def __repr__(self):
        return '<User %r>' % self.username
```



Integração do Formulário com o BD

```
@app.route('/', methods=['GET', 'POST'])
def index():
    db.create_all()
    form = NameForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.name.data).first()
        if user is None:
            user = User(username=form.name.data, role_id=0)
            db.session.add(user)
            db.session.commit()
            session['known'] = False
            session['name'] = form.name.data
        else:
            session['known'] = True
            session['name'] = form.name.data
            form.name.data = ''
        return redirect(url_for('index'))
    return render_template('index_db.html', form=form,
name=session.get('name'), known=session.get('known', False))
```

app.py

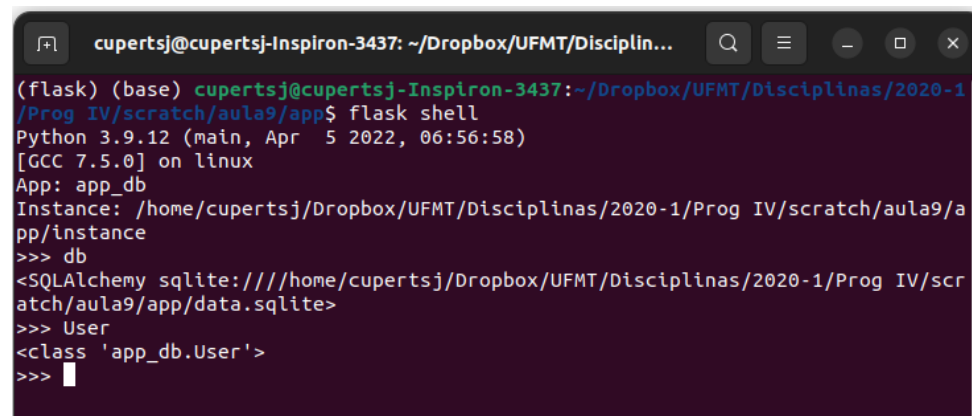
```
{% extends "base.html"%}
{% import "bootstrap5/form.html" as wtf %}
{% block title%}Flask Formulários {% endblock%}
{% block page_content%}
<h1>Hello, {% if name %} {{name}} {% else%} Stranger
{% endif%}</h1>
{% if not known%}
<p>Prazer em te conhecer!</p>
{% else%}
<p>Que bom te ver de novo!</p>
{% endif%}
{{ wtf.render_form(form) }}
{% endblock%}
```

index.html



Acesso ao BD criado via shell

```
@app.shell_context_processor
def make_shell_context():
    return dict(db=db, User=User, Role=Role)
```

A terminal window with a dark background and light-colored text. The window title is 'cupertsj@cupertsj-Inspiron-3437: ~/Dropbox/UFMT/Disziplin...'. The prompt is '(flask) (base) cupertsj@cupertsj-Inspiron-3437:~/Dropbox/UFMT/Disiplinas/2020-1/Prog IV/scratch/aula9/app\$'. The user enters 'flask shell'. The output shows the Python version 'Python 3.9.12 (main, Apr 5 2022, 06:56:58)', the GCC version '[GCC 7.5.0] on linux', the application name 'App: app_db', and the instance path 'Instance: /home/cupertsj/Dropbox/UFMT/Disiplinas/2020-1/Prog IV/scratch/aula9/app/instance'. The user then enters '>>> db', and the output is '<SQLAlchemy sqlite:///home/cupertsj/Dropbox/UFMT/Disiplinas/2020-1/Prog IV/scratch/aula9/app/data.sqlite>'. The user enters '>>> User', and the output is '<class \'app_db.User\'>'. The user enters '>>>' and the prompt returns. On the right side of the terminal window, there is a vertical watermark text 'CC-UFMT-004'.

```
cupertsj@cupertsj-Inspiron-3437: ~/Dropbox/UFMT/Disziplin...
(flask) (base) cupertsj@cupertsj-Inspiron-3437:~/Dropbox/UFMT/Disiplinas/2020-1/Prog IV/scratch/aula9/app$ flask shell
Python 3.9.12 (main, Apr 5 2022, 06:56:58)
[GCC 7.5.0] on linux
App: app_db
Instance: /home/cupertsj/Dropbox/UFMT/Disiplinas/2020-1/Prog IV/scratch/aula9/app/instance
>>> db
<SQLAlchemy sqlite:///home/cupertsj/Dropbox/UFMT/Disiplinas/2020-1/Prog IV/scratch/aula9/app/data.sqlite>
>>> User
<class 'app_db.User'>
>>>
```

'flask shell' acessa e visita tabelas criadas via aplicação py



Migração de banco de dados

- Instalar biblioteca

```
pip install flask-migrate
```

- Realizar mudança na tabela

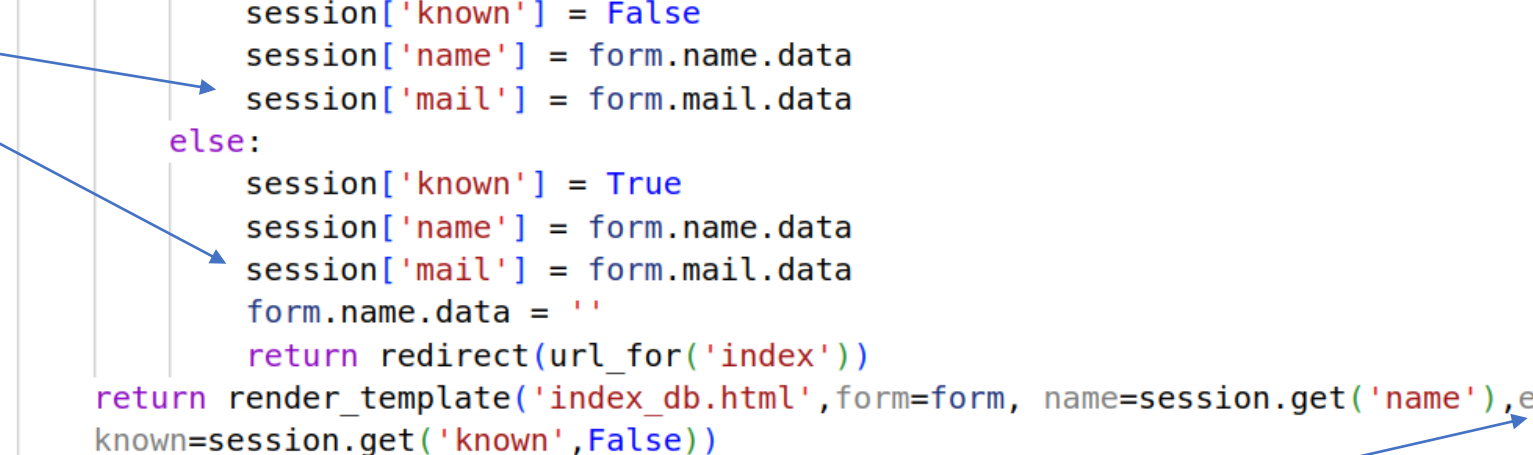
```
class User(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    username = db.Column(db.String(64), unique=True, index=True)  
    role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))  
    email = db.Column(db.String(64), index=True)
```

- Inicializar BD: `flask db init`
- Commit da atualização: `flask db migrate -m "coluna mail"`
- Finalizar: `flask db upgrade`



Adicione novo campo para testar nova coluna

```
@app.route('/', methods=['GET', 'POST'])
def index():
    db.create_all()
    form = NameForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.name.data).first()
        if user is None:
            user = User(username=form.name.data, role_id=0, email=form.mail.data)
            db.session.add(user)
            db.session.commit()
            session['known'] = False
            session['name'] = form.name.data
            session['mail'] = form.mail.data
        else:
            session['known'] = True
            session['name'] = form.name.data
            session['mail'] = form.mail.data
            form.name.data = ''
        return redirect(url_for('index'))
    return render_template('index_db.html', form=form, name=session.get('name'), email=session.get('mail'),
                           known=session.get('known', False))
```



The diagram illustrates the data flow in the provided code. Two blue arrows originate from the left side of the slide. The first arrow points to the `session['name'] = form.name.data` line in the `if` block, indicating that the name from the form is stored in the session. The second arrow points to the `session['mail'] = form.mail.data` line in the same block, indicating that the email from the form is also stored in the session. A third blue arrow originates from the bottom of the slide and points to the `name=session.get('name')` argument in the `render_template` call, showing how the name is retrieved from the session and passed to the template.