



UNIVERSIDADE
FEDERAL DO PIAUÍ

Universidade Federal do Piauí

Disciplina: Engenharia de Software II

Professor: Armando Soares Sousa

Projeto: Microgram Equipe:

- **Carlos Eduardo Mendes de Oliveira.**
- **José Pires Gayoso**
- **Fernando Vieira Rosa**
- **Markesley Ramos**
- **Bruno Estrella**

Tutorial 5

Tutorial de como fazer upload de arquivos para a aplicação web

1) Introdução

Neste tutorial, com fins de facilitar o entendimento e o aprendizado, criaremos uma aplicação web simples que apresentará todas as tecnologias necessárias para a realização de uploads de arquivos. Essa aplicação web será composta por duas rotas, uma sendo responsável por realizar o upload de arquivos e a outra por listar todos os arquivos salvos.

Para este tutorial, será utilizado alguns comandos e métodos do microframework Flask (recomenda-se visualizar o tutorial 1 para aprender sobre sua instalação) como `render_template`, `redirect`, `url_for`, `request` e `flash` e também alguns comandos da biblioteca OS, como `os.path.join`, `os.path.basename` e `os.path.splitext` para facilitar a realização de operações utilizando comandos do Sistema Operacional. Por fim, utilizou-se do framework de código-fonte aberto Bootstrap para realizar a estilização e customização da aplicação

2) Instalação

- Não será retratado a instalação do Flask nesse tutorial, então recomenda-se, caso necessário, a visualização do tutorial 1.
- A biblioteca OS do Python não requer instalação prévia, podendo ser importada diretamente no código-fonte da aplicação.
- O framework Bootstrap 5, por ser de código aberto, não necessita de instalação. Sua importação ocorre diretamente no template html.

3) Configuração necessárias iniciais

- Primeiramente, cria-se um arquivo `.py` chamado `'config'`. Neste arquivo, estarão presentes todas as configurações necessárias para a realização do upload de arquivos na aplicação. Dentre todas as configurações estão:
 - **UPLOAD_FOLDER** = Variável que apresenta o caminho do diretório responsável por armazenar todas os arquivos que serão feitos upload na aplicação web
 - **SECRET_KEY** = Chave de configuração que será responsável por conter o pedaço de informação que controla as operações de um algoritmo de criptografia. Pode ser gerada aleatoriamente no terminal usando o seguinte passo a passo de comando:
 - 1º: Abra o cmd
 - 2º: Digite python
 - 3º: Digite import os
 - 4º: Digite print(os.urandom(16))
 - 5º: Copie e cole a impressão na variável que será responsável por armazenar a chave.
 - **MAX_CONTENT_LENGTH** = Chave de configuração responsável por limitar o tamanho máximo de memória de um arquivo que sofrerá upload. Está no formato `"x" * "y" * "z"`. No caso apresentado abaixo, o limite de memória do arquivo é de 16 megabytes

```

UPLOAD_FOLDER = 'static/uploads'

SECRET_KEY = "b'\x8d\xc7\x902\xce\x1cf\x07\x9f\x151-\xe0\xa02\xf7'"

MAX_CONTENT_LENGTH = 16 * 1024 * 1024

```

Figura 1: Definição das configurações iniciais da aplicação.

- Crie seu objeto de aplicativo Flask. Nele, irão estar presentes todas as configurações necessárias para inicialização da aplicação Web, assim como todas as rotas e métodos necessários para a implementação da aplicação.
- Dentro do arquivo criado anteriormente, importe o framework flask e todos os comandos que serão utilizados na sua implementação, além da biblioteca OS.

```

from flask import Flask, request, redirect, url_for, render_template, flash
import os

```

- Inicialize a aplicação flask com Flask(__name__), configure-o para adotar as configurações definidas no arquivo “config.py” com o método config.from_object(“config.py”) e crie uma variável do tipo ALLOW_EXTENSIONS que irá armazenar todos os formatos de arquivos que serão aceitos na aplicação.

```

app = Flask(__name__)

app.config.from_object('config')

TIPOS_DISPONIVEIS = set(['txt', 'pdf', 'doc', 'docx', 'png', 'jpg', 'jpeg', 'gif', 'csv', 'xlsx'])

```

- Por fim, crie uma função que analisará todos os formatos de arquivos que serão feitos upload. Caso o formato do arquivo não esteja contido na variável definida previamente, irá ser retornada uma expressão booleana do tipo FALSE. Caso contrário, a expressão booleana será TRUE.

```

def arquivos_permitidos(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in TIPOS_DISPONIVEIS

```

4) Definição das Rotas e Funções

- Antes de definir as rotas e as funções da aplicação, crie um novo diretório no seu projeto com o nome de “templates”. Esse diretório será responsável por armazenar todos os arquivos .html que servirão de modeladores para as páginas da sua aplicação.
- Crie a rota principal da sua aplicação por meio do comando @”nome_aplicativo”.route(’/nome_rota’). Esta será a sua rota principal da aplicação.
- Crie um novo arquivo .html no diretório templates. Este arquivo modelará a página de abertura da sua aplicação

```
@app.route('/')
def home():
    return render_template('index.html')
```

- Nesta rota, é definida uma função home() que servirá apenas para expor a página inicial de abertura da aplicação. Note que esta função retorna o método render_template seguido de um nome de arquivo .html. Isso quer dizer que a função está pedindo para q seja renderizada uma página seguindo a estilização definida neste arquivo.
- Para esta mesma rota, defina os métodos HTTPs que serão utilizados. Para o upload de arquivos, apenas será utilizado o método POST. Esse método é inserido como parâmetro da rota.

```
@app.route('/', methods=["POST"])
```

- No arquivo .html criado, implemente a criação do formulário para realizar upload de imagens na aplicação por meio da tag <form>. Essa tag recebe como parâmetros o método HTTP utilizado, o modo de criptografia e a ação a ser realizada com o preenchimento deste formulário. Esta ação(action) redireciona os dados inseridos pelo usuário para a função de upload de imagem (função que será criada a seguir)

```
<form method="POST" enctype="multipart/form-data" action="{{url_for('upload_image')}}">
    <div class="input-group">
        <input type="file" class="form-control" aria-label="Upload" name="file">
        <input class="btn btn-outline-secondary"
            type="submit" id="inputGroupFileAddon04" value="Realizar Upload">
    </div>
</form>
```

Inserção do Formulário

- Defina a função de upload. Esta função deve receber os dados fornecidos pelo usuário na página da aplicação. Estes dados devem ser analisados pela função. Primeiro, analisa-se se o usuário enviou ou não os dados de um arquivo. Caso não, é enviado uma mensagem flash ao .html avisando-o de que nenhum arquivo foi selecionado pelo usuário. Por fim, a função analisa se o formato do arquivo é aceito pela aplicação. Caso não, é enviado uma mensagem flash ao .html avisando o usuário de que o arquivo enviado não era compatível com os tipos de formatos de arquivos permitidos pela aplicação. Em seguida, caso o arquivo passe por essas duas etapas, o arquivo é salvo no diretório definido anteriormente(UPLOAD_FOLDER) e seu caminho é redirecionado ao arquivo .html.

```

@app.route('/', methods=["POST"])
def upload_image():
    file = request.files['file']
    if file.filename == '':
        flash("Nenhum arquivo foi selecionado", category="file_error")
        return redirect(request.url)

    if not arquivos_permitidos(file.filename):
        flash("Utilize um tipo de arquivo permitido!", category="compatibility_error")
        return redirect(request.url)

    file.save(os.path.join(app.config['UPLOAD_FOLDER'], file.filename))
    filename = os.path.basename(os.path.join(app.config['UPLOAD_FOLDER'], file.filename))
    flash("Imagem Enviada com Sucesso", category="success")
    return render_template("index.html", filename=filename)

```

Função que armazena no diretório os arquivos fornecidos pelo usuário na aplicação

- No arquivo .html, defina o formato de exposição das mensagens flash que são enviadas ao html caso seja detectado algum erro.

```

{% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
        <ul>
            {% for category, message in messages %}
                {% if category == "file_error" %}
                    <div class="alert alert-danger">
                        <div class="bi bi-info-circle-fill">
                            {{message}}
                        </div>
                    </div>
                {% elif category == "compatibility_error" %}
                    <div class="alert alert-danger">
                        <div class="bi bi-info-circle-fill">
                            {{message}}
                        </div>
                    </div>
                {% else %}
                    <div class="alert alert-success">
                        <div class="bi bi-info-circle-fill">
                            {{message}}
                        </div>
                    </div>
                {% endif %}
            {% endfor %}
        </ul>
    {% endif %}
{% endwith %}

```

- Defina a função de display da imagem do arquivo. Sempre que o arquivo for uma imagem, assim que é feito o upload dele, logo em seguida é exposta a imagem deste arquivo na página. Ele recebe como parâmetro da rota o nome do arquivo fornecido pelo usuário e redireciona ao .html

```
@app.route('/display/<filename>')
def display_image(filename):
    return redirect(url_for('static', filename='uploads/' + filename), code=301)
```

Função de expor a imagem do arquivo na página WEB assim que for feito o upload do mesmo

```
{% if filename %}
    <div>
        
    </div>
{% endif %}
<br>
<div class="container" align="center">
    <a href="{{url_for('list_images')}}" class="btn btn-secondary" >Listar Arquivos Salvos</a>
</div>
</div>
```

Função HTML de exposição da imagem na tela

- Página HTML completa com alguns outros elementos:

Upload de Arquivos com Flask

Formato de Arquivos Permitidos

TXT

PDF

DOC/DOCX

PNG/JPG/JPEG

GIF

CSV

XLSX

Escolher arquivo

Nenhum arquivo escolhido

Realizar Upload

Listar Arquivos Salvos

Página HTML antes do upload do arquivo

Formato de Arquivos Permitidos

TXT

PDF

DOC/DOCX

PNG/JPG/JPEG

GIF

CSV

XLSX

Escolher arquivo

natureza.jpg

Realizar Upload

Listar Arquivos Salvos


Seleção do arquivo para realizar Upload

Upload de Arquivos com Flask

Imagem Enviada com Sucesso

Formato de Arquivos Permitidos

Escolher arquivo Nenhum arquivo escolhido Realizar Upload



Listar Arquivos Salvos

Página HTML após o upload do arquivo

Upload de Arquivos com Flask

Utilize um tipo de arquivo permitido!

Formato de Arquivos Permitidos

Escolher arquivo Nenhum arquivo escolhido Realizar Upload

Listar Arquivos Salvos

Exemplo de Mensagem Flash

- Agora, após definir a rota para o upload de arquivos, defina a rota para a listagem de todos os arquivos que sofreram upload.
- Defina a função para listar todos os arquivos e imagens que receberam upload
- Crie um novo arquivo .html responsável por listar todos esses arquivos
- Criar elementos que interliguem as duas páginas (como botões de redirecionamento)

```
@app.route('/images_list')
def list_images():
    images = []
    for filename in os.listdir(app.config['UPLOAD_FOLDER']):
        image = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        images.append({"path": image, "basename": os.path.basename(image),
                       "format": os.path.splitext(image)[1]})
    return render_template("list.html", images=images)
```

Função que busca todos os arquivos armazenados no diretório e repassa seus dados(caminho completo da imagem, nome do arquivo e formato da imagem) ao arquivo .html

```

{% for image in images %}
    <div class="card mb-3" style="max-width: 540px;">
        <div class="row g-0">
            {% if image.format in ['.png', '.jpg', '.jpeg', '.gif'] %}
                <div class="col-md-4">
                    
                </div>
            {% else %}
                {% if image.format == '.txt' %}
                    <div class="col-md-4">
                        <i class="bi bi-filetype-txt img-fluid rounded-start"
                            style="font-size: 80px;"></i>
                    </div>
                {% elif image.format == '.pdf' %}
                    <div class="col-md-4">
                        <i class="bi bi-filetype-pdf img-fluid rounded-start"
                            style="font-size: 80px;"></i>
                    </div>
                {% elif image.format == '.csv' %}
                    <div class="col-md-4">
                        <i class="bi bi-filetype-csv img-fluid rounded-start"
                            style="font-size: 80px;"></i>
                    </div>
                {% elif image.format == '.xlsx' %}
                    <div class="col-md-4">
                        <i class="bi bi-filetype-xlsx img-fluid rounded-start"
                            style="font-size: 80px;"></i>
                    </div>
                {% elif image.format in ['.doc', '.docx'] %}
                    <div class="col-md-4">
                        <i class="bi bi-filetype-doc img-fluid rounded-start"
                            style="font-size: 80px;"></i>
                    </div>
                {% endif %}
            {% endif %}
            <div class="col-md-8">
                <div class="card-body">
                    <h5 class="card-title"><strong>{{image.basename}}</strong></h5>
                    <h6 class="card-text">Formato do Arquivo: {{image.format}}</h6>
                </div>
            </div>
        </div>
    </div>
{% endfor %}

```

Implementação do arquivo .html que lê todos os dados de todos os arquivos em que foi feito upload e os expõe em uma lista, mostrando sua imagem (caso este arquivo seja no formato de imagem, ou seja, png, jpg, jpeg ou gif), o nome e o formato do arquivo

- Visualização da página HTML de listagem de imagens com algumas outras implementações:

Lista de Arquivos



**Cronograma curricular ciências da
computação UFPI.docx**
Formato do Arquivo: .docx



futebol.jpg
Formato do Arquivo: .jpg



Identidade.pdf
Formato do Arquivo: .pdf



natureza.jpg
Formato do Arquivo: .jpg



praia-tropical.jpg
Formato do Arquivo: .jpg

Realizar Novo Upload

Lista de Arquivos

Referência para mais detalhes e/ou dúvidas:

- <https://flask.palletsprojects.com/en/2.3.x/patterns/fileuploads/>