Criação do Projeto

Inicialmente vamos digitar:

python -m venv venv

.\venv\Scripts\activate

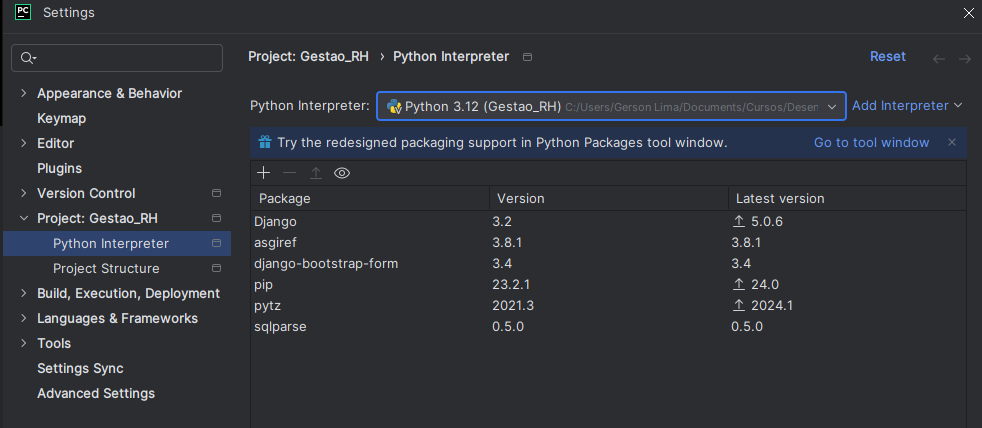
pip install Django

django-admin startproject gestao\_rh .

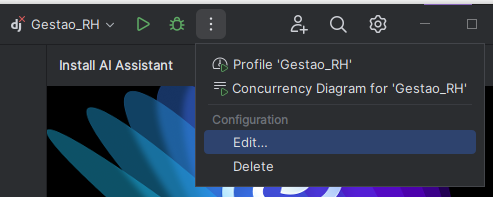
O ponto(.) serve para dizer ao Django que a estrutura deve ser criada na pasta que está rodando o código, se não, ele criará uma subpasta com o nome do projeto.

Configurando Pycharm

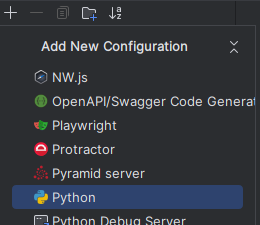
1. Configurando manage.py

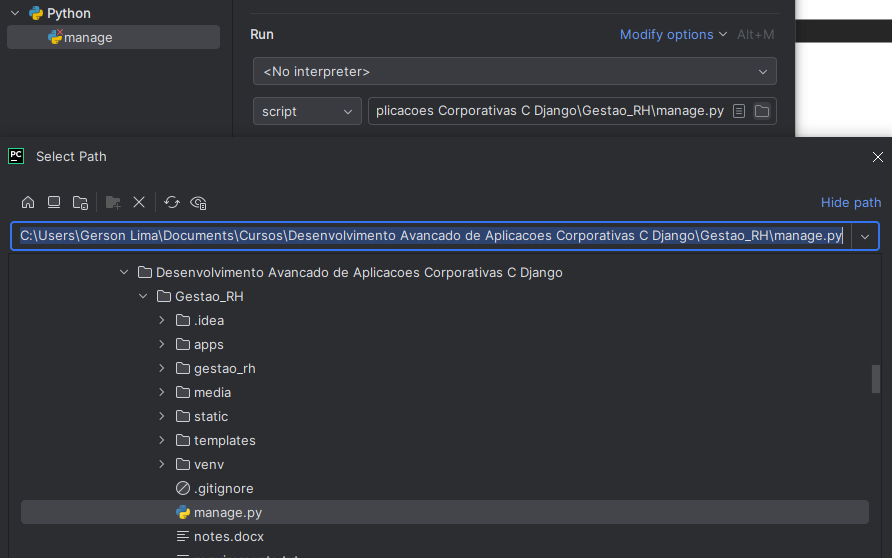


Agora, vamos no pycharm adicionar ou editar configurações:

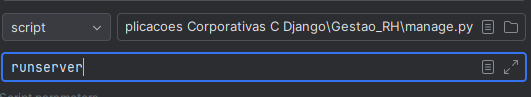


Clicar no ‘+’ e selecionar Python:

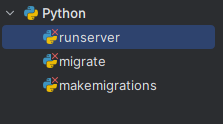


No interpretador selecionado vamos selecionar o script manage.py:  


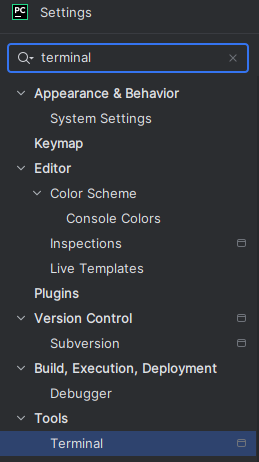
Vamos passar como parâmetro ‘runserver’:



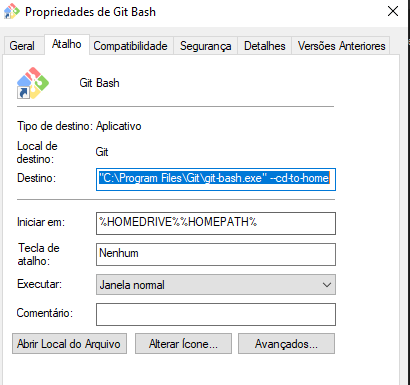
Depois vamos duplicar para os parâmetros migrate e makemigrations:



1. Git Bash

Vamos em **File>settings>terminal  
**

Pegamo o caminho do Git Bash:



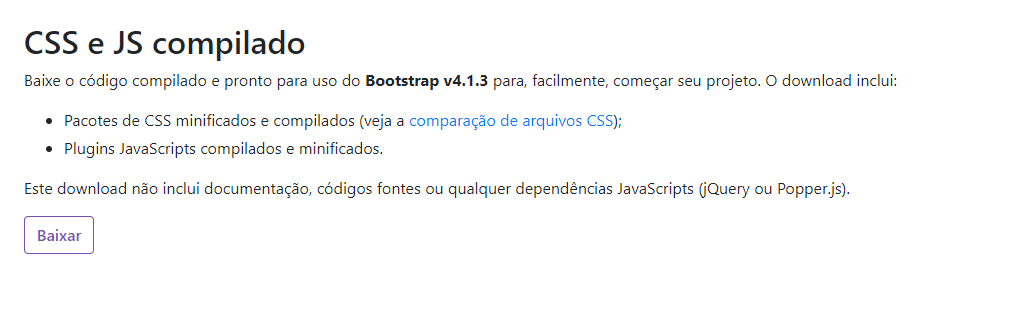
E passamos para o campo “Shell path” na aba terminal do Pycharm mas com os parâmetros –login -i:

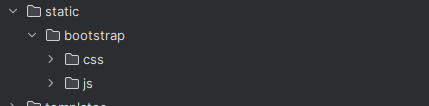


Integrando com bootstrap

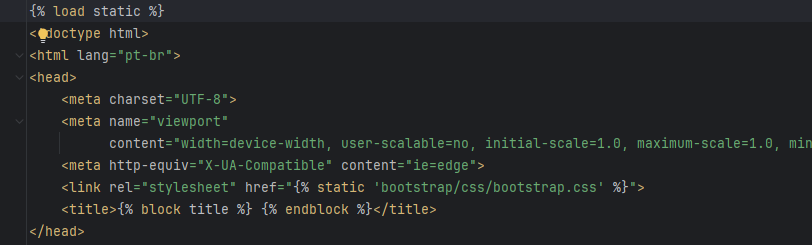
Para integrar com bootstrap primeiro vamos baixar o css e js dele no site:

<https://getbootstrap.com.br/docs/4.1/getting-started/download/>

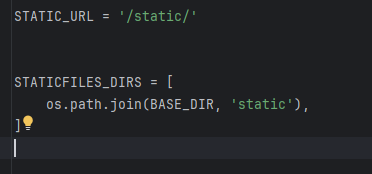


Em seguida, vamos criar uma pasta chamada “static” no diretorio raiz do projeto e jogar a pasta css e js lá:  


Feito isso, na pasta onde queremos usa-lo digitaremos {% load static %} na primeiríssima linha e na tag <link> passaremos para o href={% static ‘bootstrap/css/bootstrap.min.css’ %}



Mas para que o Django encontre-os precisaremos no setting.py passar o caminho do static:



Criando Paginas de Login e Logout

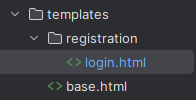
1. Login

O Django oferece nativamente urls para login, logout e etc através do caminho ‘**accouts/<ação-desejada>**’

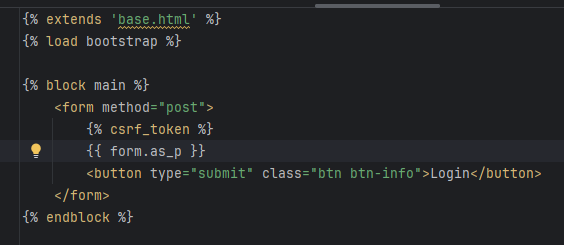
Por exemplo, vamos criar uma tela de login. Para isso, precisaremos setar a seguinte url:



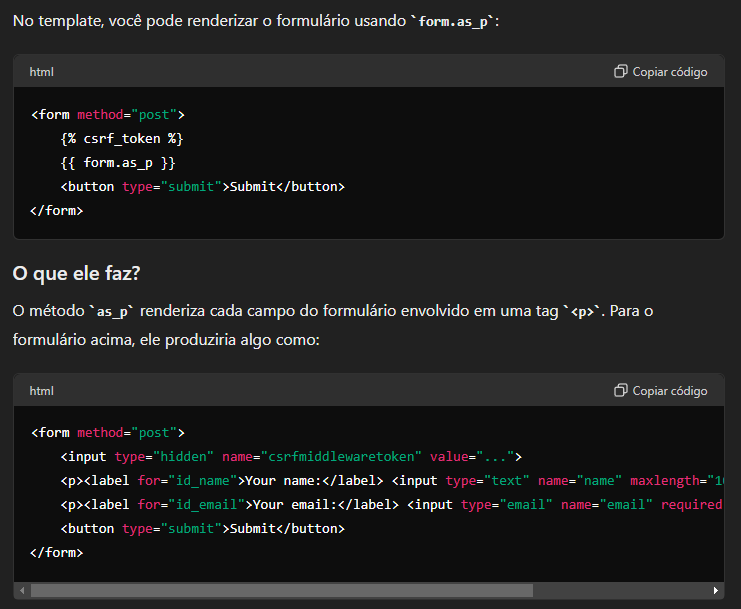
Após isso, vamos em ‘**templates>registration>**’ e criamos login.html:

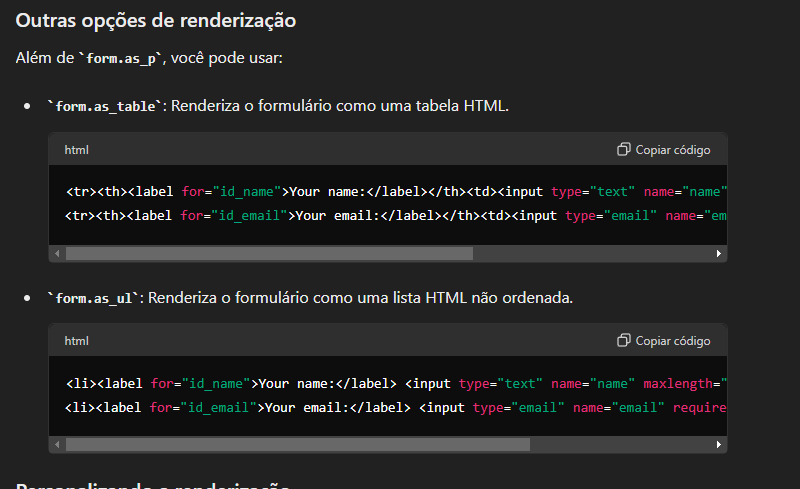


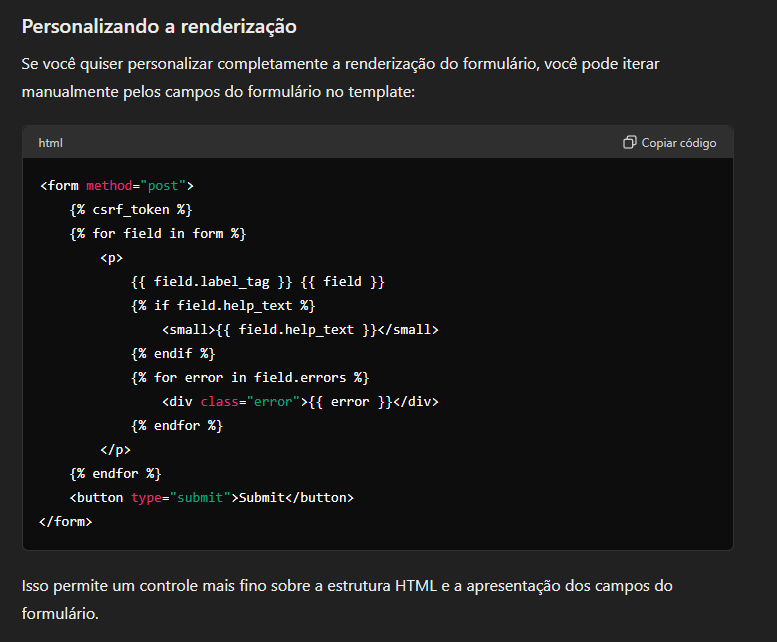
Código:



Observe que temos o **form.as\_p**, esse **form** é o Django transformando os parâmetros da classe em um formulário, e ‘**as\_p**’ você está dizendo para vir como um paragrafo **<p>**. Exemplos:



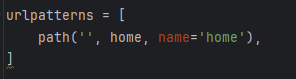




Entendido estes pontos, agora precisamos dizer ao Django que ao logar ele precisará ser redirecionado a uma determinada pagina, faremo isso usando LOGIN\_REDIRECT\_URL:



Ele só vai chamar por ‘home’ corretamente de dermos o name da url = a ‘home’. Assim:



Para garantir a segurança de que o home só será visto por quem estiver logado, user @login\_required na view:



1. Logout

Para fazer o logout é só redirecionarmos o tela do Template pra ‘logout’ da seguinte forma:

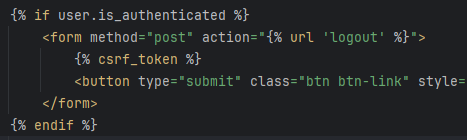


Assim como no login, você também pode fazer LOGOUT\_REDIRECT\_URL:



O ‘login’ mandará automaticamente para ‘accouts/login’

**\*ATT: Em novos testes a solução não funcionou, para isso, foi necessário fazer a seguinte adaptação, pois o Django agora só aceita logouts com requisições “post”:**

****

Class Based Views

Django inclui uma série de views genéricas pré-construídas que implementam padrões comuns, como listagem, criação, atualização e deleção de objetos. Exemplos incluem ListView, CreateView, UpdateView, DeleteView, etc.

Vamos por exemplo criar uma CreateView para empresa.

1 – Em views.py vamos criar uma classe ‘EmpresaCreate’ que recebe ‘CreateView’. As views genéricas vem de ‘**from django.views.generic.edit** **import ListView, CreateView, UpdateView, DeleteView**’.

class EmpresaCreate(CreateView):  
 model = Empresa  
 fields = ['nome']

2 – Em urls.py vamos importar a classe que criarmos e passar e chamar a função ‘**as\_view()**’:

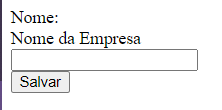
from django.urls import path  
from .views import EmpresaCreate, EmpresaEdit  
  
urlpatterns = [  
 path('novo', EmpresaCreate.as\_view(), name='create\_empresa'),  
 path('editar/<int:pk>/',  
 EmpresaEdit.as\_view(), name='edit\_empresa'),  
]

3 – No template

Feito isso, no template ‘**empresa\_form.html’** vamos passar o paremetro ‘**{{ form }}’**:

{% extends 'base.html' %}  
  
{% block main %}  
<form method="post">  
 {% csrf\_token %}  
 {{ form }}  
 <button type="submit">Salvar</button>  
</form>  
{% endblock %}

Com as 3 etapas anteriores, o Django nos dará um formulário padrão



Porém, existe mais uma etapa para quando clicar em salvar, que é no caso do formulário ser válido:

def form\_valid(self, form):  
 obj = form.save()  
 funcionario = self.request.user.funcionario  
 funcionario.empresa = obj  
 funcionario.save()  
 return HttpResponse('ok')

django-bootstrap-form

Primeiro vamos instalar o plugin:

pip install django-bootstrap-form==3.4

Como ele instalado, vamos no setting.py passar ‘**bootstrapform**’ no INSTALLED\_APPS:

INSTALLED\_APPS = [  
 ...  
 'bootstrapform',  
]

Agora, no Template precisaremos passar **{% load bootstrap %}** e onde era **{{ form }}** virará **{{ form|bootstrap }}**

Localmente os arquivos estáticos precisam estar:

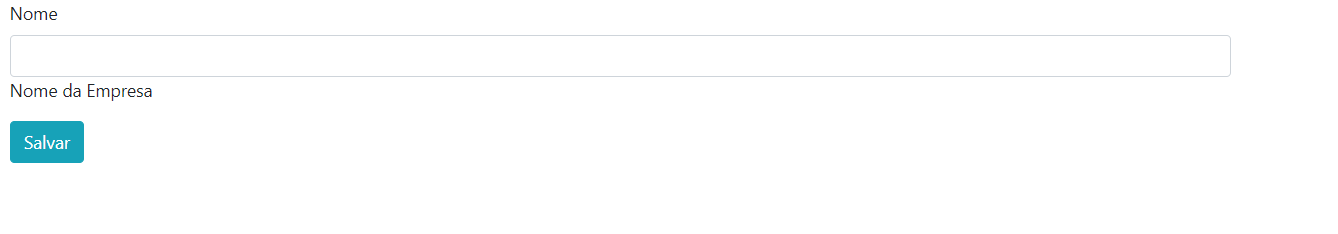
STATICFILES\_DIRS = [  
 os.path.join(BASE\_DIR, 'static'),  
]

E se mesmo assim não funcionar rodar:

python manage.py collectstatic

E assim ficará a interface:

{% extends 'base.html' %}  
{% load bootstrap %}  
  
{% block main %}  
<form method="post">  
 {% csrf\_token %}  
 {{ form|bootstrap }}  
 <button type="submit" class="btn btn-info">Salvar</button>  
</form>  
{% endblock %}



EXTRAS

1. git rm --cached

Podemos git rm --cached <arquivos> quando o arquivo foi adicionado no git ignore depois de criado e continua aparecendo no git status;