# Tarea 4 - Django Blog

Electiva Desarrollo de Software

Gustavo Adolfo Ibarra Piandoy

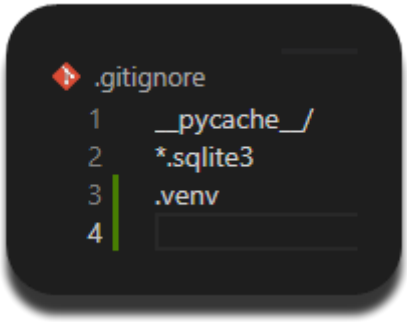
2024-07-20

## Django Blog

### Enviroment

```
# Creando carpeta del proyecto
mkdir django_blog
cd django_blog

# Activando ambiente
virtualenv .venv
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope Process
.\.venv\Scripts\activate

# Django
pip install django
django-admin startproject django_project .

# Creando repositorio
git init
code .gitignore
```

#### Añadiendo archivos a excluir




.gitignore.png

```
# Primer commit
git status
```

```
git add -A
git commit -m "First commit"
```

## Custom User Model

```
py manage.py startapp accounts
```

Añadiendo a la configuración **django_project/settings.py**

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # local
    "accounts.apps.AccountsConfig",
]
```

installed_apps.png

Modificando el archivo de modelo **accounts/models.py**

```
accounts > models.py > CustomUser
1  from django.contrib.auth.models import AbstractUser
2  from django.db import models
3
4  class CustomUser(AbstractUser):
5      name = models.CharField(null=True, blank=True, max_length=100)
```

Añadiendo **AUTH_USER_MODEL** en **django_project/settings.py**

```
AUTH_USER_MODEL = "accounts.CustomUser"
```

Aplicamos los cambios al modelo

```
py manage.py makemigrations
py manage.py migrate
py manage.py createsuperuser
```

Creando el archivo **accounts/forms.py**

```python
accounts > forms.py > ...
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from .models import CustomUser

class CustomUserCreationForm(UserCreationForm):

    class Meta(UserCreationForm):
        model = CustomUser
        fields = UserCreationForm.Meta.fields + ("name",)

class CustomUserChangeForm(UserChangeForm):

    class Meta:
        model = CustomUser
        fields = UserChangeForm.Meta.fields
```

Modificamos el archivo **accounts/admin.py**

```python
accounts > admin.py > ...
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .forms import CustomUserCreationForm, CustomUserChangeForm
from .models import CustomUser


class CustomUserAdmin(UserAdmin):

    add_form = CustomUserCreationForm
    form = CustomUserChangeForm
    model = CustomUser
    list_display = [
        "email",
        "username",
        "name",
        "is_staff",
    ]

    fieldsets = UserAdmin.fieldsets + ((None, {"fields": ("name",)}),)
    add_fieldsets = UserAdmin.add_fieldsets + ((None, {"fields": ("name",)}),)

admin.site.register(CustomUser, CustomUserAdmin)
```

## Posts App

```
py manage.py startapp posts
```

Modificamos el archivo **django_project/settings.py**

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # local
    "accounts.apps.AccountsConfig",
    "posts.apps.PostsConfig",
]
```

installed_app.png

**Post Model**

Modificamos el archivo **posts/models.py**

```python
posts > models.py > ...
from django.conf import settings
from django.db import models

class Post(models.Model):

    title = models.CharField(max_length=50)
    body = models.TextField()
    author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

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

posts_models.png

Actualizamos los cambios en los modelos

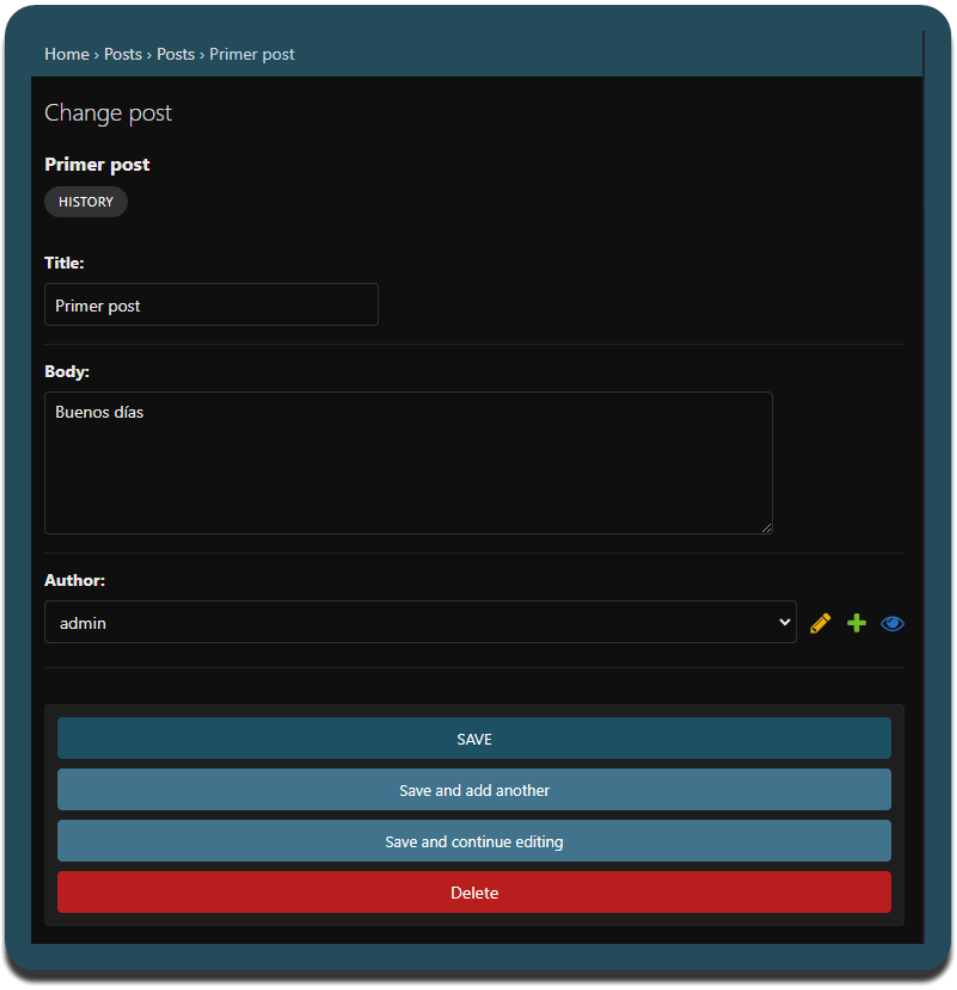
```
py manage.py makemigrations posts
py manage.py migrate
```

Modificamos el archivo **posts/admin.py**

```python
posts > admin.py
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

posts_admin.png

## Creando un post




creando post.png

## Test

Modificando el archivo **post/test.py**

```python
posts > tests.py > BlogTests > test_post_model
from django.contrib.auth import get_user_model
from django.test import TestCase
from .models import Post

class BlogTests(TestCase):
    @classmethod
    def setUpTestData(cls):
        cls.user = get_user_model().objects.create_user(
            username="testuser",
            email="test@email.com",
            password="secret",
        )
        cls.post = Post.objects.create(
            author=cls.user,
            title="A good title",
            body="Nice body content",
        )

    def test_post_model(self):
        self.assertEqual(self.post.author.username, "testuser")
        self.assertEqual(self.post.title, "A good title")
        self.assertEqual(self.post.body, "Nice body content")
        self.assertEqual(str(self.post), "A good title")
```

post test.png

Validando los el test

```
py manage.py test
```

```
(.venv) PS C:\Users\xp\Documents\Estudios\Web\Django\django_blog> py .\manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.273s

OK
Destroying test database for alias 'default'...
```

run test.png

## Configuración Docker

Crear el archivo **Dockerfile**

```
Dockerfile > ...
# Pull base image
FROM python:3.12.4-slim-bullseye

# Set environment variables
ENV PIP_DISABLE_PIP_VERSION_CHECK=1
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

# Set work directory
WORKDIR /code

# Install dependencies
COPY ./requirements.txt .
RUN pip install -r requirements.txt

# Copy project
COPY . .
```

Dockerfile.png

Crear el archivo **docker-compose.yml**

```
docker-compose.yml
version: "3.9"
services:
  web:
    build: .
    command: python /code/manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - 8000:8000
    depends_on:
      - db
  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - "POSTGRES_HOST_AUTH_METHOD=trust"
volumes:
  postgres_data:
```

docker-compose.png

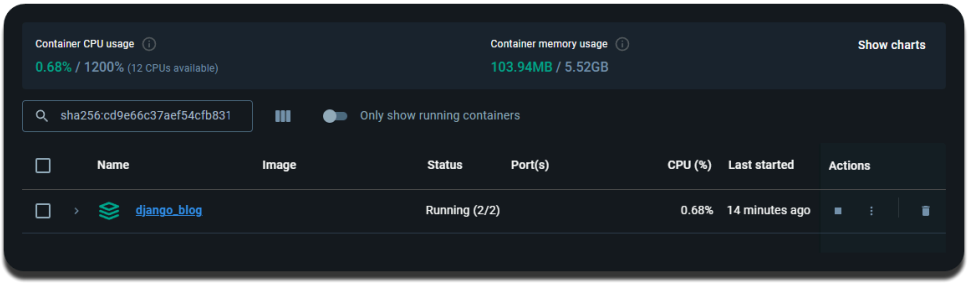
instalamos en el ambiente la librería que sirve para conectar python con postgres

```
# Activando el ambiente
py install psycopg2-binary==2.9.9
pip freeze > requirements.txt
```

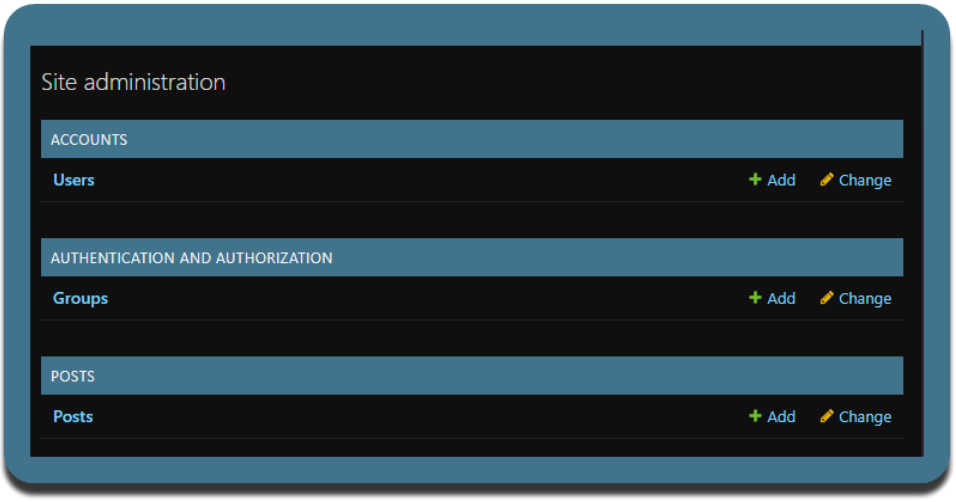
Ejecutamos comandos Docker

```
# creamos la imagen
docker build .

# Alzamos los contenedores
docker-compose up -d

# Realizamos las migraciones
docker-compose exec web python manage.py migrate

# Creamos el usuario admin
 docker-compose exec web python manage.py createsuperuser
```

Validamos el blog ingresando al usuario




Docker_Desktop.png




puser_groups_posts.png

## Respositorio

link