

Flask

Desenvolvendo aplicações Web com o Framework Flask - Aula 11

Formulários Web com Flask-WTF

Os *templates* que criamos até agora são unidirecionais, ou seja, permitem que as informações fluam do servidor para o usuário.



Porém, muitas vezes precisaremos fazer o caminho oposto, onde os usuários possam fornecer dados para que o servidor aceite e processe.



Podemos criar formulários *HTML* e submeter o conteúdo destes formulários pelo navegador ao servidor na forma de uma requisição *POST*. O objeto de requisição do *Flask*, expõe as informações enviadas pelo cliente em uma requisição, e, particularmente, para *requisições POST* contendo dados de formulário, o *Flask* nos dá acesso a estas informações por meio de *request.form*.

Embora o suporte oferecido pelo objeto de requisição do *Flask* seja suficiente para tratar formulários *web*, há uma série de tarefas que podem se tornar cansativas e repetitivas, como por exemplo, a geração de código *HTML* e a validação dos dados dos formulários.

A extensão *Flask-WTF* faz com que nosso trabalho com os formulários *web* seja muito mais agradável.

Essa extensão é um pacote para integração com o *Flask* em torno do pacote *WTForms*, que é independente de *framework*.

Podemos instalar o *Flask-WTF* com o *pip*:

```
$ pip install flask-wtf
```

Fugindo à regra de extensões *Flask*, o *Flask-WTF* não precisa ser inicializado no nível da aplicação, mas espera que a aplicação tenha uma chave secreta configurada.

Uma chave secreta é uma *string* utilizada como uma chave de criptografia ou assinatura para aumentar a segurança da aplicação.

Essa *string* pode ser qualquer valor aleatório e único, não pode ser conhecida por ninguém e deve ser diferente para cada aplicação que for criada.

Veja como podemos criar uma chave secreta para nossa aplicação.

```
app = Flask(__name__)  
app.config['SECRET_KEY'] = 'string_chave_secreta'
```

O dicionário *app.config* é um local de propósito geral para armazenar variáveis de configuração usadas pelo *Flask*, pelas extensões ou pela própria aplicação.

Podemos usar a sintaxe padrão de dicionários para adicionar valores de configuração ao objeto *app.config*.

O objeto *app.config* também tem métodos para importar os valores de configuração de arquivos ou do ambiente.

A chave secreta é usada pelo *Flask-WTF* para proteger os formulários contra ataques de *CSRF* (*Cross-Site Request Forgery*). Um ataque de *CSRF* ocorre quando um site malicioso envia requisições para o servidor da aplicação no qual o usuário está logado no momento. O *Flask-WTF* gera *tokens* de segurança para todos os formulários e os armazena na sessão do usuário, que é protegida com uma assinatura criptográfica gerada a partir da chave secreta.

Flask-WTF - Classes de formulário

Quando usamos o *Flask-WTF*, cada formulário *web* é representado no servidor por uma classe que herda da classe *FlaskForm* e, esta, por sua vez, define a lista de campos do formulário, cada um representado por um objeto.

Cada objeto que representa um campo pode ter um ou mais validadores associados. Um validador é uma função que verifica se os dados submetidos pelo usuário são válidos.

Campos *HTML* padrões aceitos pelo *WTForms*:

Tipo	Descrição	Tipo	Descrição
BooleanField	Caixa de seleção com valores True e False	PasswordField	Campo de texto para senha
DateField	Campo de texto que aceita um valor datetime.date em um dado formato	IntegerField	Campo de texto que aceita um valor inteiro
DateTimeField	Campo de texto que aceita um valor datetime.datetime em um dado formato	FormField	Formulário incluído como um campo em um formulário contêiner
DecimalField	Campo de texto que aceita um valor decimal.Decimal	RadioField	Lista de botões de rádio
FileField	Campo para upload de arquivo	SelectField	Lista suspensa de opções
HiddenField	Campo de texto oculto	SelectMultipleField	Lista suspensa de opções com múltipla seleção
MultipleFileField	Campo para upload de vários arquivos	SubmitField	Botão para submissão de formulário
FieldList	Lista de campos de um dado tipo	StringField	Campo de texto
FloatField	Campo de texto que aceita um valor de ponto flutuante	TextAreaField	Campo de texto multilinha

Flask-WTF - Classes de formulário

Validadores do *WTForms*:

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

Vamos criar agora um exemplo de uso do Flask-WTF. Vamos começar com o arquivo “app.py”. Os exemplos completos são encontrados nos recursos desta aula e em nosso repositório no GitHub.

Flask-WTF - Tratamento de formulários em funções de view

Primeiro vamos acrescentar uma rota “/formulario”. Adicionamos o argumento *methods* ao decorador `app.route` para registrar a função de *view* como um *handler* para requisições *GET* e *POST* no mapa de *URLs*. Quando não usamos *methods*, a função de *view* é registrada para tratar somente requisições *GET*. O método *POST* é necessário, pois as submissões de formulário são tratadas de modo muito mais conveniente. É possível submeter um formulário como uma requisição *GET*, mas como essas requisições não têm um corpo, os dados serão concatenados ao *URL* na forma de uma string de consulta e se tornarão visíveis na barra de endereço do navegador. Por esse e vários outros motivos, as submissões de formulário são quase universalmente efetuadas como requisições *POST*.

```
@app.route('/formulario', methods=['GET', 'POST'])
def formulario():
    ...
```


Flask-WTF - Tratamento de formulários em funções de view

A variável local “nome” vai armazenar o nome recebido do formulário se este for enviado. Quando a página é aberta pela primeira vez, será executado o método GET e não será fornecido um nome, desta forma o conteúdo da variável “nome” será None.

```
@app.route('/formulario', methods=[ 'GET', 'POST' ])
def formulario():
    nome = None
```

Flask-WTF - Tratamento de formulários em funções de view

A variável `form` é uma instância da classe `NomeForm` que vamos criar logo a seguir. O método `validate_on_submit` retorna `True` se o formulário for submetido e os dados forem aceitos por todos os validadores de campo. A primeira vez que executarmos a view, o servidor receberá uma requisição `GET` sem dados do formulário, desta forma `validate_on_submit()` retornará `False`. O template será renderizado, porém, a variável “nome” será `None`.

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

Flask-WTF - Classes de formulário

Quando o usuário digitar um valor no campo “nome”, e clicar no botão “Enviar”, o servidor receberá uma requisição POST com os dados e `validate_on_submit()` chama o validador `DataRequired()` associado ao campo de nome. Se o nome não for vazio, o validador aceitará e `validate_on_submit()` devolverá `True`.

Ainda no `app.py` vamos criar a classe `NomeForm` (nossa classe do formulário) que herda de `FlaskForm` e possui dois objetos, sendo “nome”, um `StringField`, que é um campo de texto e `submit`, um `SubmitField`, que é um botão de envio de dados.

O primeiro argumento para os construtores é o rótulo que será utilizado na renderização do formulário para HTML.

```
class NomeForm(FlaskForm):  
    nome = StringField('Qual seu nome?', validators=[DataRequired()])  
    submit = SubmitField('Enviar')
```

Flask

Flask-WTF – Arquivo app.py completo

```
from flask import Flask, render_template
from flask_bootstrap import Bootstrap
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired
from flask_wtf import FlaskForm

app = Flask(__name__)
app.config['SECRET_KEY'] = 'string_chave_secreta'
bootstrap = Bootstrap(app)

@app.route('/', methods=['GET', 'POST'])
def principal():
    return render_template('principal.html')
...
```

```
...
@app.route('/formulario', methods=['GET', 'POST'])
def formulario():
    nome = None
    form = NomeForm()
    if form.validate_on_submit():
        nome = form.nome.data
        form.nome.data = ""
    return render_template('formulario.html', form=form, nome=nome)

class NomeForm(FlaskForm):
    nome = StringField('Qual seu nome?', validators=[DataRequired()])
    submit = SubmitField('Enviar')

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



Flask-WTF - Renderização de HTML em formulários

Agora vamos criar o template “formulario.html”.

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

{% block title %}Aula Flask{% endblock %}

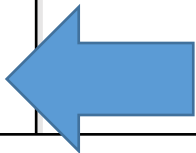
{% block page_content %}
<div class="page-header">
    <h1>Olá, {% if nome %}{{ nome }}{% else %}Estranho{% endif %}!</h1>
</div>
{{ wtf.quick_form(form) }}
{% endblock %}
```

Flask-WTF - Renderização de HTML em formulários

Vamos criar também o arquivo base.html como segue.

```
{% extends "bootstrap/base.html" %}
{% block title %}Aula Flask{% endblock %}
{% block navbar %}

<div class="navbar navbar-inverse" role="navigation">
  <div class="container">
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a href="/">Principal</a></li>
        <li><a href="/formulario">Formulário</a></li>
      </ul>
    </div>
  </div>
</div>
{% endblock %}
...
```



```
...
{% block content %}
  <div class="container">
    {% block page_content %}{% endblock %}
  </div>
{% endblock %}
```

Flask-WTF - Renderização de HTML em formulários

Vamos criar também o arquivo principal.html como segue.

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

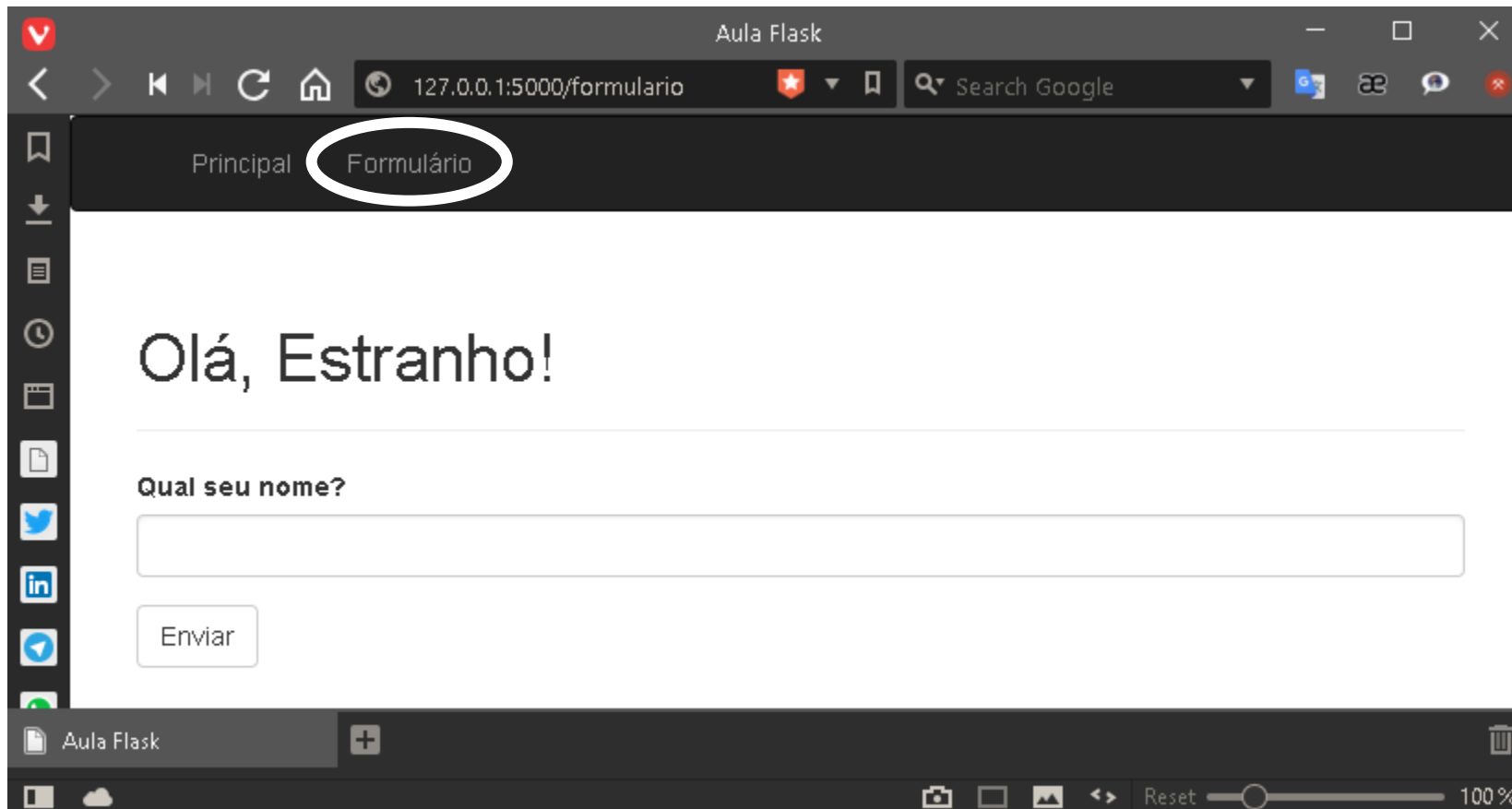
{% block title %}Aula Flask{% endblock %}

{% block page_content %}
<div class="page-header">
  <h1>Página principal</h1>
</div>
{% endblock %}
```

Flask

Flask-WTF - Executando o exemplo

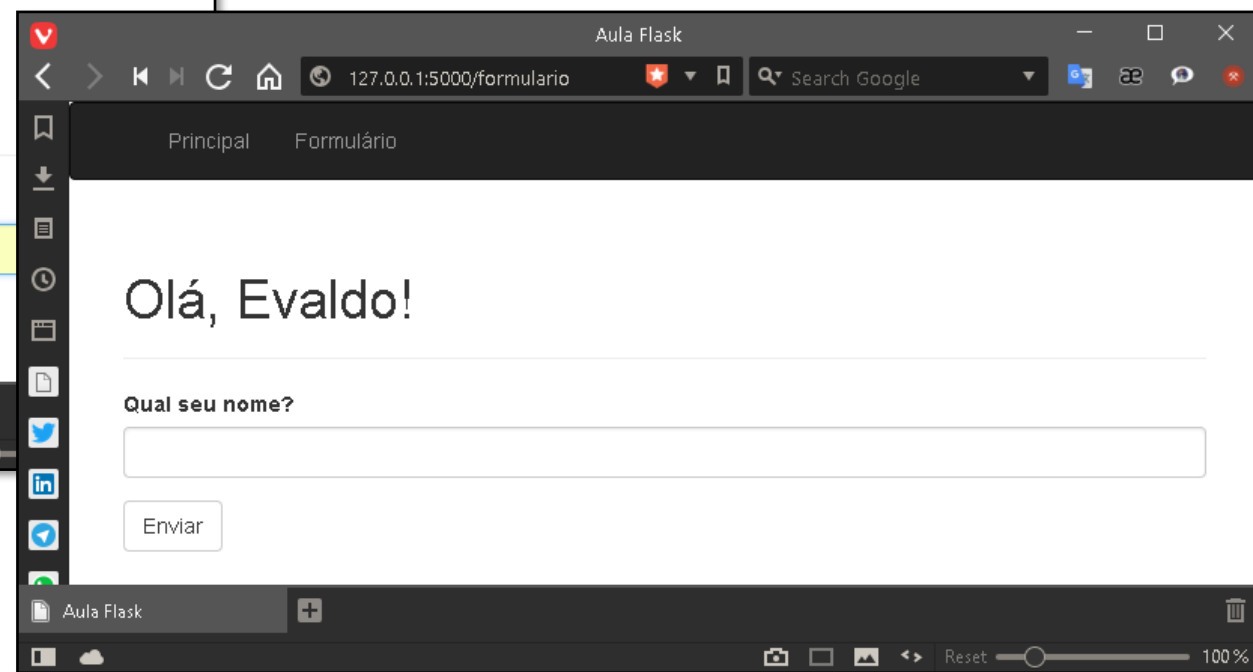
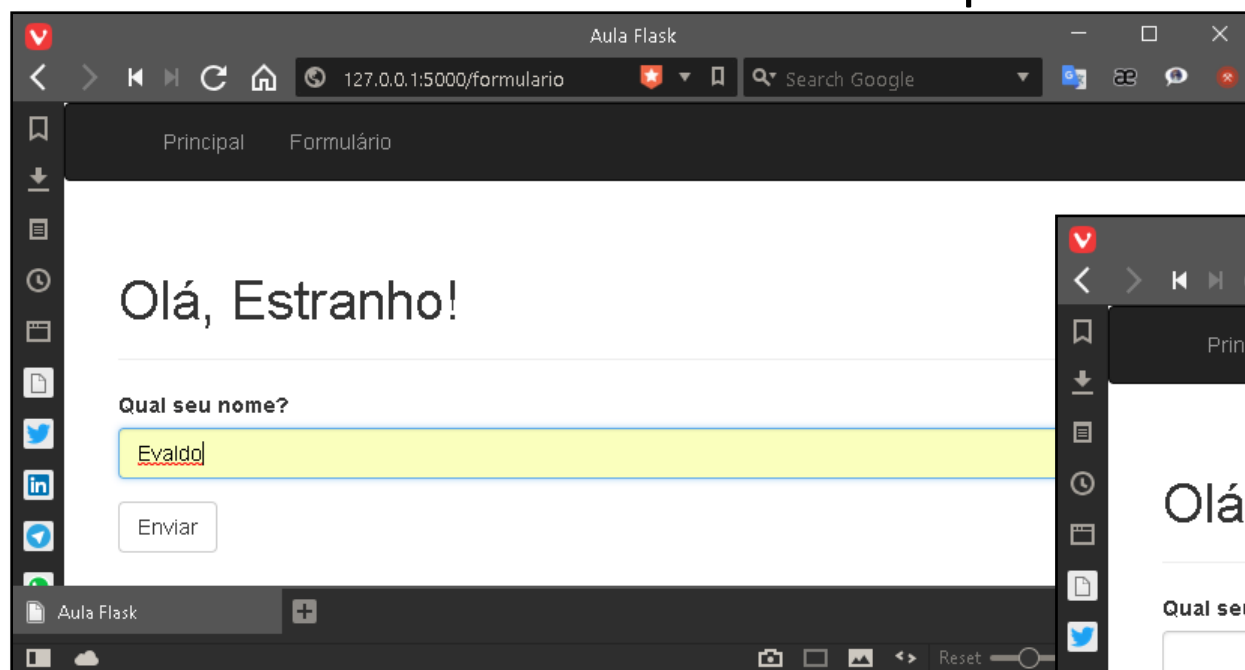
Execute o programa e clique no link Fomulário para chamar a view pela primeira vez.



Flask

Flask-WTF - Executando o exemplo

Enviando o formulário com o nome preenchido.



Flask-WTF - Classes de formulário

Explicando algumas partes do formulario.html

```
# estendendo nosso modelo criado anteriormente.
{% extends "base.html" %}

# importando o estilo de formulário predefinido
# do Bootstrap para o Flask-WTF
{% import "bootstrap/wtf.html" as wtf %}

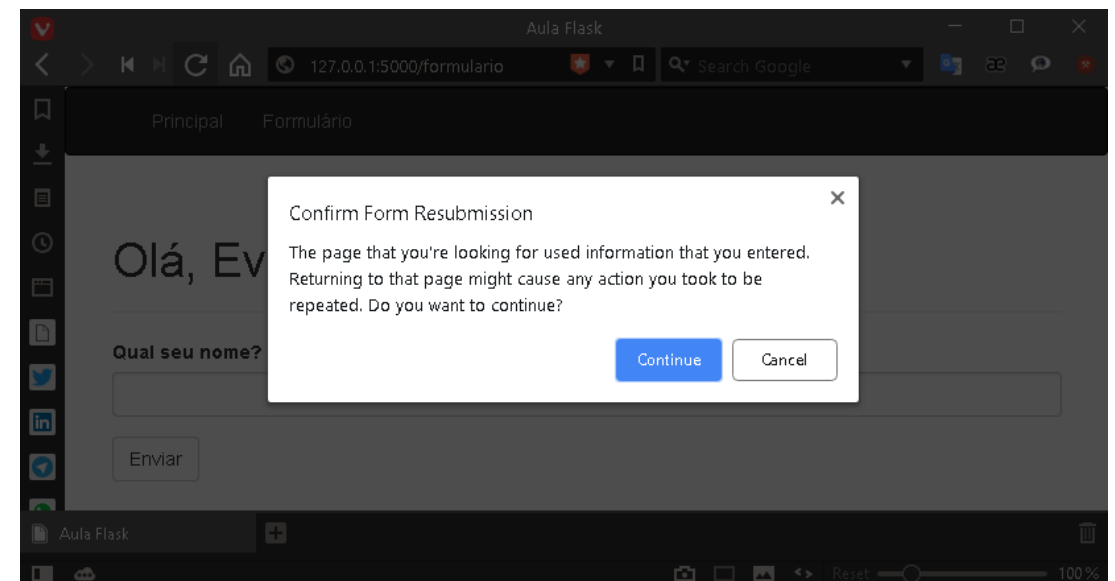
# Aqui estamos usando a estrutura condicional IF do Jinja2, que é:
# {% if condição %}FAÇA ISSO{% else %}FAÇA AQUILO{% endif %}
<h1>Olá, {% if nome %}{{ nome }}{% else %}Estranho{% endif %}!</h1>

# Aqui estamos executando uma função que recebe um objeto de
# formulário do Flask-WTF e renderiza o mesmo usando estilos
# padrões do Flask-Bootstrap.
{{ wtf.quick_form(formulario) }}
{% endblock %}
```

Flask-WTF - Redirecionamentos e sessões de usuário

Após submetermos o formulário com o campo “nome” preenchido, se tentarmos atualizar a página do navegador (usando o botão “refresh”), o mesmo emitirá uma solicitação de confirmação de submissão do formulário, ou seja, se você confirmar, o formulário será submetido novamente. Isso é ruim, porque pode fazer com que sejam realizados registros duplicados em caso de um formulário de cadastro por exemplo.

Podemos resolver este problema respondendo à requisição POST com um redirecionamento em vez de uma resposta usual. Um redirecionamento é um tipo especial de resposta que contém um URL em vez de uma string com código HTML.



Flask-WTF - Redirecionamentos e sessões de usuário

Quando o navegador recebe uma resposta de redirecionamento, ele gera uma requisição GET para o URL de redirecionamento, e essa é a página que será exibida.

A página poderá demorar alguns milissegundos a mais para carregar por causa da segunda requisição que deve ser enviada ao servidor, mas, exceto por isso, o usuário não verá nenhuma diferença.

Sendo a última requisição um GET, o comando de atualização da página funcionará conforme esperado. Esse truque é conhecido como padrão Post/Redirect/Get.

Porém, ao redirecionarmos a requisição perderemos o nome do usuário enviado no formulário, desta forma, precisaremos usar variáveis de sessão para mantermos esta informação disponível.

Flask

Flask-WTF - Redirecionamentos e sessões de usuário

Nosso próximo passo é importar *session*, *redirect* e *url_for*:

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

Vamos agora modificar a rota “/formulario” no arquivo app.py para utilizarmos *redirect*, *url_for* e *session*.

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

Após esta alteração, ao realizarmos um refresh na página do browser, não será enviado para o servidor os dados do formulário novamente. Se você realizar um refresh da página, não haverá uma nova requisição.



Flask-WTF - Apresentação de flash message

Flash message são mensagens que podem ser exibidas temporariamente com uma informação para o usuário. Por exemplo, quando tentamos realizar login no sistema com dados inválidos.

A mensagem é exibida e descartada posteriormente.

Vamos começar adicionando o import para flash no nosso arquivo app.py.

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

Flask-WTF - Apresentação de flash message

Agora vamos mudar nosso método formulario().

```
@app.route('/formulario', methods=['GET', 'POST'])
def formulario():
    form = NomeForm()
    if form.validate_on_submit():
        nome_anterior = session.get('nome')
        if nome_anterior is not None and nome_anterior != form.nome.data:
            flash(f'Parece que você alterou seu nome de {nome_anterior} para {form.nome.data}!')
        session['nome'] = form.nome.data
        return redirect(url_for('formulario'))
    return render_template('formulario.html', form=form, nome=session.get('nome'))
```

Flask-WTF - Apresentação de flash message

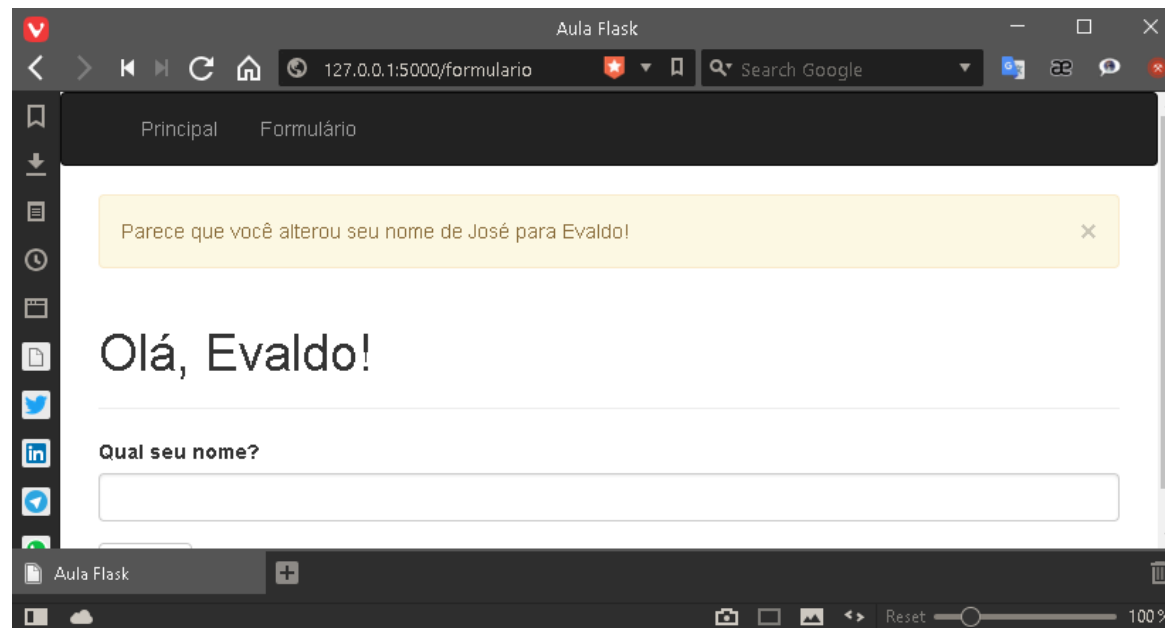
Precisamos agora definir em qual local da página a flash message será exibida. Vamos alterar a div container do bloco content do arquivo base.html para isso.

```
{% 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 %}
```


Flask

Flask-WTF - Apresentação de flash message

Execute o programa, informe um texto para o campo nome, execute o formulário e veja o resultado. Mude o texto e informe um nome diferente, em seguida execute o formulário. Veja a mensagem sendo exibida.



FIM

