Antes de comenzar con el proyecto, modifique mi base de datos por defecto a mysql workbench, también utilicé los templates y static para el acceso a las páginas y sus decoraciones, y por último, agregue en la app installer, la aplicación de accounts para tener acceso.

Bien ahora comenzamos con el proyecto

1)Creamos una app llamada ‘accounts’ y desde ahi, agregamos urls de register y login

1. *def* registerPage(*request*):
2. context ={}
3. return render(request,'register.html',context)
4. *def* loginPage(*request*):
5. context ={}
6. return render(request,'login.html',context)

2) configuramos estas urls desde el el archivo maestro ‘validacion’

from django.contrib import admin

from django.urls import path

from accounts import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('register/', views.registerPage, *name*='register'),

    path('login/', views.loginPage, *name*='login'),

3)utilizamos la libreria:

from django.contrib.auth.forms import UserCreationForm

y modificamos la def de ‘registerPage’

*def* registerPage(*request*):

    form = UserCreationForm()

    if request.method=='POST':

        form = UserCreationForm(request.POST)

        if form.is\_valid():

            form.save()

    context ={'form':form}

    return render(request,'register.html',context)

y además, lo mostramos en el html de ‘register.html’

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *http-equiv*="X-UA-Compatible" *content*="IE=edge">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>register</title>

</head>

<body>

    <form *method*="post" *action*="">

        {% csrf\_token %}

        {{form.as\_p}}

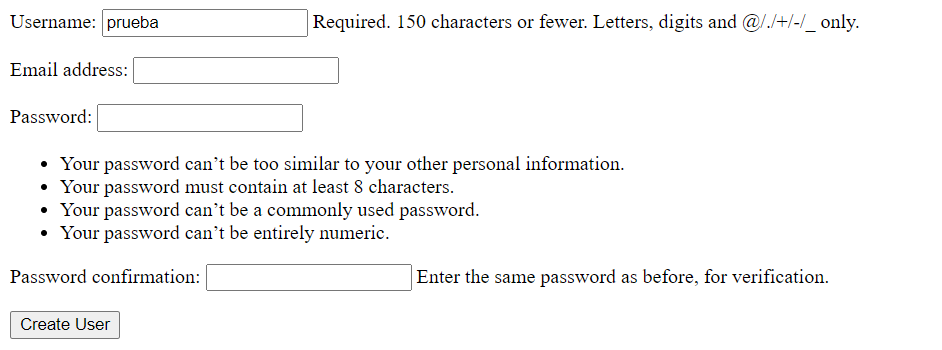
        <input *type*="submit" *value*="Create User"*name*="">

    </form>

</body>

</html>

**CON ESTA INFORMACION, YA PODEMOS CREAR UN USUARIO BASICO DESDE EL FORMULARIO CREATEUSER**

****

4) Ahora pasamos a crear el forms.py con las siguientes librerías e información:

from django.forms import ModelForm

from django.contrib.auth.forms import UserCreationForm

from django import forms

from django.contrib.auth.models import User

*class* CreateUserForm(*UserCreationForm*):

*class* Meta:

        model = User

        fields =['username', 'email', 'password1', 'password2']

con esto, pasamos a modificar el campo de UserCreationForms

4) ahora reflejamos estos cambios dentro de las views

from .forms import CreateUserForm

*def* registerPage(*request*):

    form = CreateUserForm()

    if request.method=='POST':

        form = CreateUserForm(request.POST)

        if form.is\_valid():

            form.save()

    context ={'form':form}

    return render(request,'register.html',context)

5) para que no nos salga todas las recomendaciones de la creación de un usuario, modificamos nuestro “register.html” con la siguiente información

<body>

    <form *method*="post" *action*="">

        {% csrf\_token %}

        {{form.username.label}}

        {{form.username}}

        {{form.email.label}}

        {{form.email}}

        {{form.password1.label}}

        {{form.password1}}

        {{form.password2.label}}

        {{form.password2}}

        <input *type*="submit" *value*="Create User"*name*="">

    </form>

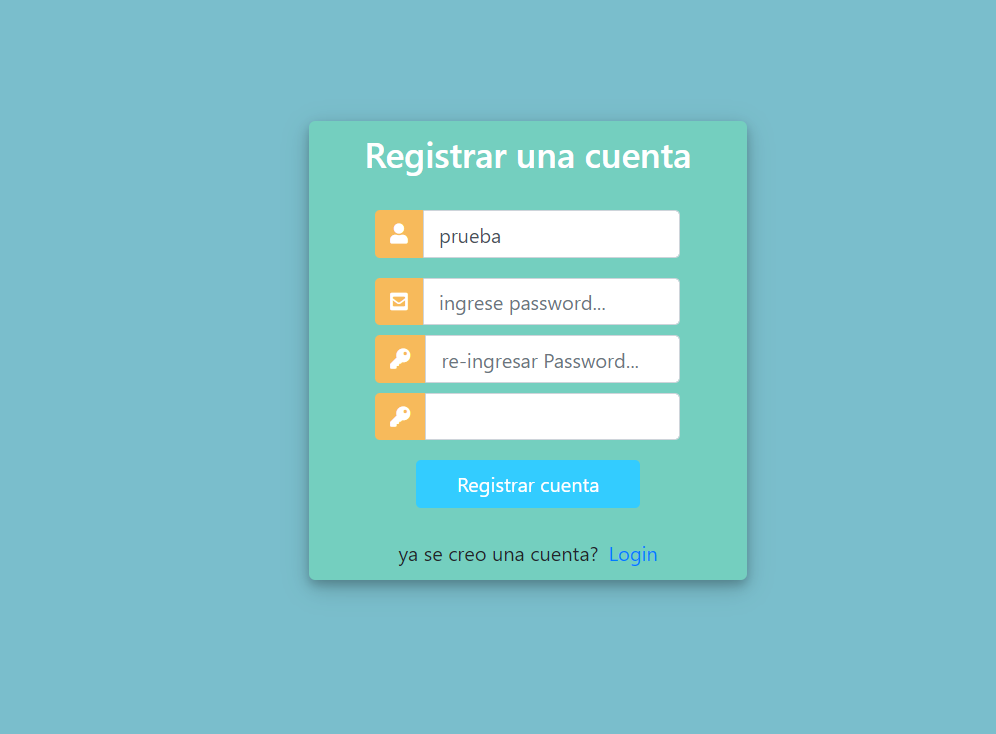
</body>

</html>

De forma visual:



Ahora usamos la magia del css y javascript para transfórmalo asi:



6) una vez finalizado la parte del registro, continuamos con el login. Modificamos nuestro html usando nuevamente la magia del css y javascript.

Ahora modificamos desde las views, el método de ‘RegisterPage’ para redireccionar. Para esto, usamos la librería:

from django.shortcuts import render, redirect

y ahora la modificación del método:

*def* registerPage(*request*):

    form = CreateUserForm()

    if request.method=='POST':

        form = CreateUserForm(request.POST)

        if form.is\_valid():

            form.save()

            return redirect('login')

    context ={'form':form}

    return render(request,'register.html',context)

7) Para completar el registro, usamos una variable de Django para que nos tire un error en cualquier campo que consideremos invalido, gracias a que este framework tiene sus librerías y métodos predefinidos:

{{form.errors}}

8)

Ahora para tener una mejor visualización con respecto a los mensajes, agregamos la siguiente librería:

from django.contrib import messages

y nuevamente modificamos nuesta función “registerPage”:

*def* registerPage(*request*):

    form = CreateUserForm()

    if request.method=='POST':

        form = CreateUserForm(request.POST)

        if form.is\_valid():

            form.save()

            user = form.cleaned\_data.get('username')

            messages.success(request, 'la carga ha sido exitosa' + user)

            return redirect('login')

    context ={'form':form}

    return render(request,'register.html',context)

con esto, obtenemos la instancia del mensaje, pero ahora nos falta mostrarlo en nuestra pagina, entonces ponemos le siguiente comando

{% for message in messages %}

           <p *id*="messages">{{message}}</p>

{% endfor %}

Dentro de nuestro login.html

**AHORA EMPEZAMOS LA PARTE DE LA AUTENTIFICACION**

9) Primero que nada, cargamos las siguientes librerías:

from django.contrib.auth import authenticate,login, logout

Ahora editamos nuestra function ‘LoginPage’

*def* loginPage(*request*):

    if request.method == 'POST':

        username = request.POST.get('username')

        password = request.POST.get('password')

        user = authenticate(request, *username*=username, *password*=password)

if user is not None:

            login(request, user)

            redirect('home')

    context ={}

    return render(request,'login.html',context)

con esta informacion, lo que hacemos es auntenficarnos con los usuarios que tenemos creados, usamos los metodos POST.get para obtener tanto el usuario, como el password, una vez obtenido, mandamos un user con el método de auntenficacion, y preguntamos que si el usuario ‘Si existe’ ( is not none) nos loguemos y además, nos redireccione en la home page.

En caso de que sea incorrecto, agregamos un “messages” con la siguiente sentencia y que además, nos redireccione a la misma pagina

else:

      messages.info(request,'el usuario o la contra, son invalidos')

y agregamos un token dentro de login.html

{% csrf\_token %}

10) Nuestro nuevo objetivo será que nos aparezca como logueado, para eso creamos una función llamada ‘logoutUser’

*def* logoutUser(*request*):

    logout(request)

    return redirect('login')

y los agregamos en las urls

path('logout/', views.logoutUser, *name*='logout'),

bien, ahora que definimos nuestro logout, necesitamos mostrarlo en nuestro html, con la siguiente información:

  <p *class*="nav-item">hello, {{request.user}}</p>

 <a *class*="nav-link" *href*="{% url 'logout' %}">Logout</a>

**opcional**

11) Importamos unos decoradores para el login. Esta opción nos permite restringir el contenido de nuestra pagina para que se necesite loguear en cada de que quiera acceder

from django.contrib.auth.decorators import login\_required

@login\_required(*login\_url*='login')

Y abajo, nuestras funciones de las páginas que queramos restringir

**Permisos y auntentificaciones**

1)Primero empezamos creando un archivo py, llamado decorators.py. Dentro de esta, agregamos las librerías:

from django.http import HttpResponse

from django.shortcuts import redirect

creamos una función que nos permita usarla como decorador para las paginas.

*def* unaunthenticated\_user(*view\_func*):

*def* wrapper\_func(*request*, \**args*, \*\**kwargs*):

        if request.user.is\_authenticated:

            return redirect('home')

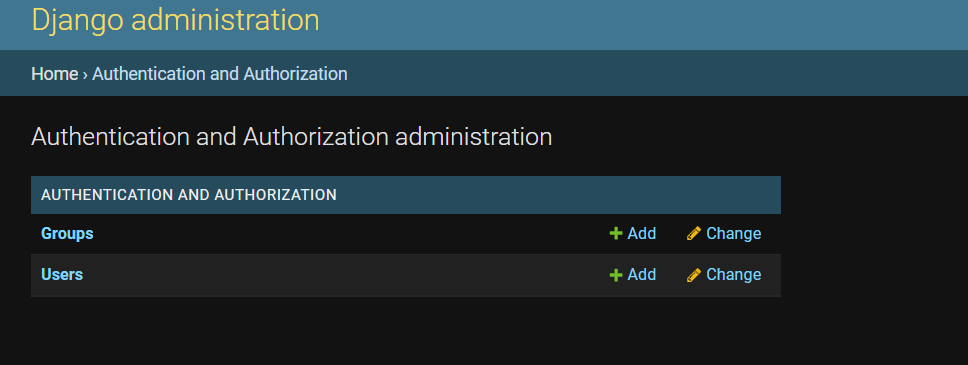
        else:

            return view\_func(request, \*args, \*\*kwargs)

    return wrapper\_func

Esto sirve para que cuando nos loguemos, no nos vuelva a aparece al hacer click o ingresar a la página de login, su contenido. Esto es redundante, ya que.¿Para que queremos loguearlos, si ya lo estamos?

2) paso a mostrar los distintos usuario que tengo cargado en mi proyecto



En este paso nos basamos en los roles que tendrá cada usuario creado en nuestra base de datos, para esto, definimos en el decorators.py la siguiente función:

*def* allowed\_user(*allowed\_roles*=[]):

*def* decorators(*view\_func*):

*def* wrapper\_func(*request*, \**args*, \*\**kwargs*):

            group= None

            if request.user.groups.exists():

                group= request.user.groups.all()[0].name

            if group in allowed\_roles:

                return view\_func(request, \*args, \*\*kwargs)

            else:

                return HttpResponse('no esta permitido entrar a esta pagina')

        return wrapper\_func

    return decorators

y cargamos esta función de la parte de las views.py

from .decorators import unaunthenticated\_user,allowed\_user

y su metodo:

@allowed\_user(*allowed\_roles*=['admin'])

En este caso puse admin como el unico que puede ver esta funcion sobre una pagina

ahora tenemos cargado la función y podemos usarla como método para visualizar las vistas que podran tener los distintos tipos de usuarios, por ejemplo:

@allowed\_user(*allowed\_roles*=['admin'])

*def* home(*request*):

    return render(request,'home.html')

en caso de no ser el usuario cargado, nos mandara un error:

