**CURSA - Curso Desenvolvimento Web com Python e Django**

**Início 30/04/2025**

[https://www.cursa.com.br/home/course/curso-de-introdu%C3%A7%C3%A3o-ao-desenvolvimento-web-com-python-e-django/479](https://www.cursa.com.br/home/course/curso-de-introdução-ao-desenvolvimento-web-com-python-e-django/479)

**1. Criação de Ambiente Virtual**

- Para instalar pacote via Prompt de Comando (o CMD mesmo)

pip install <nome do pacote>[==<versão>]

**OBS.:** Se omitir a versão, instala a mais atual.

- Para atualizar pacote para última versão

pip install –-upgrade <nome do pacote>

Ex.:

***pip install --upgrade virtualenv***

***pip install --upgrade virtualenvwrapper-win***

- Para instalar apenas para o usuário atual do SO

pip install –-upgrade <nome do pacote> --user

- Para mostrar pacotes instalados e suas versões

pip freeze

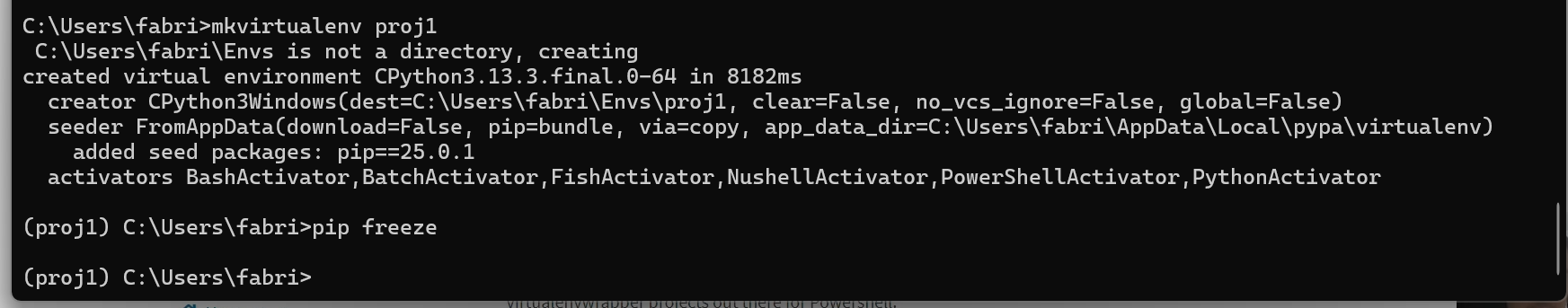
- Para outros comandos e opções

pip help

- Para criar ambiente virtual de desenvolvimento específico para o projeto

mkvirtualenv <nome do ambiente> (Ex.: Proj1)

**OBS.:** Logo após a criação do ambiente, o Python já assume que você passará a trabalhar naquele ambiente. Se comandar o Freeze dentro do ambiente, verá que não há nenhum pacote instalado nele.



- Para sair do ambiente virtual

deactivate

- Para trabalhar em um ambiente existente

workon <nome ambiente>

Para o projeto “Proj1”, conforme o curso, instalamos o pacote django na versão 2.2.12:  
pip install django ==2.2.12

E depois os demais pacotes indicados:

pip install django-crispy-forms

pip install django-braces

pip install django-cleanup

**OBS.:** Ao instalar os demais pacotes, como o django-crispy-forms, o Python automaticamente atualizou o django para a última versão.

- Para copiar saída de comando para algum arquivo, por exemplo, do comando “freeze”

pip freeze > <caminho\nome do arquivo>

- Carregar com um só comando todos os pacotes necessários ao projeto, precisa-se ter toda a listagem “freeze” dos pacotes e:

pip install -r <caminho\nome do arquivo>

Exemplo: para o proj1, geramos o arquivo “pacotes.txt”

*(proj1) C:\Users\fabri\Envs\proj1>* pip freeze > pacotes.txt

E instalamos os mesmos pacotes no proj1:

pip install -r pacotes.txt

- Para desinstalar pacote via Prompt de Comando

pip uninstall <nome do pacote>

- Para desinstalar a partir de uma lista de pacotes em um arquivo

pip uninstall -r packages.txt -y

- Para desinstalar os pacotes a partir da resposta do “freeze”

pip freeze | xargs pip uninstall -y

**2. Preparação do VSCode**

- Baixando da internet (site oficial Microsoft Visual Studio Code) e instalando.

- Barra lateral esquerda – menu “Explorer”: “Open folder” para abrir a pasta (folder) em que será trabalhado o projeto (workspace). Ou simplesmente escolhê-la, se já estiver aberta.

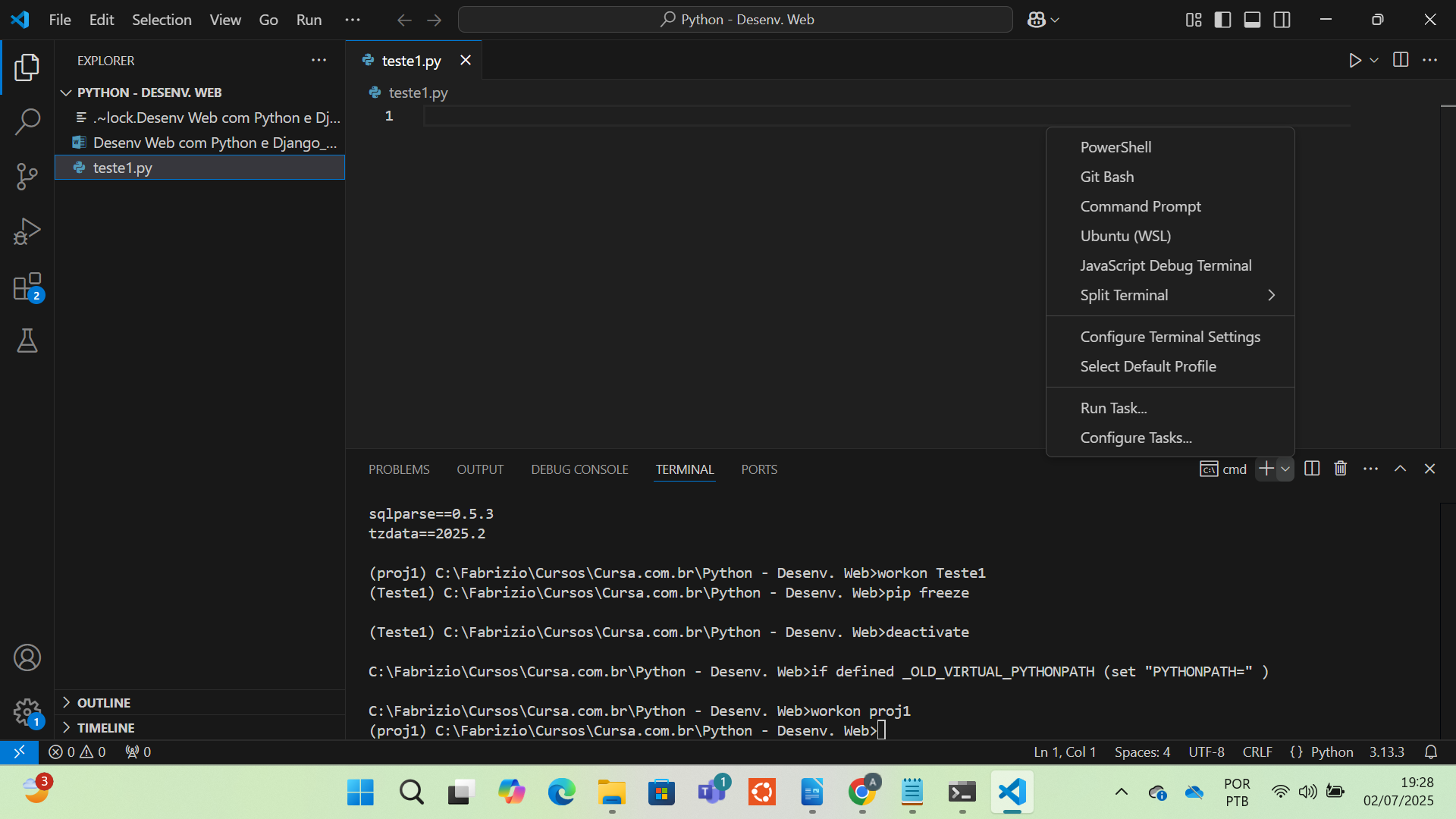
- Barra lateral esquerda – menu “Extensions”: instalar os pacotes necessários, digitando o nome deles na busca e clicando para instalar ou atualizar. Os seguintes pacotes foram instalados:

* **Portuguese brazil language pack** (para se ter a opção de ter toda a interface do VS traduzida para português, se desejado)
* **Python** (microsoft)
* Boostrap 4 Font awesome…
* jQuery Code Snippets
* Atom Keymap (para opções facilitadas de teclas de atalho, no lugar das nativas do VS)

- Os pacotes instalados passam a aparecer na sessão INSTALLED (ou Habilitados) do painel de Extensions.

- No painel de Explorer, criar um novo arquivo e salvar como arquivo Python (extensão **.py**). O VS já vai salvar na própria pasta do workspace.

- No menu superior “Terminal”, abrir um terminal. No quadro do terminal, escolher opção de perfim “Command prompt”, que funciona melhor com a linha de comando python.



- Usando o painel do terminal, já pode-se escolher trabalhar no ambiente virtual desejado. Escolhemos trabalhar no mesmo “proj1” criado na aula anterior.

- No Terminal, instalar 2 pacotes que ajudam o Vscode a identar melhor o código python:

pip install -U pylint

pip install -U autopep8

**-------------------------------------------------------------**

**Para inicializar a pasta do projeto Vscode como repositório e sincronizar com o Github nuvem:**

- Seleciona a pasta/projeto, no menu lateral esquerdo Explorer

- No menu lateral esquerdo “Source Control”, clicar no botão “Initialize repository” – o próprio nome da pasta se tornará o repositório.

- No item do repositório, clicar na opção de sincronização remota, à direita no nome da branch, para indicar a conta do Github em nuvem. Será solicitada autorização.

- A partir daqui o repositório já figura no Github, mas ainda sem a branch “main”.

- No terminal “Command Prompt”, digitar os comandos de configuração de usuário, para o Git local conseguir sincronizar as mudanças (commit) corretamente:

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

- Se a branch “main” já aparece no item do repositório e no canto esquerdo da barra de rodapé, então ela já existe localmente. Se comandarmos o “git status” a resposta será:

*On branch main*

*No commits yet*

Ou seja, a branch “main” existe localmente, mas nenhum commit ainda foi feito. Por isso ela ainda não aparece como branch válida e não é sincronizada no Github em nuvem.

Agora precisamos realizar um primeiro commit em algum arquivo da branch local.

- Escrever algum pequeno código no arquivo fonte – **no caso, teste1.py** – e salvá-lo.

- Na sessão “Source control”, o arquivo estará marcado com “M” (modified)

- Clicar no ícone “+” no arquivo para que ele fique com status “staged”. Somente os arquivos com status “staged” serão considerados no Commit.

- Escrever mensagem na caixa de texto acima do botão “Commit” para descrever brevemente a alteração.

- Clicar no botão “Commit” e aguardar finalização do processo.

- Após isso, o comando “git status” já deverá mostrar apenas:

*On branch main*

- O comando “git branch” já deve mostrar a branch válida:

*\* main*

- E o comando “git show-ref” deve mostrar algo do tipo:

*31e6a229ba0615c8aac087fea4d52be0cd9f10cd refs/heads/main*

*31e6a229ba0615c8aac087fea4d52be0cd9f10cd refs/remotes/origin/main*

- Verificar se o Github já está devidamente configurado como remoto:

git remote -v

A resposta deve ser algo do tipo:

*origin https://github.com/fabrizioduailibe/Python---Desenv.-Web.git (fetch)*

*origin https://github.com/fabrizioduailibe/Python---Desenv.-Web.git (push)*

- Se ainda não estiver configurado corretamente como remoto, usar o comando:

git remote add origin <URL\_DO\_SEU\_REPOSITORIO\_GITHUB>

- Fazer o PUSH da branch para o Github

git push -u origin main

Isso pode requerer novamente autenticação do usuário do Github. A resposta será algo do tipo:

*info: please complete authentication in your browser...*

*Enumerating objects: 9, done.*

*Counting objects: 100% (9/9), done.*

*Delta compression using up to 8 threads*

*Compressing objects: 100% (4/4), done.*

*Writing objects: 100% (9/9), 738 bytes | 147.00 KiB/s, done.*

*Total 9 (delta 0), reused 0 (delta 0), pack-reused 0*

*To https://github.com/fabrizioduailibe/Python---Desenv.-Web.git*

*\* [new branch] main -> main*

*branch 'main' set up to track 'origin/main'.*

- Configurar ou verificar qual é o repositório remoto associado à branch “main”:

git config branch.main.remote

A resposta deverá ser:

*origin*

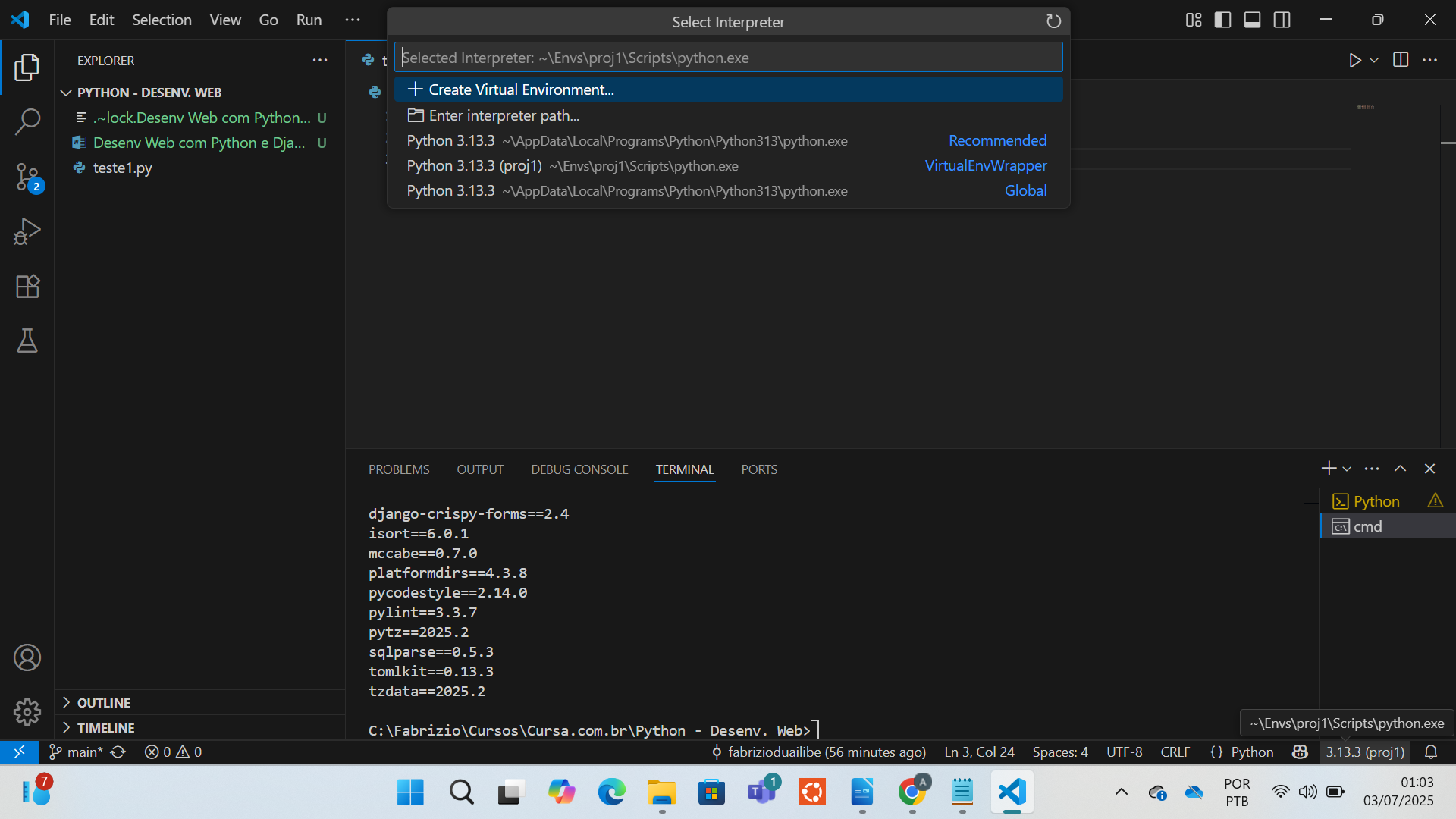
- Definir a branch para a qual a branch atual será mesclada quando você usar o comando “git merge” sem especificar um destino:

git config branch.main.merge

**-------------------------------------------------------------**

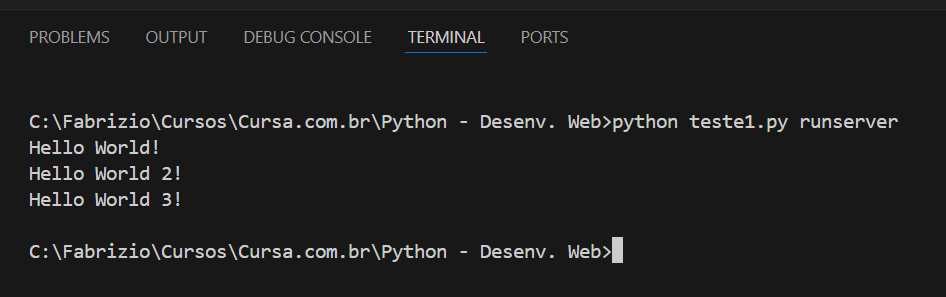
(CONT.)

- Podemos selecionar o ambiente virtual pela linha de comando no terminal, como mencionado na aula 1, ou podemos selecioná-lo clicando sobre a versão do python que aparece no canto inferior direito e selecionando a interpretador com ambiente virtual desejado da lista que aparece no topo.



- Para rodar um projeto diretamente pela linha de comando:

python <nome projeto.py> runserver



- Para instalar outras linguagens e seus acessórios para permitir melhor identação e visualização na codificação, pasta pesquisar nas extensões.

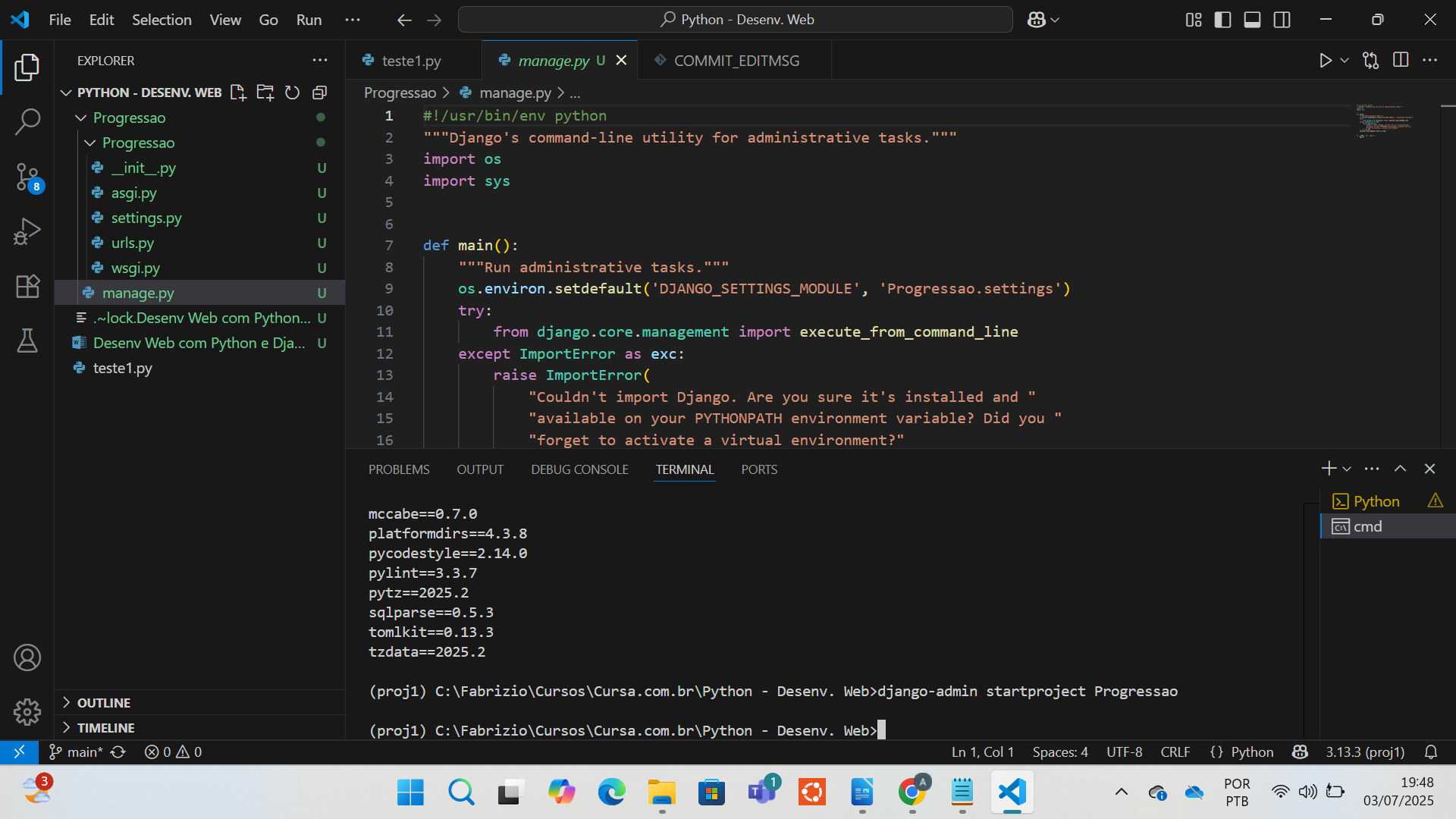
**3. Python e Django 01 – Criação de um projeto e configuração para Português BR**

- Escolher o ambiente virtual em que vai trabalhar (com os pacotes que atendem a necessidade do projeto) – comando workon

- Dentro da pasta que deseja instalar o projeto Python, usar o comando de criação do projeto básico:

django-admin startproject <nome projeto>

O arquivo “manage.py” é o arquivo de iniciação do servidor e de condução das conexões com bancos de dados, etc.



O arquivo “settings.py” é o arquivo de configurações. É nele, por exemplo, que está a configuração da linguagem a usar.

- Mudar a linguagem para Português BR

LANGUAGE\_CODE = ‘pt-br’

- Ajustar o time zone para o nosso fuso horário

TIME\_ZONE = ‘America/Sao\_Paulo’

- Executar o comando de geração da base de dados básica do projeto – “db.sqlite3” –, dentro da pasta onde está o arquivo “manage.py”.

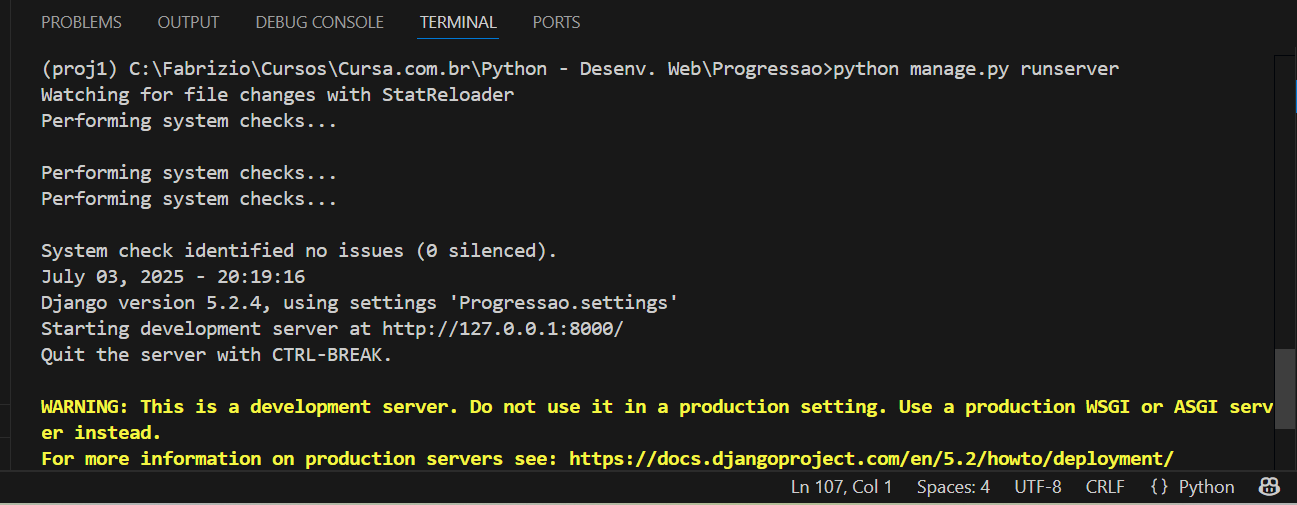
python manage.py migrate

O tipo e o nome da base de dados foram configurados no arquivo “settings.py”, sessão “DATABASES”.

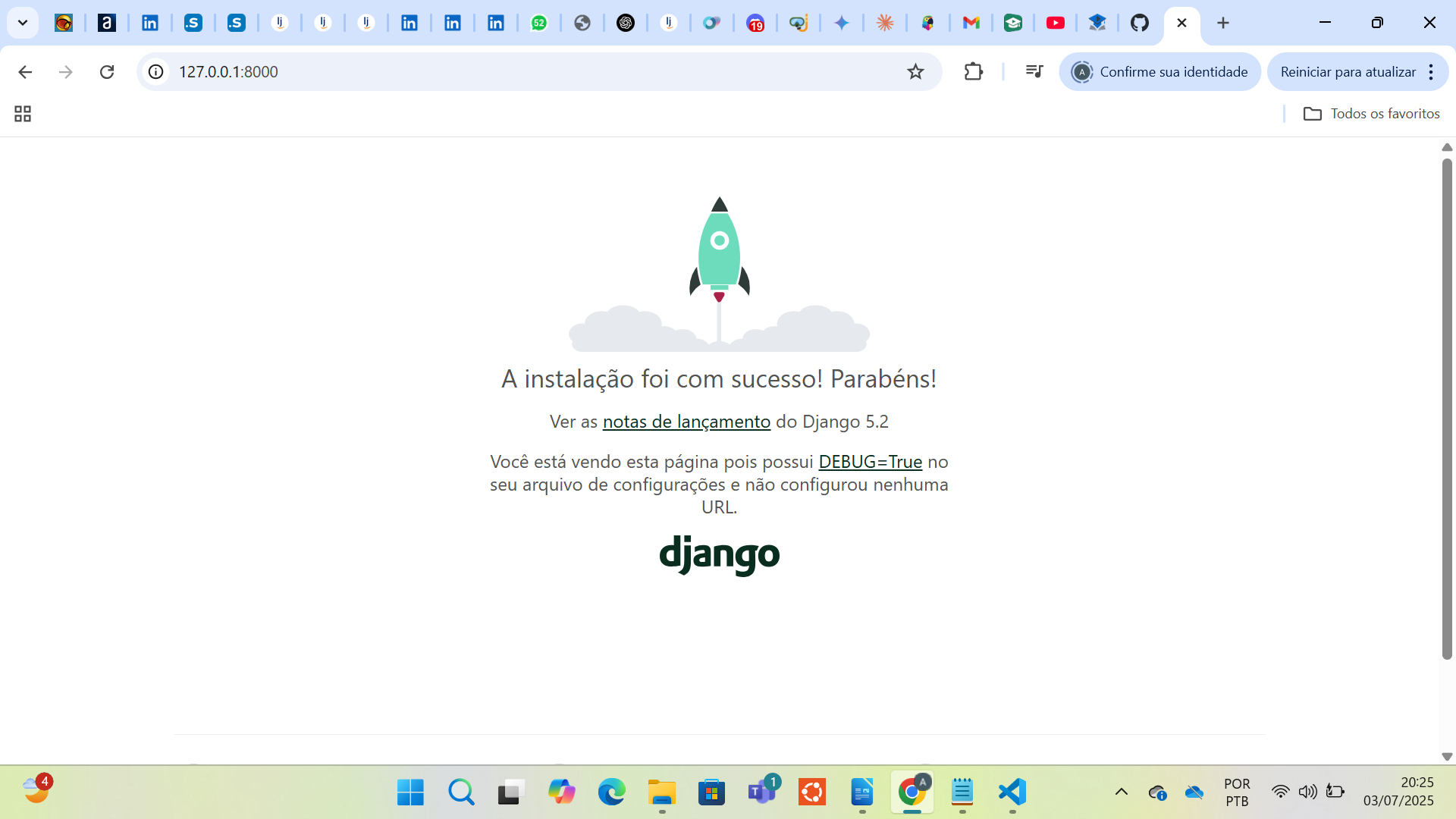
- Rodar o servidor do projeto django:

python manage.py runserver

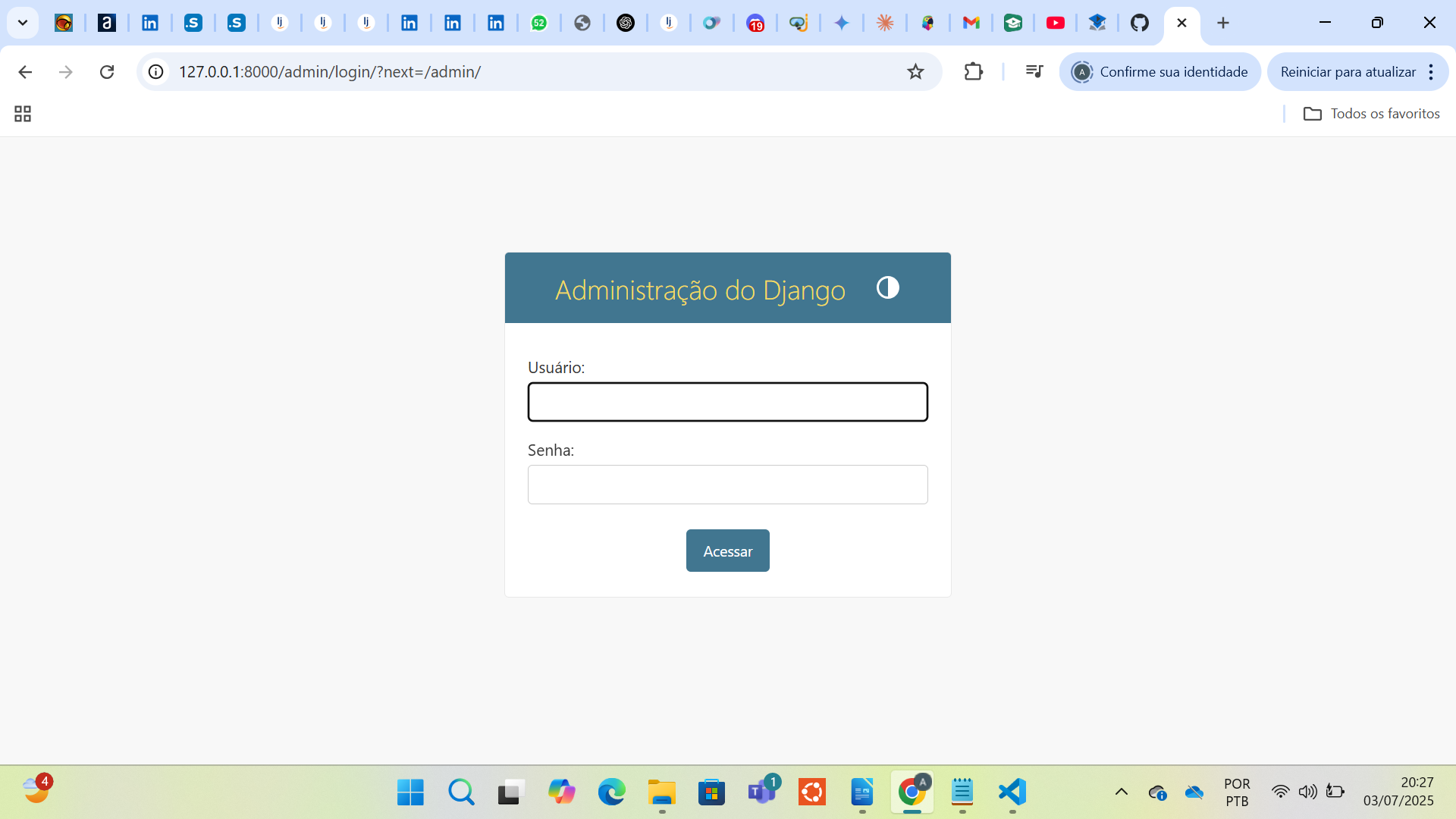
- **ALT + clique** no link de http local (<http://127.0.0.1:8000/>) atribuído ao servidor, para abrir no seu navegador padrão.



Esta é a tela inicial do link:



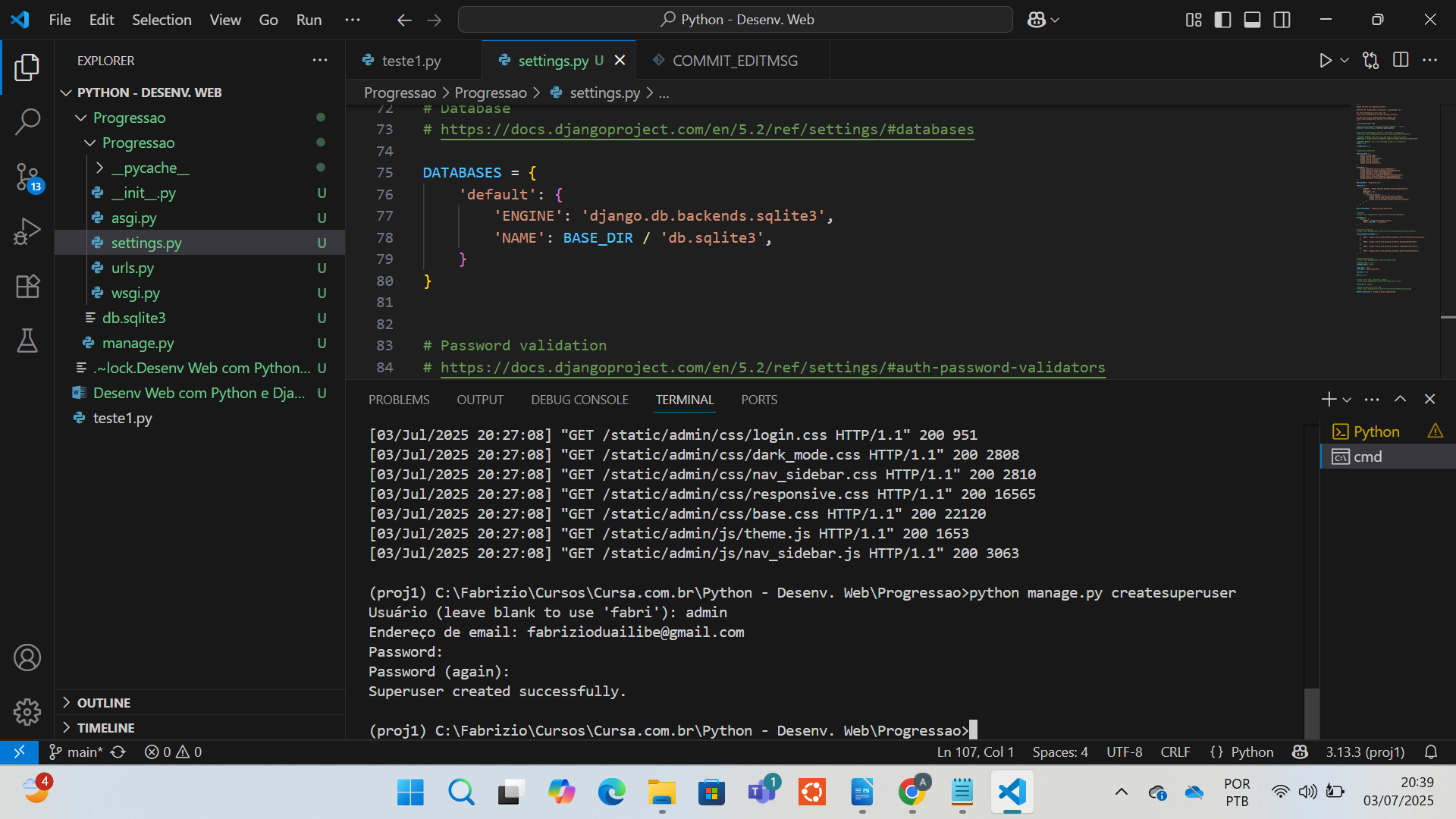
- Adicionando “/admin” ao endereço http, acessamos a página administrativa do servidor Django



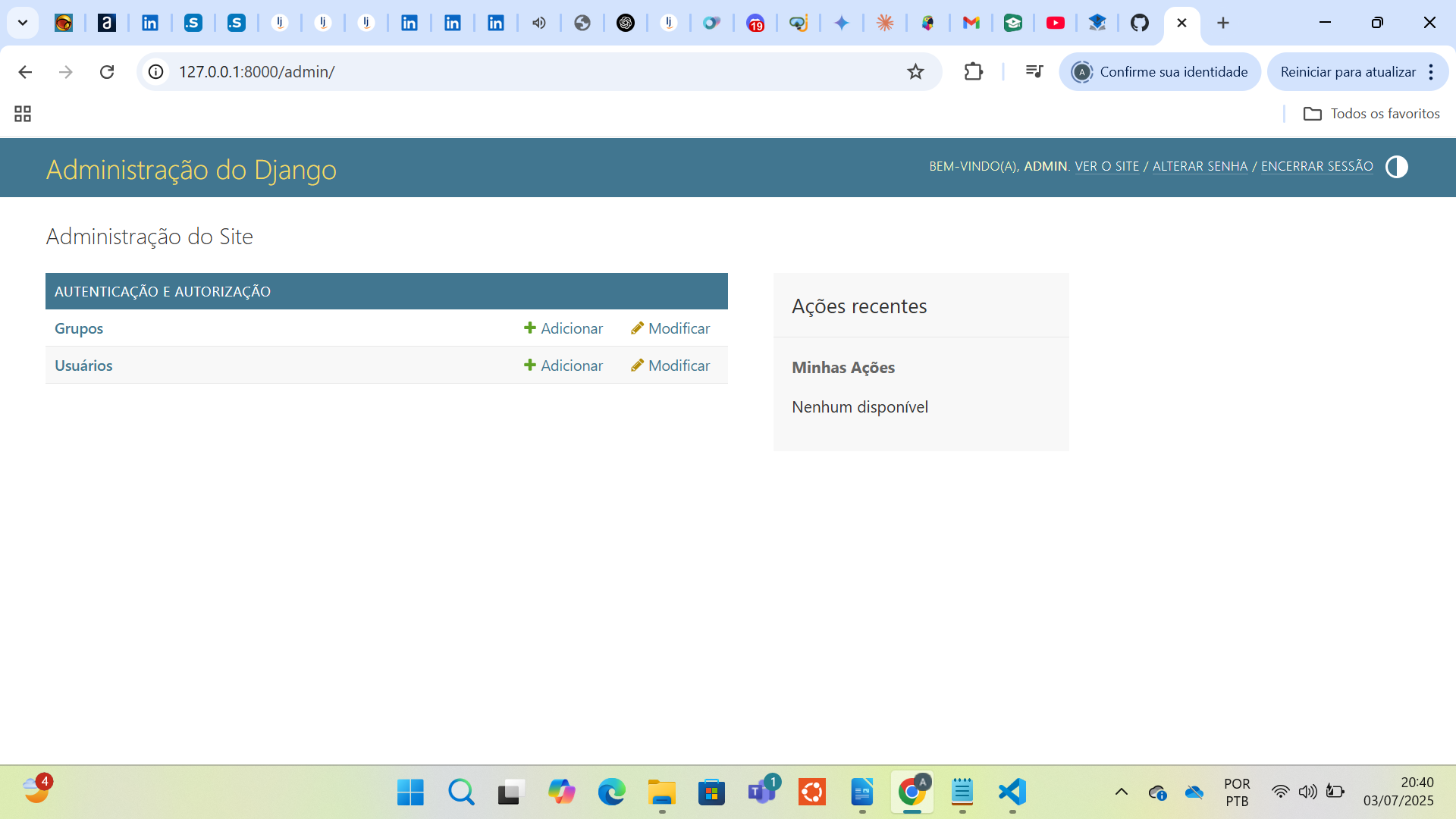
- Para criar o primeiro usuário, precisamos parar o servidor (CTRL+”C”) e criar o usuário master:

python manage.py createsuperuser

Após isso, fornecer nome de usuário (normalmente “admin”), e-mail, senha e confirmação de senha ao ser questionado. OBS.: senha bacuritu$4ba.



- Executando novamente o servidor e entrando com o nome de usuário e a senha configuradas, conseguimos acessar o servidor.



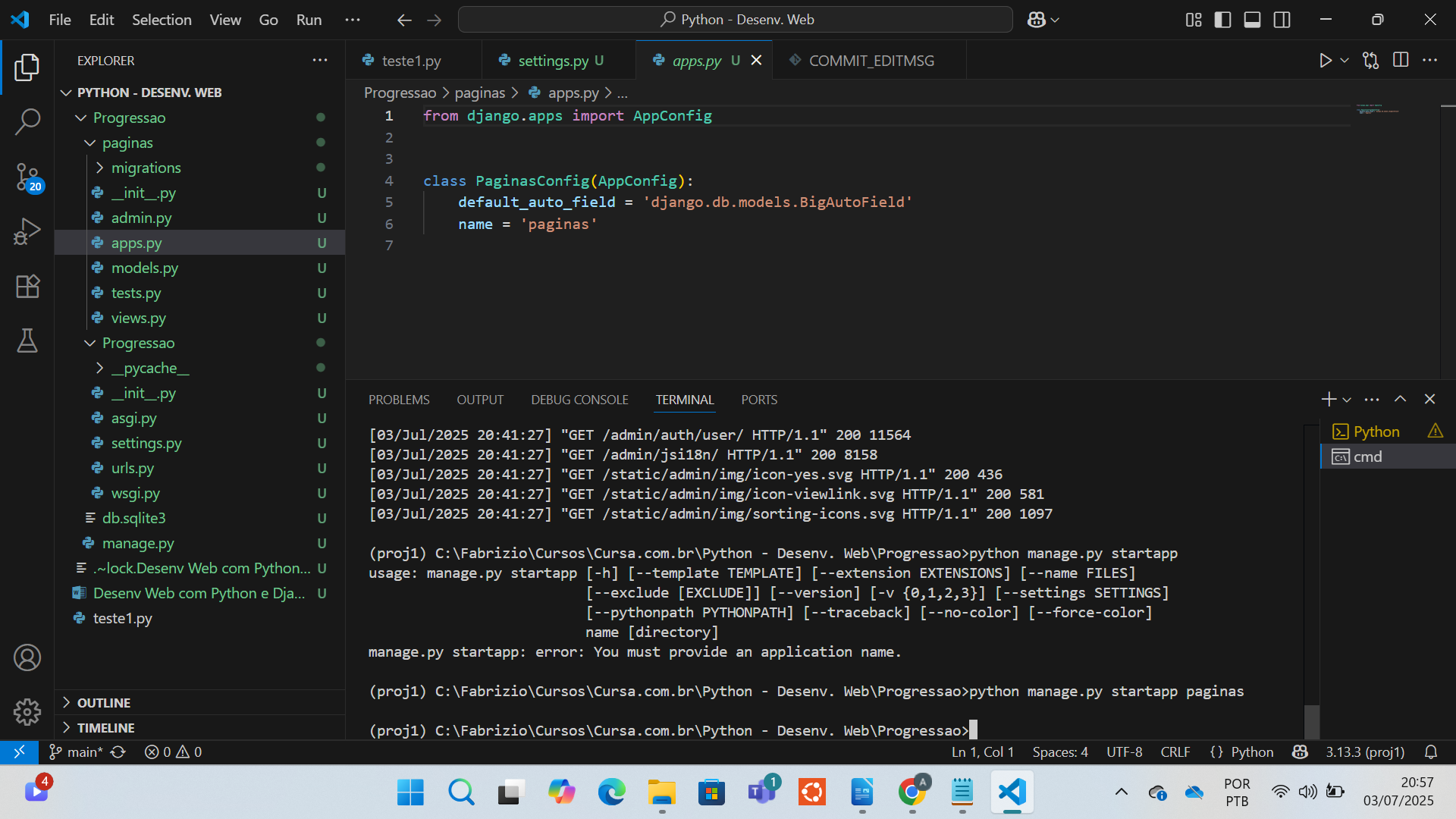
As classes Usuários e Grupos já vem com interface preparada com várias configurações importantes, como membro, permissões de acesso, rastreio de login e ações, etc. O painel administrativo básico do site já vem pronto!

- O Python é inerentemente modular, permitindo melhor organização das diferentes funcionalidades/mecanismos do sistema em módulos.

- Parar novamente o servidor e usar o comando “startapp” para criar automaticamente um novo módulo.

python manage.py startapp <nome módulo>

- O Python cria automaticamente uma subpasta na pasta de projeto com o mesmo nome e já povoa com os arquivos básicos necessários.



**4. Python e Django 02 – Criação de um app e configuração de views e urls**

- “**\_init\_.py**” → serve para indicar ao Pyhton quais pastas deve interpretar pacotes.

- “**admin.py**” → para registrar as classes que desejamos que apareçam no painel administrativo, como “usuários” e “grupos”.

- “**apps.py**” → é chamado para ativar o módulo dentro do projeto.

- “**models.py**” → armazena todas as classes do sistema. Por exemplo, “produto”, “pedido”, etc.

- “**tests.py**” → utilizado para realização de testes automatizados.

- “**views.py**” → armazena as views (interfaces visuais), onde vai acontecer grande parte do código.

- “**urls.py**” → configura o endereço para chamada das páginas de cada módulo, sempre unindo o nome de endereço fornecido ao endereço base do servidor.

- Começando no arquivo “**views.py**”, importamos a biblioteca genérica de templates de views que será usada e instanciando objeto da classe TemplateView, que é uma classe já disponibilizada para template padrão. Indicamos que essa view vai ser renderizada no template index.html:

from django.views.generic import TemplateView

class IndexView(TemplateView):

template\_name = “index.html”

Também é necessário indicar no arquivo “**settings.py**” a pasta onde o módulo (app) deve buscar os templates Html e salvar o arquivo.

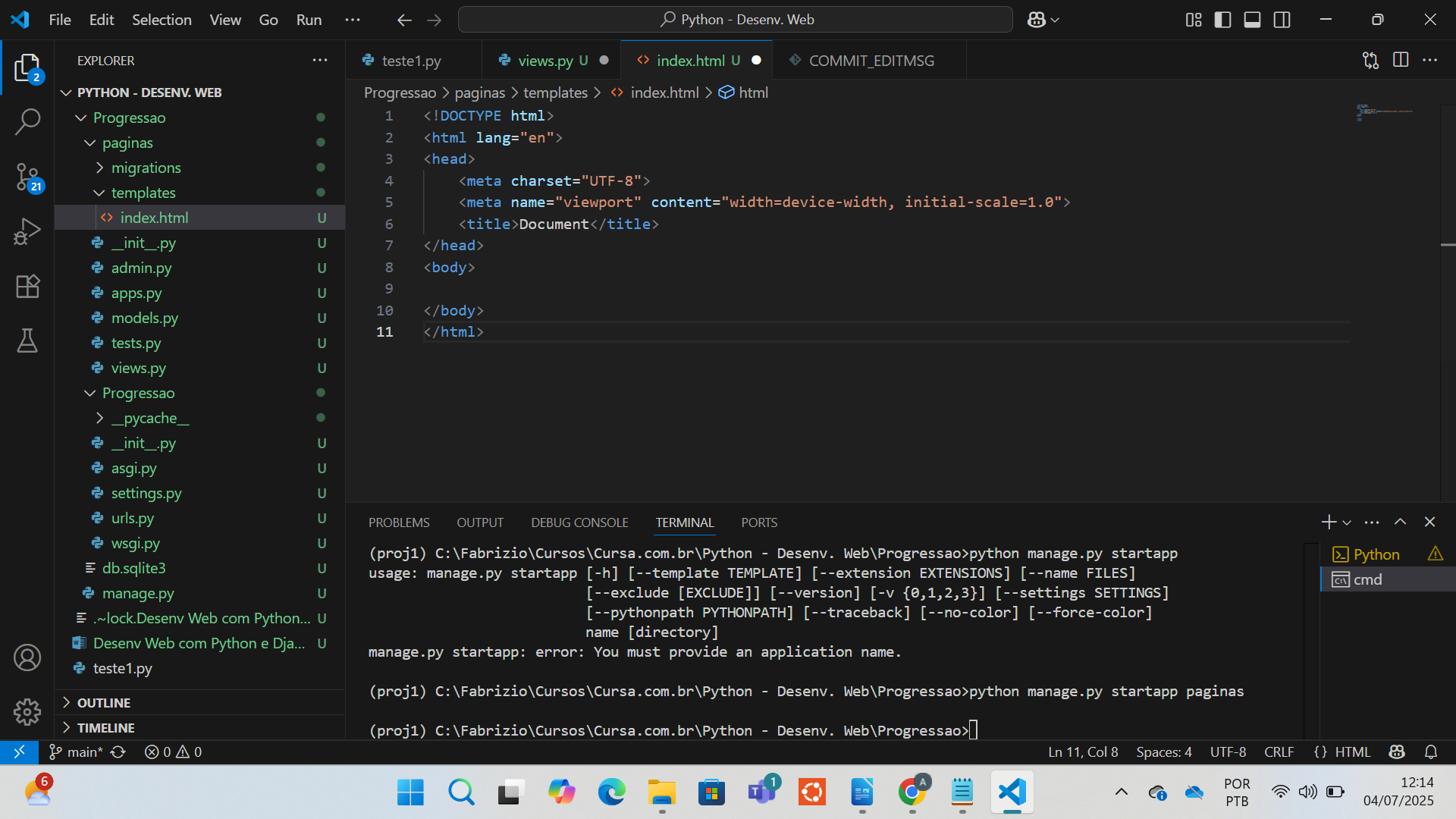
'DIRS': [os.path.join(BASE\_DIR, 'templates')],

**OBS.:** Só funciona se a biblioteca “os” também já está importada no arquivo “settings.py”:

import os

**Criar manualmente, dentro da pasta do módulo “paginas”, a subpasta com mesmo nome “templates”.**

Criar dentro dessa subpasta o arquivo “**index.html**” que indicamos no arquivo “views.py”. No Vscode, basta digital “html” e selecionar na lista de recomendações a opção “html: 5” em diante, que ele preenche o arquivo com conteúdos básicos completos.



Personalizar a linguagem (“lang”) e o título da página (“title”), e inserir um código simples no “body” para teste.

- Indicar qual a URL que vai acionar a visualização dessa página, criando o arquivo **“urls.py” dentro da pasta do módulo “paginas”** e informando lá o caminho.

from django.urls import path

from .views import IndexView

**OBS.:** Usar o “.” antes da biblioteca indica que o Python deve buscá-la nos arquivos locais do módulo ou do projeto. E não é necessário adicionar a extensão “.py”, senão ele vai entender que precisa buscar especificamente a classe “py” dentro da biblioteca “views” local.

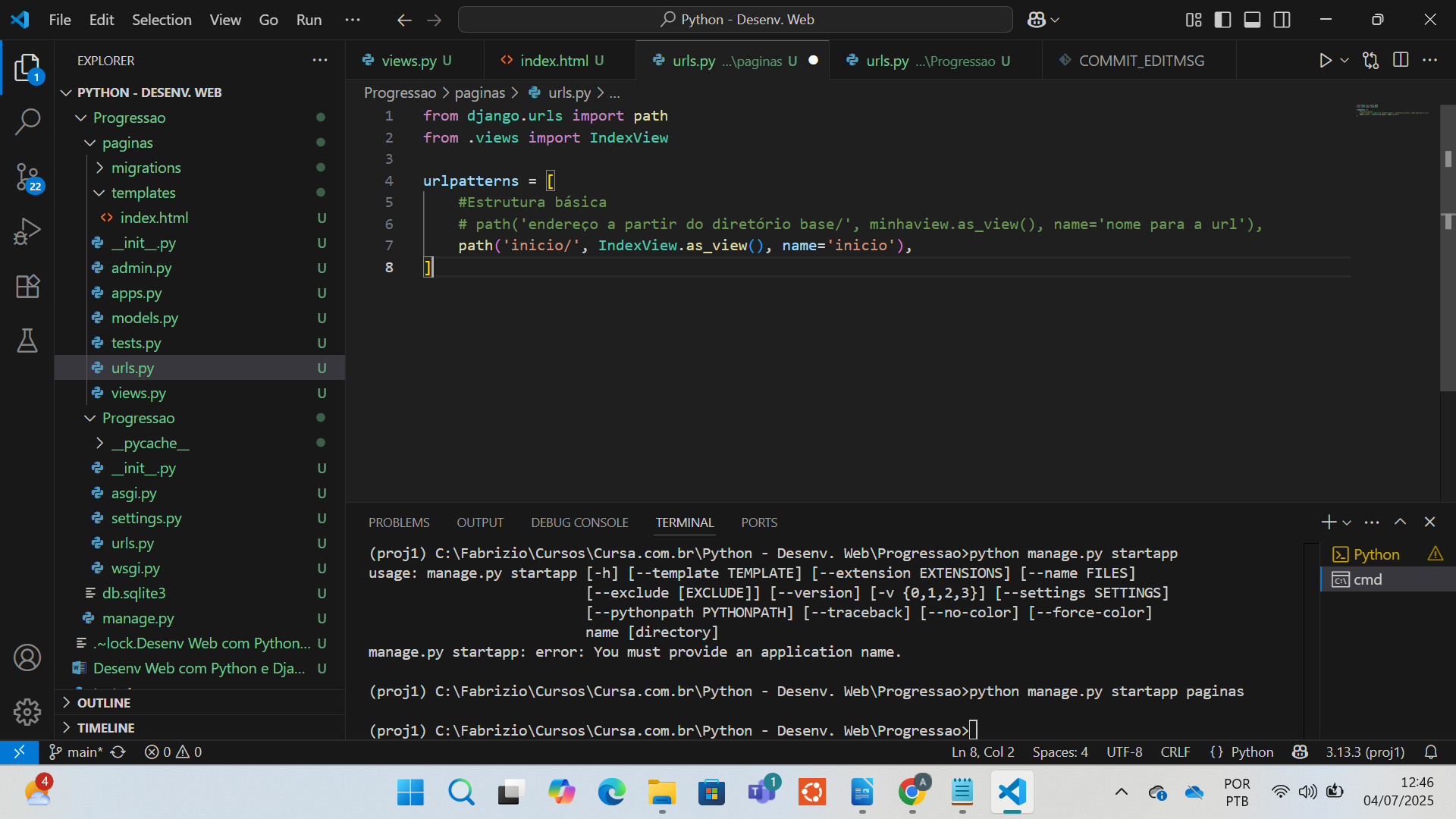
urlpatterns = [

# Estrutura básica

# path('endereço a partir do diretório base/', minhaview.as\_view(), name='nome para a url'),

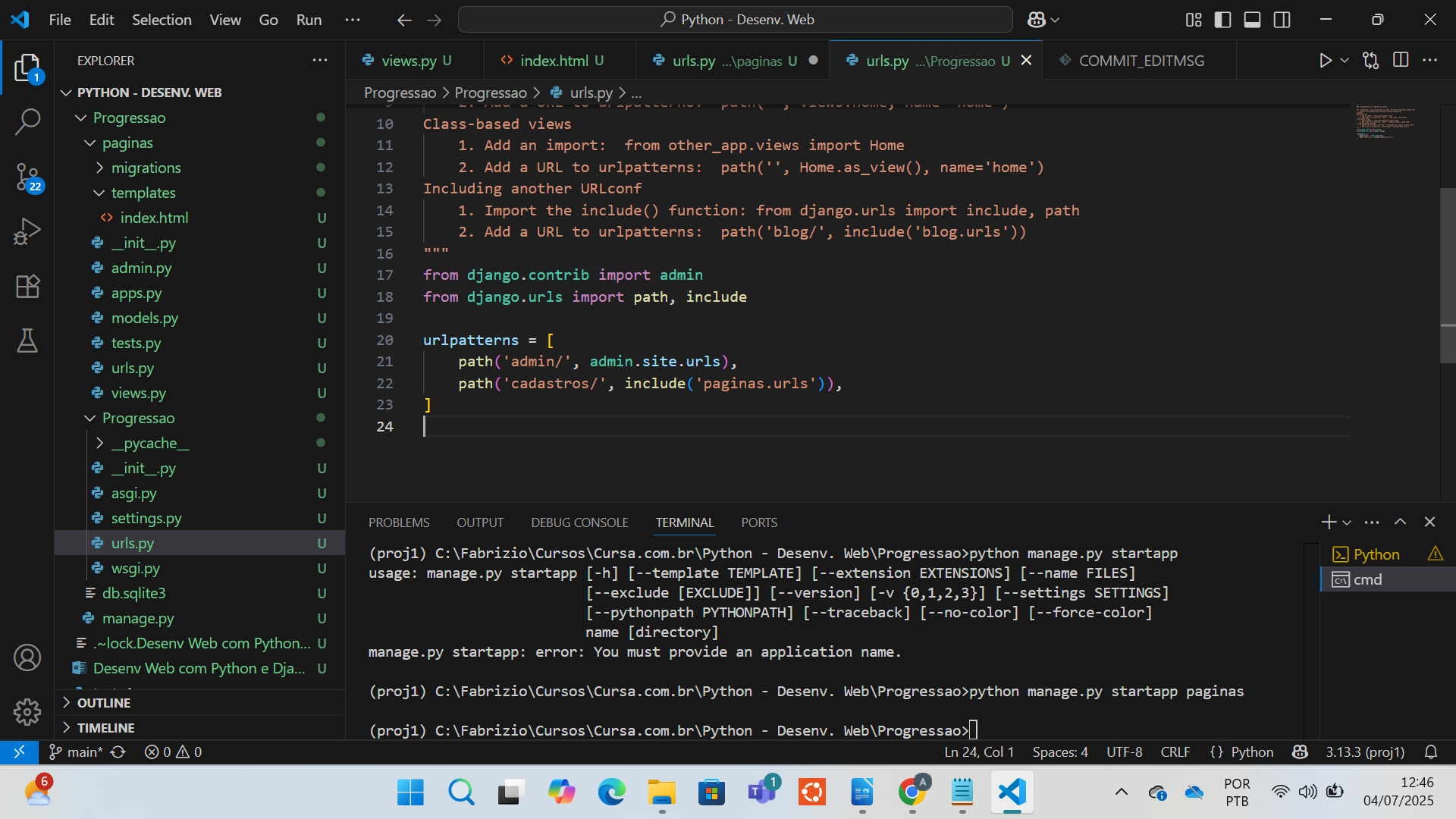
path('inicio/', IndexView.as\_view(), name='inicio'),

]



E, como o arquivo **“urls.py” do projeto completo (“Progressao”)** também precisa ter conhecimento das urls de todos os módulos contidos nele, precisamos atualizá-lo também, inserindo no arranjo de o caminho para encontrar as Urls do novo módulo “paginas”:

path('cadastros/', include('paginas.urls')),



**OBS.:** **Usamos um endereço intermediário “\cadastros” para melhor organização.** Todas as urls do “paginas” serão acessadas chamado [url\_base]\cadastros\[url\_paginas]. Se não quisermos ter essa url intermediária e usar diretamente as urls do módulo, basta deixar ‘’ (vazio) no primeiro parâmetro do “path”.

- Por fim, para finalizar e realizar o primeiro teste, precisamos informar o nosso módulo (nome da classe criada automaticamente no arquivo “**apps.py**” do “paginas”) na lista de INSTALLED\_APPS do “**settings.py**” do projeto:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

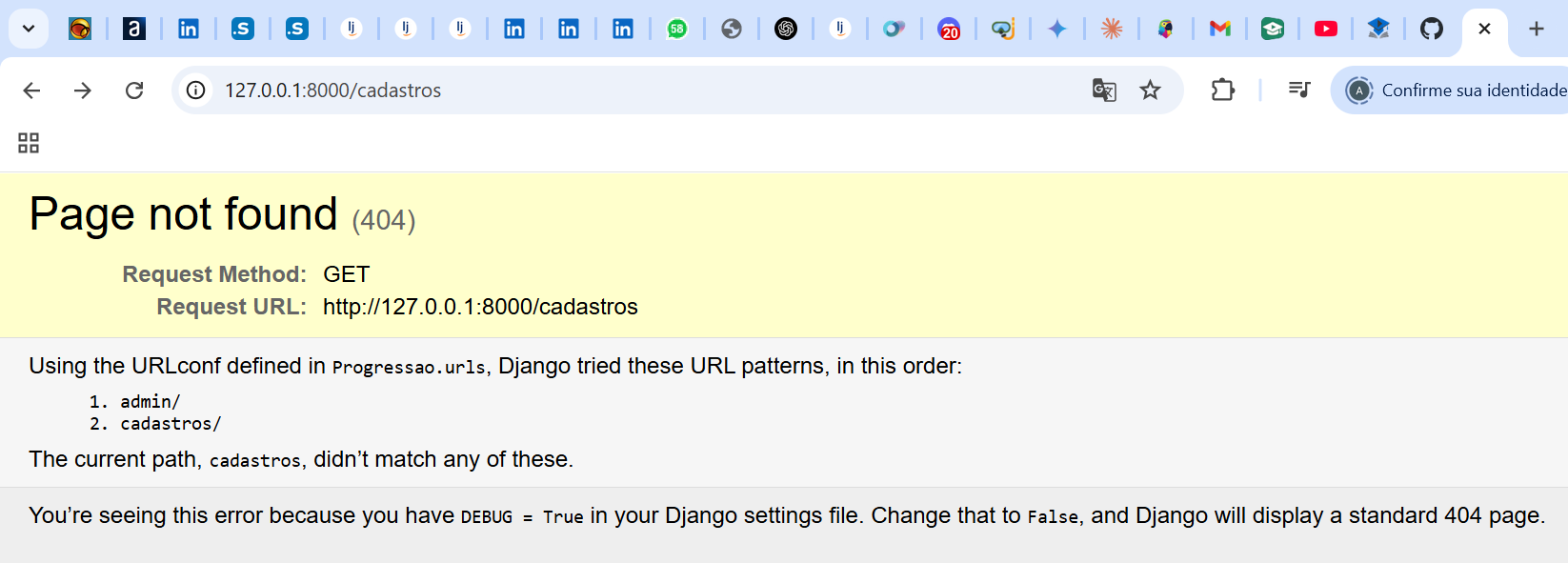
'django.contrib.messages',

'django.contrib.staticfiles',

'paginas.apps.PaginasConfig',

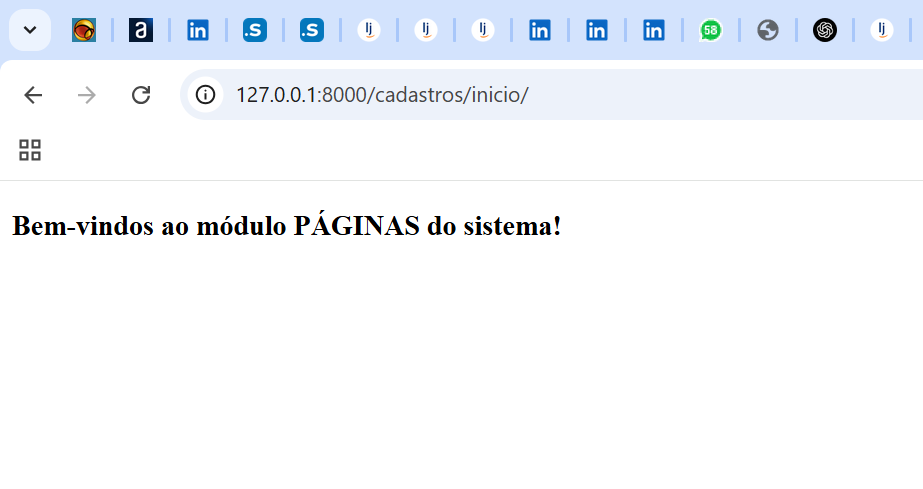
]

Teste – usando a url apenas com “cadastros”, sem “inicio”:



**OBS.:** Estas informações sobre o erro só aparecem no modo DEBUG – configurado no “setting.py”:

DEBUG = True

Teste – usando a url com “inicio” – **sucesso**:  


**5. Python e Django 03 – Configurar arquivos estáticos (CSS, JS e imagens)**

- Para que todos os apps do projeto possam utilizar os mesmos arquivos estáticos, **criamos a pasta “static”** no mesmo nível do arquivo “manage.py”. Dentro dela podemos organizar subpastas ‘js”, “css”, “img”, “fonts”, etc.

- Confirmar se o no arquivo “**settings.py**” estão declaradas as constantes STATIC\_URL e STARTICFILES\_DIR, que precisam ser direcionadas para a pasta criada:

STATIC\_URL = 'static/'

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'static')

]

- Para que qualquer **template (html)** acesse componentes externos, deve-se carregar a biblioteca (pasta ou arquivo) no início do arquivo, com a tag django “**{% load <biblioteca> %}**”.

**No caso de componentes estáticos, já existe a tag pré-definida “load static”** – para usá-la corretamente foram necessárias as configurações nos 2 itens acima, da forma que estão.

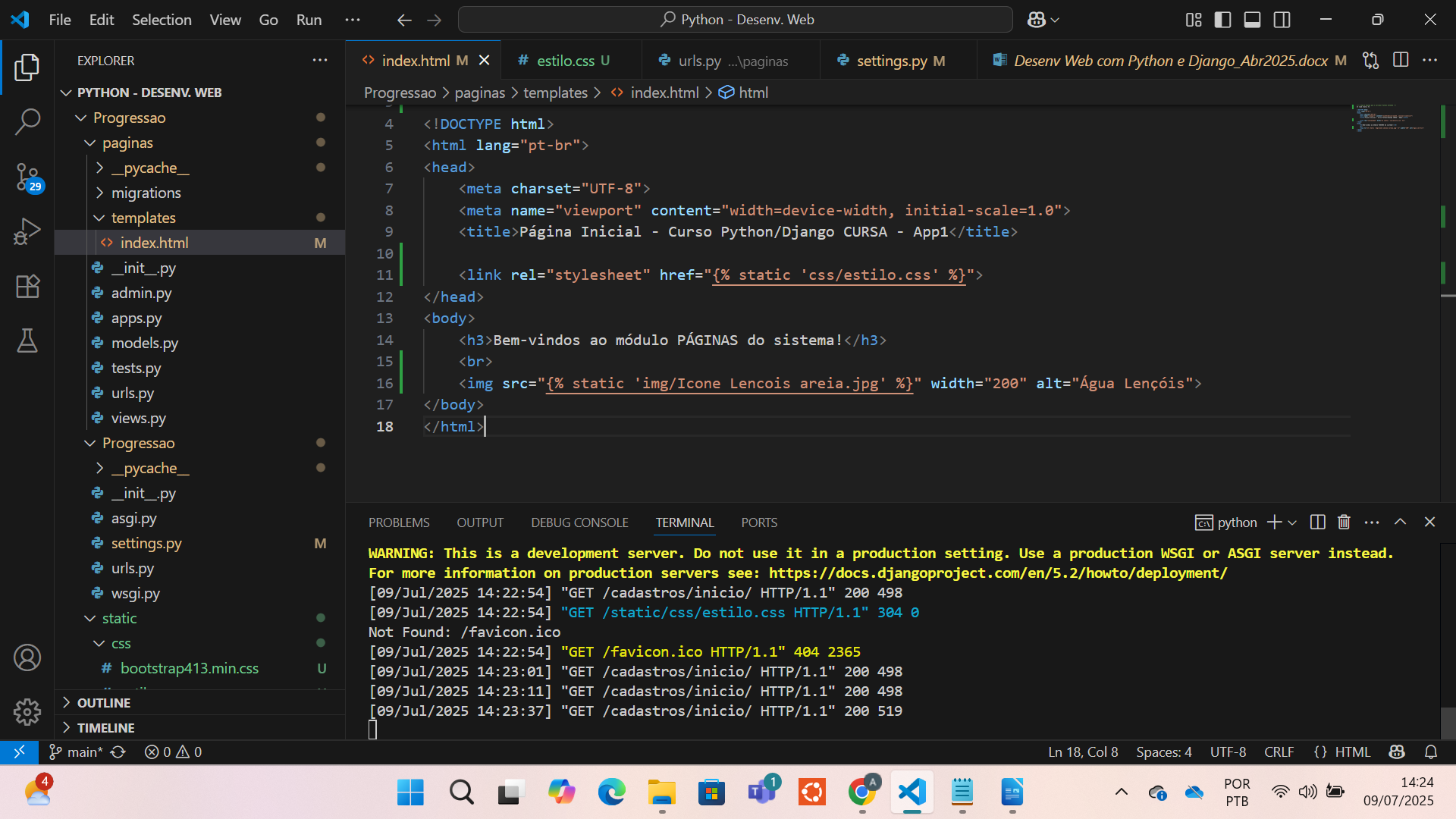
Para referenciar os componentes da biblioteca, usa-se o formato **{% <nome biblioteca> ‘<caminho do arquivo>’ %}**.

Por exemplo, no “index.html”:

{% load static %} (no início)

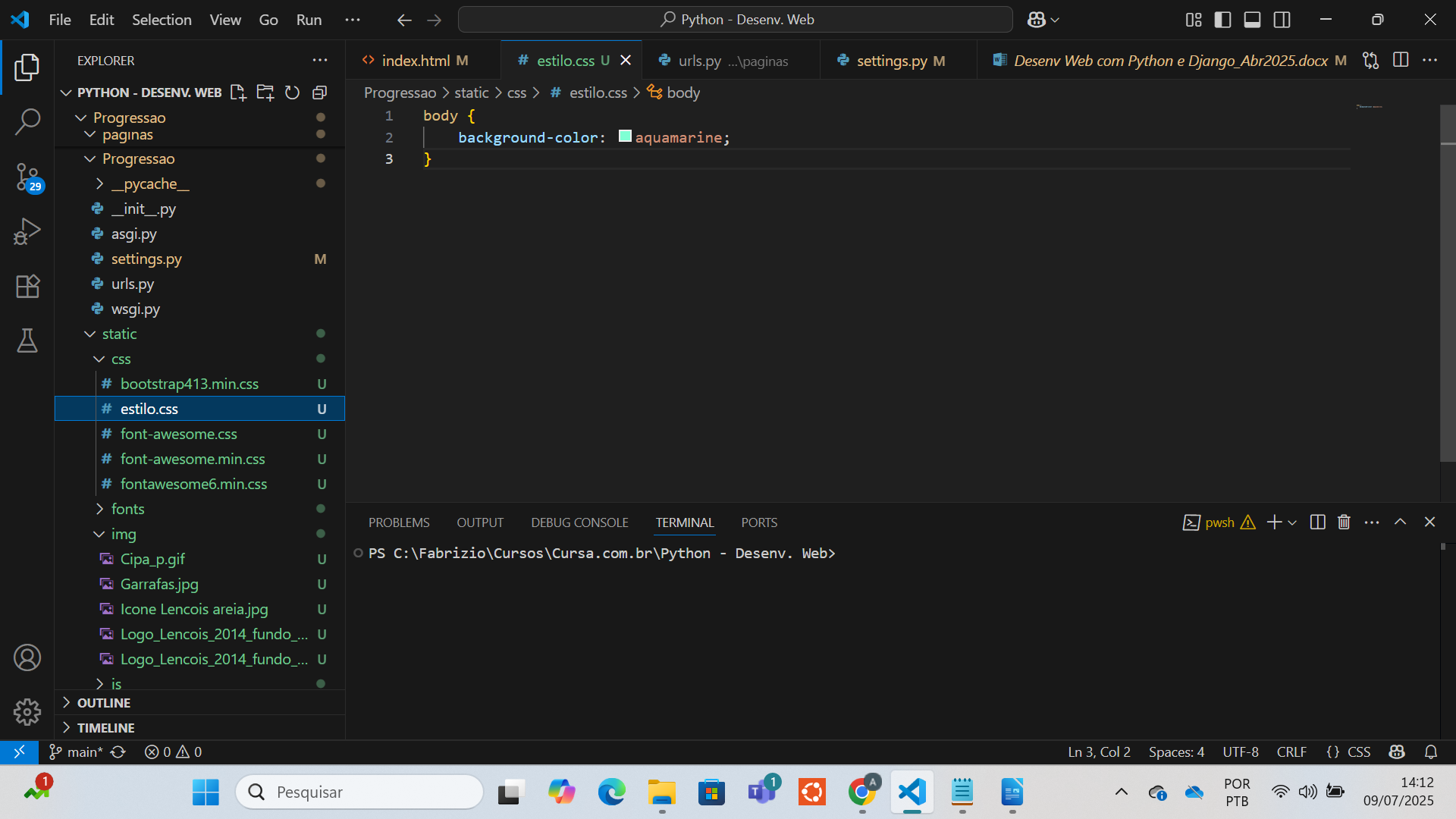
<link rel="stylesheet" href="{% static 'css/estilo.css' %}"> (no Head, onde se carregam os CSSs)

<img src="{% static 'img/Icone Lencois areia.jpg' %}" alt="Água Lençóis"> (no Body)

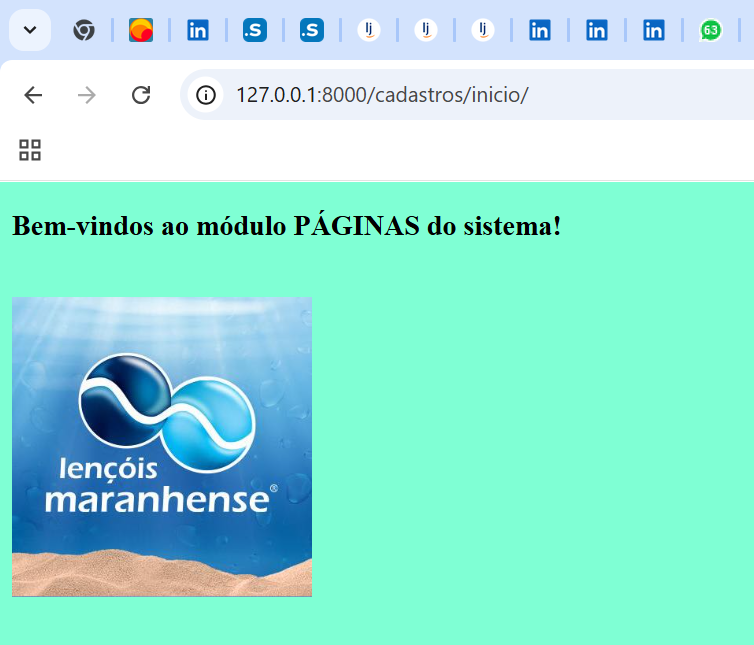


- Precisa-se providenciar dentro das subpastas “css” e “img” da pasta “static”, respectivamente, os arquivos “estilo.css” e “Icone Lencois areia.jpg” declarados no código acima, para que eles sejam encontrados e carregados devidamente.

- No arquivo “estilo.css”, podemos configurar por exemplo a cor do background:



- Resultado ao se carregar a página:



**6. Python e Django 04 – Configurar e reutilizar templates com Blocos: {% bloco conteúdo %}**

- Antecipadamente baixamos outros componentes estáticos úteis, como FONTES Awesome, módulos JAVASCRIPT como bootstrap.js, jquery.js e popper.js, módulos CSS como as font-awesome.css, etc.

- **Criar novo arquivo template** – ex.: modelobase.html –, **que servirá como base para todas as páginas que queremos que tenham a mesma estrutura básica**.

- Preparar o template para usar as bibliotecas estáticas:

(no topo)

{% load static %}

(no Head)

<link rel="stylesheet" href="{% static 'css/bootstrap413.min.css' %}">

(no Body)

<script src="{% static 'js/bootstrap.min.js' %}"></script>

<script src="{% static 'js/jquery-3.7.1.min.js' %}"></script>

<script src="{% static 'js/popper.min.js' %}"></script>

- Para testar se tudo foi carregado, codificar no Body alguns blocos Html de exibição. Por exemplo:

    <div class="bg-light">

        Código do meu Menu

    </div>

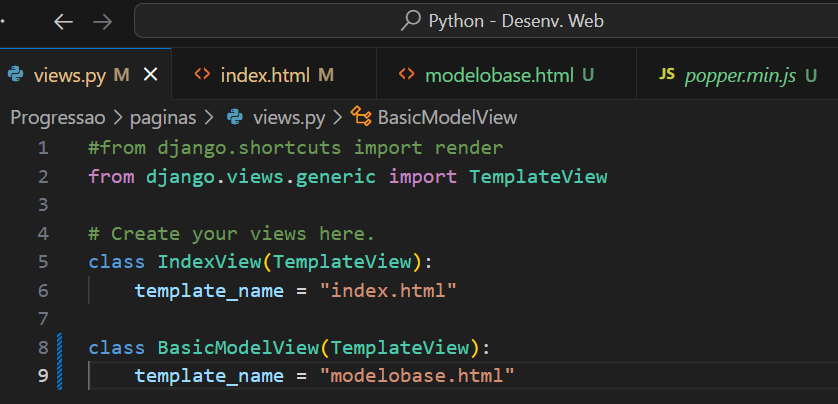
    <div class="container">

        <h3>Template Modelo</h3>

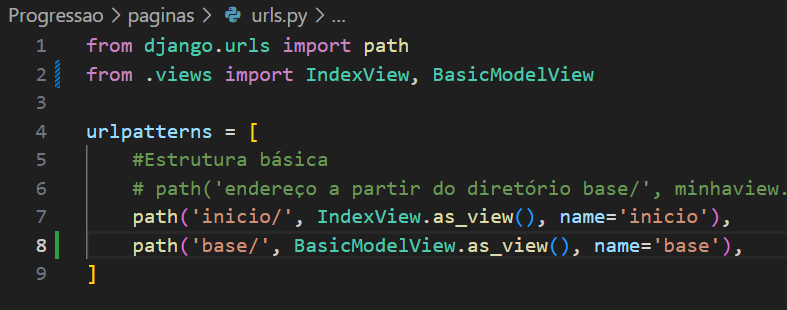
        <p>Conteúdo do site</p>

    </div>

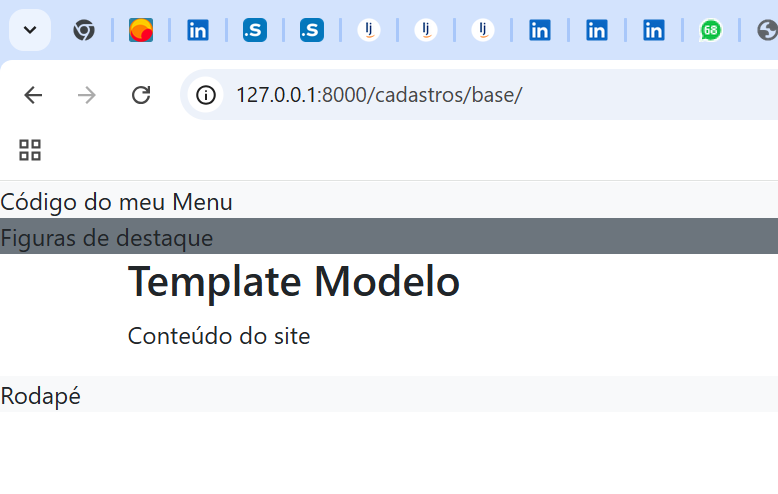
- Para que esse novo template seja acessado no site, precisamos configurá-lo em uma View (views.py):



E configurar a Url para acessar a view (urls.py)



- Rodando a página “base” que vinculamos ao novo template:



**OBS.:** Notar que a simples variação do atributo “class” nas tags Html muda o formato aplicado no bloco. “container”, “bg-light” e outras configurações de formato prontas para uso estão configuradas no CSS carregado.

- Todos os pontos do template Html que desejarmos adaptar durante a reutilização precisam ser envolvidos com a tag django **{% block %}**, em qualquer parte do arquivo, **inclusive no Head**.

Por exemplo:

    {% block titulo %}

    <title>Modelo Base</title>

    {% endblock %}

- Agora vamos criar uma nova página “sobre”, que queremos que tenha a mesma estrutura do “modelobase”. Mas ao invés de copiar a estrutura e personalizar os blocos, vamos usar tags Django para reaproveitar o template “modelobase”.

Criar a página “sobre.html” e referenciá-la em “views.py” e “urls.py”.

Em “sobre.html”, usamos apenas a tag **{% extends %}** e assim já trazemos todo o conteúdo do template “modelobase.html”.

{% extends 'modelobase.html' %}

- Para realizar as adaptações da página “sobre.html”, reescrevemos os blocos delimitados por {% block %}. Por exemplo:

{% extends 'modelobase.html' %}

{% block titulo %}

<title>Sobre o projeto</title>

{% endblock %}

{% block conteudo %}

<h3>Sobre o sistema</h3>

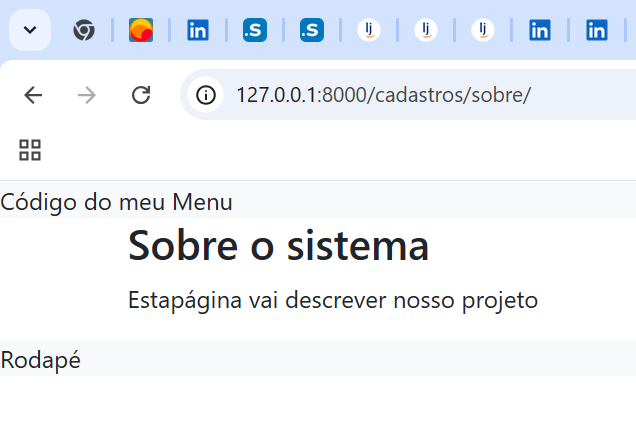
<p>Estapágina vai descrever nosso projeto</p>

{% endblock %}

- Caso não se queira algum dos blocos personalizáveis nessa nova página, basta referenciá-lo vazio. Exemplo:

{% block destaque %}

{% endblock %}



**OBS.:** O {% extends %} só importa todo o conteúdo da página, exceto as tags de carregamento {% load %}. Assim, quando reaproveitar qualquer arquivo, precisa repeti-las no novo arquivo também:

{% extends 'modelobase.html' %}

{% load static %}

**7. Python e Django 05 – Criando links para as URLs (baseado no “name” da url)**

**-**