08

Mão na massa: Retocando aplicação e código

Chegou a hora de você executar o que foi visto na aula! Para isso, execute os passos listados abaixo.

> Se você quiser o projeto completo feito no treinamento anterior, pode baixá-lo aqui (https://caelum-online-public.s3.amazonaws.com/739-python-flask2/03/aula3.zip).

---

1) Para começarmos a embelezar nossa página, vamos mexer em **novo.html**. Dentro da *tag* `figure`, adicione as classes `thumb` e `col-md-4`:

```
<figure class="thumb col-md-4">
```

2) Dentro da tag `img`, adicione a classe `img-responsive`:

```
<img class="img-responsive" src="{{ url_for('imagem', nome_arquivo='capa_padrao.jpg') }}">
```

3) Agora vamos fazer a mudança na frase **No File Chosen**. Adicione a classe `fileContainer` e mude o conteúdo da `label`:

```
<label class="fileContainer">
    Mudar capa
    <input type="file" name="arquivo" accept=".jpg">
</label>
```

4) Em seguida, dentro da pasta **static**, crie um CSS chamado **app.css**. O seu conteúdo será:

```
body {
    padding-top: 10px;
}

.btn {
    margin-bottom: 10px;
}

.container {
    border-radius: 4px;
    margin: auto;
    width: 80%;
}

.little-container {
    width: 40%;
    margin: auto;
}

figcaption {
```

```
    text-align: center;
    margin: 3px auto;
}

.fileContainer {
    overflow: hidden;
    position: relative;
}

.fileContainer [type=file] {
    cursor: pointer;
    display: block;
    font-size: 999px;
    filter: alpha(opacity=0);
    min-height: 100%;
    min-width: 100%;
    opacity: 0;
    position: absolute;
    right: 0;
    text-align: right;
    top: 0;
}

td {
    text-align: center;
}
```

5) Dentro de **template.html**, abaixo do link que fizemos para o Bootstrap, adicione o link para o CSS criado acima:

```
<link rel="stylesheet" href="{{ url_for('static', filename='app.css') }}">
```

6) Agora, para começarmos a configurar a *preview* da imagem, baixe o jQuery (https://caelum-online-public.s3.amazonaws.com/739-python-flask2/04/jquery.zip) com o arquivo com a versão que usamos, extraia o zip e coloque-o na pasta **static**. Além disso, crie nessa mesma pasta o arquivo **app.js**, com o seguinte conteúdo:

```
$('form input[type="file"]').change(event => {
  let arquivos = event.target.files;
  if (arquivos.length === 0) {
    console.log('sem imagem pra mostrar')
  } else {
      if(arquivos[0].type == 'image/jpeg') {
        $('img').remove();
        let imagem = $('<img class="img-responsive">');
        imagem.attr('src', window.URL.createObjectURL(arquivos[0]));
        $('figure').prepend(imagem);
      } else {
        alert('Formato não suportado')
      }
  }
});
```

7) Agora precisamos *linkar* esses códigos recém-adicionados ao nosso projeto. Para isso, depois de fechar a `div container` no **template.html**, adicione:

```
<script type="text/javascript" src="{{ url_for('static', filename='jquery.js') }}"></script>
<script type="text/javascript" src="{{ url_for('static', filename='app.js') }}"></script>
```

8) Em **editar.html**, adicione também o `enctype` :

```
<form action="{{ url_for('atualizar') }}" method="post" enctype="multipart/form-data">
```

9) Também é preciso fazer as mudanças similarmente às que foram feitas em **novo.html**, de modo a ficar:

```
<figure class="thumb col-md-4">
    <img class="img-responsive" src="{{ url_for('imagem', nome_arquivo=capa_jogo) }}">
    <figcaption>
        <label class="fileContainer">
            Mudar capa
            <input type="file" name="arquivo" accept=".jpg">
        </label>
    </figcaption>
</figure>
```

10) Diferencie cada imagem, para isso use um *timestamp* para cada imagem. Para fazer isso, em **jogoteca.py**, na função `criar` modifique da seguinte forma:

```
@app.route('/criar', methods=['POST',])
def criar():
    nome = request.form['nome']
    categoria = request.form['categoria']
    console = request.form['console']
    jogo = Jogo(nome, categoria, console)
    jogo = jogo_dao.salvar(jogo)

    arquivo = request.files['arquivo']
    upload_path = app.config['UPLOAD_PATH']
    timestamp = time.time()
    arquivo.save(f'{upload_path}/capa{jogo.id}-{timestamp}.jpg')
    return redirect(url_for('index'))
```

Não se esqueça de importar o módulo `time` !

11) Para não selecionar as imagens aleatoriamente, pegue os nomes das imagens. Para isso, ainda em **jogoteca.py**, na função `editar` , antes do seu retorno, recupere a imagem:

```
nome_imagem = recupera_imagem(id)
```

12) Agora, implemente essa função logo depois da função `editar` :

```
def recupera_imagem(id):
    for nome_arquivo in os.listdir(app.config['UPLOAD_PATH']):
        if f'capa{id}' in nome_arquivo:
            return nome_arquivo
```

13) Modifique o render_template na função `editar`, passando o nome da imagem recuperado para `capa_jogo`, veja abaixo:

```
return render_template('editar.html', titulo='Editando Jogo', jogo=jogo
                           , capa_jogo=nome_imagem or 'capa_padrao.jpg')
```

14) Na função `atualizar` atualize o timestamp e salve a imagem, vai ficar assim:

```
@app.route('/atualizar', methods=['POST',])
def atualizar():
    nome = request.form['nome']
    categoria = request.form['categoria']
    console = request.form['console']
    jogo = Jogo(nome, categoria, console, id=request.form['id'])

    arquivo = request.files['arquivo']
    upload_path = app.config['UPLOAD_PATH']
    timestamp = time.time()
    deleta_arquivo(jogo.id)
    arquivo.save(f'{upload_path}/capa{jogo.id}-{timestamp}.jpg')
    jogo_dao.salvar(jogo)
    return redirect(url_for('index'))
```

15) Agora, delete as imagens criadas anteriormente (exceto a capa_padrao.jpg), que são desnecessárias e podem causar confusão. Em seguida, crie a função `deleta_arquivo` após a função recupera_imagem:

```
def deleta_arquivo(id):
    arquivo = recupera_imagem(id)
    os.remove(os.path.join(app.config['UPLOAD_PATH'], arquivo))
```

16) E chame-a dentro da função `atualizar`, antes de salvar o arquivo, ou vai ficar acumulando as imagens antigas:

```
deleta_arquivo(jogo.id)
```

17) Pode testar a aplicação agora!