SEÇÃO 07 – DISSECANDO O DJANGO USER MODEL

Entendendo o Django User Model:

Quando uma aplicação Django é criada, tem todo um sistema de manutenção que é criado automaticamente pelo Framework. Entretanto, esse modelo pode ser integrado a outros, ou modificado.

O projeto irá se chamar “Django User Model”

```
pip3 install django
django-admin startproject djangousermodel .
```

Para entender o funcionamento do User Model:

```
python manage.py shell
python manage.py migrate
```

Fazendo uso do Django Shell:

```
from django.contrib.auth.models import User
```

O que é o Model User?

É o módulo models e dentro desse modo é importado os modelo de dados. Ou seja, é o modelo de dados.

dir(User) ou **help(User)** para saber o que pode ser usado, ou olhar a documentação necessária.

```
help(manageruser.create)
```

Pode usar o método create user ou create superuser.

Criando um usuário:

```
>>> usuario = User.objects.create_user(username='teste', password='123456', email='test@gmail.com')
>>> usuario
<User: teste>
>>> usuario.save()
```

Como consultar um usuário:

```
ret = User.objects.all()
ret
<QuerySet [<User: teste>]>
>>> ret[0].username
'teste'
>>> ret[0].password
'pbkdf2_sha256$260000$r1nvLTJn6YnwHkpOcCWUhd$YQj kWMvriWB3J+5LWAkmB3kMwpap4SaOjs2NQrT2Vs='
```

OBS: Ou seja, o Django criptografou a senha que era “123456”.

Se criptografar o 123456 com Django fica:
'pbkdf2_sha256$260000$r1nvLTJn6YnwHkpOcCWUhd$YQj/kWMvriWB3J+5LWAkmB3kMwpap4SaOjs2NQrT2Vs='

Um Staff pode logar na área administrativa, mas não pode visualizar ou modificar nada. É preciso criar um superuser. Ou seja, um usuário pode ser staff, mas não necessariamente é um super usuário.

ESTENDENDO O DJANGO USER MODELS

O projeto irá se chamar Django User Model 2

```
django-admin startproject djangousermodel2 .
Django-admin startapp core
```

Fazer todas as configurações básicas do settings.py
Criar um models na aplicação core
Isso inclui:

- Permitir todos usuários [‘*’]
- Indicar o novo app: ‘core’
- Mudar o Banco de Dados se necessário
- Mudar a região “America/Sao_Paulo”
- E fazer os Medias e Statics Root

Existem 3 formas de fazer a aplicação para integração:

1ª Forma:

```
from django.contrib.auth.models import User

class Post(models.Model):
    autor = models.ForeignKey(User, verbose_name='Autor', on_delete=models.CASCADE)
    titulo = models.CharField('Titulo', max_length=100)
    texto = models.TextField('Texto', max_length=400)

    def __str__(self):
        return self.titulo
```

O User presente nessa forma, é o próprio do Django o que dificulta a customização da página administrativa.

2ª Forma:

A 2ª forma é utilizada quando fazemos a customização da área administrativa do Web Site:

Primeiro há a necessidade de fazer algumas modificações no settings.py:

```
AUTH_USER_MODEL = 'usuarios.CustomUsuario'
```

Ou seja, ocorre a substituição do módulo de usuário padrão para o criado customizado.

E no arquivo models:

```
from django.db import models

from django.conf import settings


class Post(models.Model):
    autor = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name='Autor',
on_delete=models.CASCADE)
    titulo = models.CharField('Titulo', max_length=100)
    texto = models.TextField('Texto', max_length=400)

    def __str__(self):
        return self.titulo
```

Ou seja, precisa haver a indicação de mudança para o padrão customizado indicado anteriormente no arquivo de settings.

3ª Forma:

Obs: É a forma mais indicada para introduzir uma área administrativa customizada.

O módulo de autentificação do Django tem um get_user_model para a executar a função customizada. Se tiver sobrescrito, irá trazer a forma customizada, caso contrário ele irá exibir a área administrativa oficial.

```
from django.db import models

from django.contrib.auth import get_user_model


class Post(models.Model):
    autor = models.ForeignKey(get_user_model(), verbose_name='Autor', on_delete=models.CASCADE)
    titulo = models.CharField('Titulo', max_length=100)
    texto = models.TextField('Texto', max_length=400)

    def __str__(self):
        return self.titulo
```

Ao acessar o módulo com `python manage.py runserver`

O modulo irá ser exibido.

Registrar no Admin.py o modelo modificado criado em models,

```
from django.contrib import admin

from .models import Post


@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('titulo', 'autor')
```

Como fazer para exibir nome e sobrenome na área administrativa:

Existe um método em user chamado “get_full_name”

Então:

```
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('titulo', '_autor')

    def _autor(self, instance):
        return f'{instance.autor.get_full_name()}'
```

Depois dessas alterações, na área administrativa irá aparecer o Nome Completo ao invés do nome de usuário utilizado para fazer login.

Imagina que há um sistema com vários usuários, e você não quer que os demais tenham acesso aos dados dos demais usuários.

Como fazer para cada usuário observar somente os próprios Posts?

```
def get_queryset(self, request):
    qs = super(PostAdmin, self).get_queryset(request)
    return qs.filter(autor=request.user)
```

Essa modificação é feita no Admin.py e ela entende que é pra mostrar somente os dados do usuário que está logado, tudo relacionado a outro usuário, fica oculto.

Para ter acesso ao model:
Pressione CTRL e clique no model.

Como pegar somente o dado da sessão que tá logado e ocultar a exibição de usuários na hora de fazer um novo Post.

```
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('titulo', '_autor')
    exclude = ['autor', ]
```

Isso irá excluir a lista de autores que aparecia no módulo administrativo antes. Porém haverá um erro na hora de salvar o novo post. Portanto é preciso fazer uma alteração no modo salvar do Post.

```
def save_model(self, request, obj, form, change):
    obj.autor = request.user
    super().save_model(request, obj, form, change)
```

No admin.py essa alteração significa que a pessoa que está logada é a autora do Post. Dessa forma, o erro de salvar para de ocorrer.

Como alterar o textos que indicam o nome da parte de administração:

No arquivo de URLS do Projeto:

```
urlpatterns = [
    path('admin/', admin.site.urls),
]

admin.site.site_header = 'Geek University'
admin.site.site_title = 'Evolua seu lado geek!'
admin.site.index_title = 'Sistema de Gerenciamento de Posts'
```

Todas as alterações serão mostradas na sessão administrativa.

CRIANDO UM USER MODEL CUSTOMIZADO

O novo projeto de modificação de área Administrativa irá se chamar “Django User Model 3”

```
django-admin startproject djangousermodel3 .
django-admin startapp usuarios
```

A aplicação desse projeto é chamada de usuários.

Para logar em um site, o django pede nome de usuário e senha, isso não é interessante, portanto o indicado é pedir e-mail e senha. E em outros casos pode ser CPF e senha ou telefone e senha.

Aqui entra o Model User customizado.

Existem 2 formas de criar um usuário customizado:

1ª Forma:

```
from django.db import models

from django.contrib.auth.models import AbstractBaseUser


class CustomUsuario(AbstracBasetUser):
    pass

    def __str__(self):
        return self.email
```

2ª Forma:

Todo o procedimento idêntico, entretanto, utilizando “AbstractUser”

Qual a diferença de AbstractBaseUser para AbstractUser?

Ambos tratam de usuário abstrato. Entretanto, o Base é um usuário mais básico. Ou seja, faltam funcionalidades, que precisam ser implementadas manualmente.

Obs: Portanto o recomendado é utilizar o AbstractUser

Será necessário utilizar um Gerenciador de Usuário:

No arquivo models.py:

```
from django.db import models
from django.contrib.auth.models import AbstractUser, BaseUserManager


class UsuarioManager(BaseUserManager):
    use_in_migrations = True

    def _create_user(self, email, password, **extra_fields):
        if not email:
            raise ValueError('O e-mail é obrigatório')
        email = self.normalize_email(email)
        user = self.model(email=email, username=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, email, password=None, **extra_fields):
        # extra_fields.setdefault('is_staff', True) O recomendado é deixar como False, padrão do Django
        extra_fields.setdefault('is_superuser', False)
        return self._create_user(email, password, **extra_fields)

    def create_superuser(self, email, password, **extra_fields):
        extra_fields.setdefault('is_superuser', True)
        extra_fields.setdefault('is_staff', True)

        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser precisa ter is_superuser=True')

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser precisa ter is_staff=True')

        return self._create_user(email, password, **extra_fields)


class CustomUsuario(AbstractUser):
    email = models.EmailField('E-mail', unique=True)
    fone = models.CharField('Telefone', max_length=15)
    is_staff = models.BooleanField('Membro da Equipe', default=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name', 'fone']

    def __str__(self):
        return self.email

    objects = UsuarioManager()
```

No **create_user** que indica o usuário padrão, o recomendado é deixar o setUp de staff como *False*. E deixar como True somente nas configurações de Super User que indica o usuário administrador.

O campo **objects=** no final do código: Indica que todo o processo vai ser gerenciado pelo UsuárioManager.

Finalizando com o models.py, é preciso criar dois formulários para preenchimento da sessão de usuário.

Dentro do diretório da aplicação usuários criar um arquivo **forms.py**

No forms.py:

Serão criados os formulários de criação do usuário (UserCreateForm) e de alteração de usuário (UserChangeForm)

```
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from .models import CustomUsuario


class CustomUsuarioCreateForm(UserCreationForm):

    class Meta:
        model = CustomUsuario
        fields = ['first_name', 'last_name', 'fone']
        labels = {'username': 'Username/E-mail'}

        def save(self, commit=True):
            user = super().save(commit=False)
            user.set_password(self.cleaned_data["password1"])
            user.email = self.cleaned_data["username"]
            if commit:
                user.save()
            return user


class CustomUsuarioChangeForm(UserChangeForm):

    class Meta:
        model = CustomUsuario
        fields = ['first_name', 'last_name', 'fone']
```

Esses são os dois formulários necessários, um para criação e outro que permite alterar os dados.

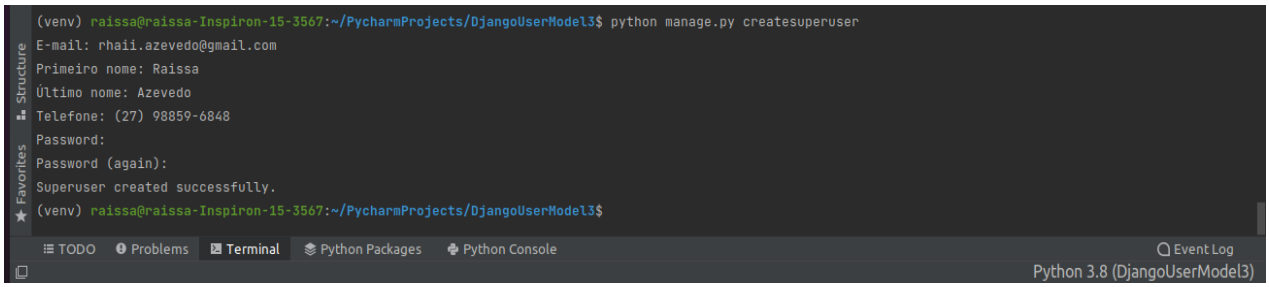
Em seguida, registrar todos os models no Admin.py:

Fazer o import de tudo que for necessário para a administração do usuário, uma vez que é na página de administração que as alterações estão sendo feitas.

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin

from .forms import CustomUsuarioCreateForm, CustomUsuarioChangeForm
from .models import CustomUsuario


@admin.register(CustomUsuario)
class CustomUsuarioAdmin(UserAdmin):
    add_form = CustomUsuarioCreateForm
    form = CustomUsuarioChangeForm
    model = CustomUsuario
    list_display = ['first_name', 'last_name', 'fone', 'is_staff']
    fieldsets = (
    (None, {'fields': ('email', 'password')}),
    ('Informações Pessoais', {'fields': ('first_name', 'last_name', 'fone')}),
    ('Permissões', {'fields': ('is_active', 'is_staff', 'is_superuser', 'groups', 'user_permissions')}),
    ('Datas Importantes', {'fields': ('last_login', 'date_joined')}),
)
```

O **fieldsets** são todas as configurações padrões do Django que pedem os dados de cadastro do usuário. Há outros dados que podem ser acrescentados.

As mudanças já ocorrem dentro do próprio terminal:



LOGIN E AUTENTICAÇÃO

Quer uma nova tela de autentificação, que seja diferente da oferecida pelo Django. O projeto utilizado será o mesmo.
Ir na parte de URLS:

```
from django.contrib import admin
from django.urls import path, include
from django.views.generic.base import TemplateView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('contas/', include('django.contrib.auth.urls')),
    path('', TemplateView.as_view(template_name='index.html'), name='index'),
]
```

Em seguida é preciso criar uma estrutura de templates para as telas de login. Dessa vez, o diretório de templates precisa ser criado na raiz do projeto, ao invés da aplicação como nos projetos anteriores. Porque o que vai ser modificado é a parte administrativa, e não o site.
Então, criar um diretório de templates na raiz do projeto.
Criar um arquivo html denominado **base.html** e fazer as modificações necessárias.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Geek University</title>
    <meta name="viewport" content="widthdevice=width, initial-scale=1, shrink-to-fit-no">
    <link rel="icon" href="https://www.geekuniversity.com.br/static/images/favicon.4fcb819d32bf.ico">
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet"
id="bootstrap">
    <link href="https://getbootstrap.com/docs/4.0/examples/sign-in/signin.css" rel="stylesheet">
</head>
<body class="text-center">
    {% block content %}
    {% endblock %}
    <script src="http://maxcdn.bootstrapcdn.com/bootstrap/4.00/js/bootstrap.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
</body>
</html>
```

Garantir ter adicionado o favicon, o css e o Javascript. Por se tratar da página base. Adicionar os blocos de conteúdo no body padrões do Django.
Dentro do diretório templates, criar o **index.html**

Nas configurações da pagina “index” indicar que se o usuário for anônimo colocar uma área para Login
E se o usuário tiver autenticado exibir a mensagem de “Bem Vindo” e o botão de logoff

```
{% extends 'base.html' %}
{% block content %}
  <div class="container">
    <h1>Geek University</h1>
    {% if user.is_anonymous %}
      <a class="btn btn-primary" href="{% url 'login' %}">Login</a>
    {% else %}
      <div class="alert alert-primary" role="alert">
        Seja bem vindo(a), {{ user.get_full_name }}!
      </div>
      <a class="btn btn-primary" href="{% url 'logout' %}">Logout</a>
    {% endif %}
  </div>
{% endblock %}
```

Por fim, é necessário criar a página de Login:
Dentro do diretório de templates, criar o diretório “registration” criar o arquivo **login.html**

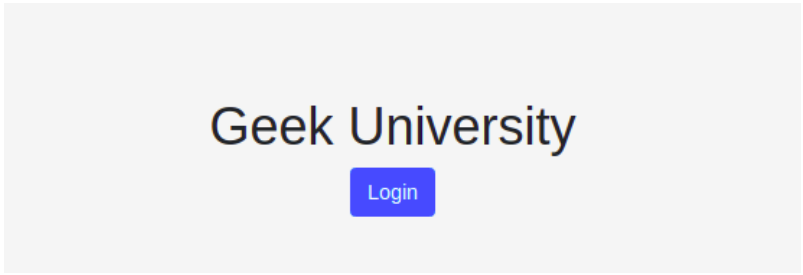
```
{% extends 'base.html' %}
{% block content %}
  <form class="form-signin" method="post" autocomplete="off">
    {% csrf_token %}
    <img class="mb-4" src="https://www.geekuniversity.com.br/static/images/favicon.4fcb819d32bf.ico" alt="" 
width="72" height="72">
    <h1 class="h3 mb-3 font-weight-normal">Informe seus dados</h1>

    <label for="username" class="sr-onlu">E-mail</label>
    <input type="email" id="username" name="username" class="form-control" placeholder="Informe seu e-mail" 
required autofocus>

    <label for="password" class="sr-only">Senha</label>
    <input type="password" id="password" name="password" class="form-control" placeholder="Informe sua senha" 
required>

    <button class="btn btn-lg btn-primary btn-block" type="submit">Acessar</button>
    <p class="mt-5 mb-3 text-muted">&copy; {% now 'Y' %}</p>
  </form>
{% endblock %}
```

Fazer os ajustes de redirecionamento de login no settings.py

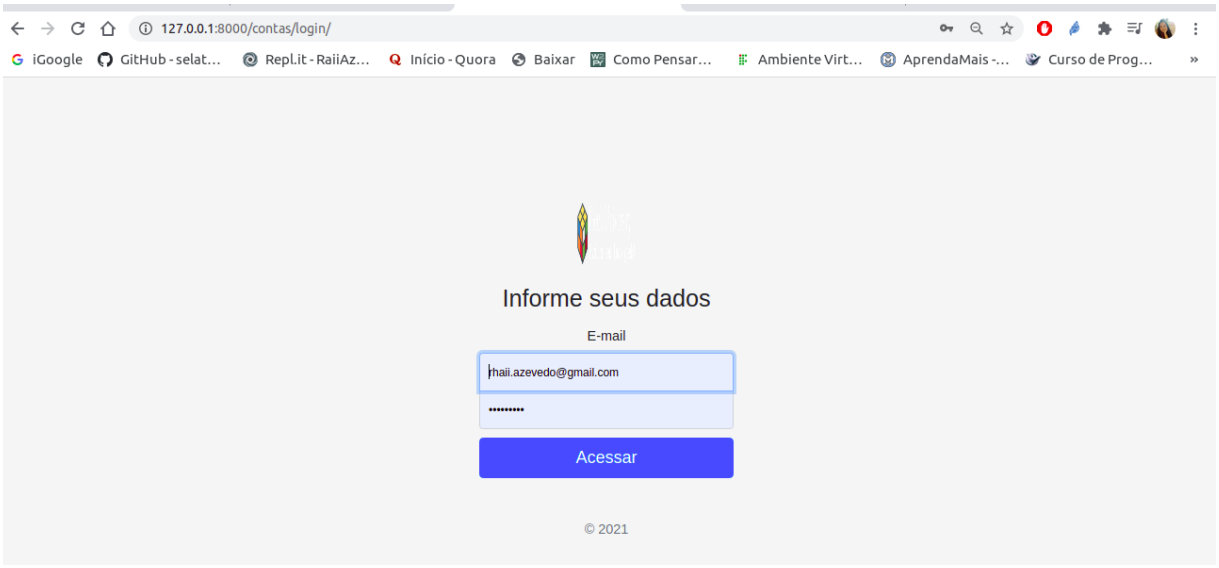
```
LOGIN_REDIRECT_URL = 'index'
LOGOUT_REDIRECT_URL = 'index'
```

Essas configurações indicarão que volte para a pagina principal automáticamente.

Acessando o resultado das configurações de Página administrativa customizada:



Irá pedir o Login:



E depois irá exibir a página de Boas Vindas como indicado.