02

Arrumando o casa

Transcrição

De volta, alunos e alunas! Nós vimos que temos muito código no módulo `jogoteca.py`, e é interessante isolarmos algumas funcionalidades em outros arquivos.

Por exemplo, não é ideal mantermos as configurações da nossa aplicação (`app.config`) no mesmo arquivo que o nosso código. Passaremos esses dados para um local específico, e, em `jogoteca.py`, chamaremos esse local com `app.config.from_pyfile('config.py')`.

Na raiz da nossa aplicação, criaremos o arquivo Python chamado `config.py`, no qual coloremos o código das configurações:

```
app.secret_key = 'alura'

app.config['MYSQL_HOST'] = "localhost"
app.config['MYSQL_USER'] = "root"
app.config['MYSQL_PASSWORD'] = "admin"
app.config['MYSQL_DB'] = "jogoteca"
app.config['MYSQL_PORT'] = 3306
app.config['UPLOAD_PATH'] = os.path.dirname(os.path.abspath(__file__)) + '/uploads'
```

Em seguida, importaremos o pacote `os` para que esse seja funcional nesse módulo, e transformaremos o resto das configurações em um texto com chaves e valor:

```
SECRET_KEY = 'alura'
MYSQL_HOST = "localhost"
MYSQL_USER = "root"
MYSQL_PASSWORD = "admin"
MYSQL_DB = "jogoteca"
MYSQL_PORT = 3306
UPLOAD_PATH = os.path.dirname(os.path.abspath(__file__)) + '/uploads'
```

Em seguida, criaremos um arquivo Python separado especificamente para as funções utilitárias, que chamaremos de `helpers.py`. Nele, precisaremos importar o pacote `os` e, do módulo `jogoteca`, `app`:

```
import os
from jogoteca import app

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

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

Também removeremos as *views* (as regras de rotas) de `jogoteca.py` e as colocaremos em um lugar específico - um arquivo Python criativamente chamado de `views.py`. Nele, colocaremos todo o código relativo a nossas rotas, além dos *imports* e dos objetos necessários para que elas funcionem.

```
from flask import render_template, request, redirect, session, flash, url_for, send_from_directo
import time

from models import Jogo
from dao import JogoDao, UsuarioDao

from helpers import deleta_arquivo, recupera_imagem
from jogoteca import db, app

jogo_dao = JogoDao(db)
usuario_dao = UsuarioDao(db)

@app.route('/')
def index():
    lista = jogo_dao.listar()
    return render_template('lista.html', titulo='Jogos',
                           jogos=lista)

@app.route('/novo')
def novo():
    if 'usuario_logado' not in session or session['usuario_logado'] == None:
        return redirect(url_for('login', proxima=url_for('novo')))
    return render_template('novo.html', titulo='Novo jogo')

@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'))

@app.route('/login')
def login():
    proxima = request.args.get('proxima')
    return render_template('login.html', proxima=proxima)

@app.route('/autenticar', methods=['POST',])
def autenticar():
    usuario = usuario_dao.buscar_por_id(request.form['usuario'])
    if usuario:
        if usuario.senha == request.form['senha']:
            session['usuario_logado'] = usuario.id
            flash(usuario.nome + ' logou com sucesso!')
            proxima_pagina = request.form['proxima']
            return redirect(proxima_pagina)
```

```
    else :
        flash('Não logado, tente de novo!')
        return redirect(url_for('login'))

@app.route('/logout')
def logout():
    session['usuario_logado'] = None
    flash('Nenhum usuário logado!')
    return redirect(url_for('index'))

@app.route('/editar/<int:id>')
def editar(id):
    if 'usuario_logado' not in session or session['usuario_logado'] == None:
        return redirect(url_for('login', proxima=url_for('editar')))
    jogo = jogo_dao.busca_por_id(id)
    nome_imagem =  recupera_imagem(id)
    capa_jogo = f'capa{id}.jpg'
    return render_template('editar.html', titulo='Editando jogo', jogo=jogo,
                           capa_jogo = nome_imagem)


@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'])
    jogo_dao.salvar(jogo)

    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')
    return redirect(url_for('index'))

@app.route('/deletar/<int:id>')
def deletar(id):
    jogo_dao.deletar(id)
    flash('O jogo foi removido com sucesso!')
    return redirect(url_for('index'))

@app.route('/uploads/<nome_arquivo>')
def imagem(nome_arquivo):
    return send_from_directory('uploads', nome_arquivo)
```

Nesse momento, se rodarmos nossa aplicação e tentarmos acessar qualquer rota, o navegador retornará um erro:

> **Not Found**
>
> The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

Mas por que isso aconteceu?

O arquivo `jogoteca.py` é um executável no qual criamos nossa aplicação, configuramos ela e o nosso banco de dados, e ao final inicializamos a aplicação. Porém, não inserimos um contexto de *views*, e precisamos configurá-las de alguma forma.

Se importarmos o arquivo `views.py`, cairemos no mesmo problema que tivemos anteriormente: um módulo estará importando o outro. Dessa forma, mesmo que importemos `views.py` de acordo com a ordem de execução (depois que a aplicação `jogoteca.py` é iniciada), o código `from jogoteca import app` daquele módulo fará com que a aplicação seja executada novamente, em *loop*.

Quando estamos executando um arquivo Python, podemos configurá-lo para que ele não seja executado em uma importação ou por outro arquivo. Para isso, os arquivos Python têm uma peculiaridade: quando são executados diretamente, o nome do módulo muda para `__main__`. Portanto, criaremos uma condição que se aproveita dessa peculiaridade:

```
from flask import Flask

from flask_mysqldb import MySQL

app = Flask(__name__)
app.config.from_pyfile('config.py')

db = MySQL(app)

from views import *

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

O código `from views import *` está sendo utilizado para importar todo o código em `views.py`.

Salvando essas alterações, nossa aplicação voltará a funcionar normalmente no navegador! Ótimo resultado, não acha?