



MATERIA

Ingeniería Web.

ALUMNO

*Fragoso Martínez Vanessa Alin
Santana Islas Gerardo Leonardo*

TEMA

Construcción y Diseño del Sistema.

Grupo: 2TM3

Fecha de entrega:

16/06/23

Ficha del documento

Fecha	Revisión	Autor	Verificado dep. calidad.
16/06/23	1°	Fragoso Martínez Vanessa Alin. Santana Islas Gerardo Leonardo.	



Configura un repositorio de Git en tu proyecto.

Para instalar Django necesitamos de un gestor de dependencias o en otras palabras de un instalador de paquetes, y el más popular en Python es **PIP** (Package Installer for Python) ya que viene integrado junto con Python en el momento de la instalación.

Si vienes de JavaScript de seguro conoces **npm** y se te hará muy familiar los comandos a la hora de instalar paquetes con **pip**.

- Un paso adicional antes de empezar a instalar paquetes es tener actualizado **pip** en nuestro entorno virtual, que si recuerdas ya se encuentra activado en la consola de VS Code, ejecuta el siguiente comando:

pip install --upgrade pip

- Ahora procederemos a instalar Django en el entorno virtual:

pip install django

Esta herramienta permite crear una interfaz web, a través de lenguajes de programación como HTML, CSS o JavaScript; los cuales permiten adaptar el contenido de la web a los diferentes tipos de pantallas (diseño responsive) y mejorando, por tanto, la experiencia de usuario.

Crear un proyecto de Django.

Ahora que ya tenemos instalado Django tenemos disponible el comando **django-admin** que nos permitirá crear nuestro primer proyecto.

Vamos a la consola de VS Code con tu entorno virtual activado y ejecuta el siguiente comando:

django-admin startproject proyecto_django

Creando archivos de configuración para cada entorno

En un entorno profesional de trabajo es ideal manejar un servidor diferente por cada etapa del proyecto, es decir, cuando realizamos cambios al código del proyecto, usamos un entorno de desarrollo, cuando el cliente prueba funcionalidades nuevas para aprobarlas, se usa un servidor de staging, y finalmente cuando la aplicación es accesible para todos, se usa un servidor de producción.

Es por ello importante manejar archivos de configuración diferentes por cada entorno, ya que esas configuraciones serán diferentes para cada servidor, por lo que sería tedioso manejar un solo archivo y estar cambiando a cada rato cuando nos cambiamos de entorno.



Django por defecto maneja toda la configuración del proyecto en un solo archivo (proyecto_django/settings.py), pero ahora queremos manejar múltiples archivos de configuración por cada entorno manejado.

Para ello vamos a agrupar todos esos archivos de configuración en una sola carpeta llamada settings, creamos la carpeta dentro de la subcarpeta proyecto_django y creamos la siguiente estructura de archivos:

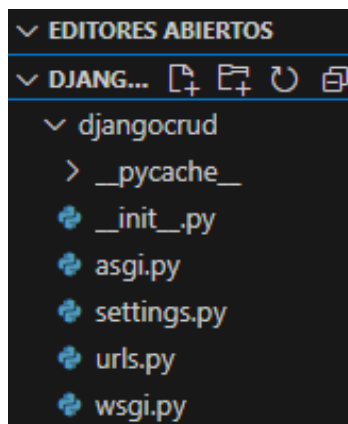


Figura 1. Carpetas del Proyecto en Django.

Vamos a describir cada archivo de la carpeta settings:

- `__init__.py`: Un archivo vacío que le dice a Python que esta carpeta se trata de un paquete.
- `Asgi.py`: Configuración ASGI para el proyecto djangocrud.
- `Settings.py`: Configuraciones de Django para el proyecto Django Crud (Middleware, DataBase, Security, Aplicaciones definidas, etc).
- `Urls.py`: Configuración de URL sin procesar de Django.
- `Wsgi.py`: Configuración de WSGI para el proyecto djangocrud.

Creando nuestra primera app.

Si ejecutamos el siguiente comando, Django va a crear una app en la raíz del proyecto:

`python manage.py startapp task`

Aunque esto es correcto, si el proyecto va creciendo y se van creando más apps, se tendrá en la raíz del proyecto una lista interminable de carpetas.

Entonces vamos a aplicar desde el inicio del proyecto buenas prácticas. Crea en la raíz del proyecto una carpeta llamada apps, aquí englobaremos todas las apps creadas.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

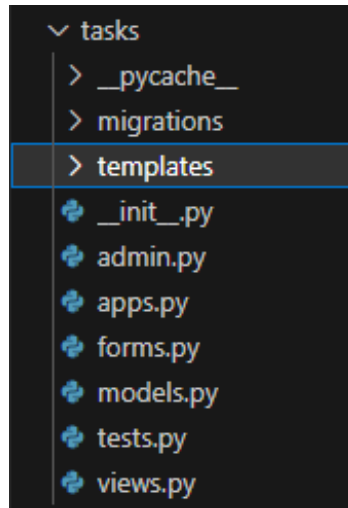


Figura 2. App Task

Vamos a explicar rápidamente cada archivo:

- views.py: Contiene la lógica o funcionalidad de la aplicación.
- models.py: Contiene un conjunto de clases que definen la estructura de datos a utilizar por la aplicación.
- migrations: Una carpeta que es usada por Django para manejar las versiones de la base de datos.
- apps.py: Configuraciones propias de la aplicación
- admin.py: Creación de la interfaz administrativa para el manejo de modelos.
- tests.py: Creación de pruebas.



Creando nuestra primera vista

Modifica el archivo task/views.py para crear una vista que nos retorne un saludo como respuesta:

```
1 from django.shortcuts import render, redirect, get_object_or_404
2 from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
3 from django.contrib.auth import login, logout, authenticate
4 from django.contrib.auth.models import User
5 from django.db import IntegrityError
6 from django.utils import timezone
7 from .models import Task
8
9 from .forms import TaskForm
10
11 # Create your views here.
12
13
14 def signup(request):
15     if request.method == 'GET':
16         return render(request, 'signup.html', {"form": UserCreationForm})
17     else:
18         if request.POST["password1"] == request.POST["password2"]:
19             try:
20                 user = User.objects.create_user(
21                     request.POST["username"], password=request.POST["password1"])
22                 user.save()
23                 login(request, user)
24                 return redirect('home')
```

Figura 3. Importaciones de django.



```
def signup(request):
    if request.method == 'GET':
        return render(request, 'signup.html', {"form": UserCreationForm})
    else:
        if request.POST["password1"] == request.POST["password2"]:
            try:
                user = User.objects.create_user(
                    request.POST["username"], password=request.POST["password1"])
                user.save()
                login(request, user)
                return redirect('tasks')
            except IntegrityError:
                return render(request, 'signup.html', {"form": UserCreationForm, "error": "El usuario ya existe."})
        return render(request, 'signup.html', {"form": UserCreationForm, "error": "Las contraseñas no coinciden."})

def tasks(request):
    tasks = Task.objects.filter(user=request.user, datecompleted__isnull=True)
    return render(request, 'tasks.html', {"tasks": tasks})

def tasks_completed(request):
    # ...
```

Figura 4. Creación de las funciones de inicio de sesión y de las tareas.

```
def tasks_completed(request):
    tasks = Task.objects.filter(user=request.user, datecompleted__isnull=False).order_by('datecompleted')
    return render(request, 'tasks.html', {"tasks": tasks})

def create_task(request):
    if request.method == 'GET':
        return render(request, 'create_task.html', {"form": TaskForm})
    else:
        try:
            form = TaskForm(request.POST)
            new_task = form.save(commit=False)
            new_task.user = request.user
            new_task.save()
            return redirect('tasks')
        except ValueError:
            return render(request, 'create_task.html', {"form": TaskForm, "error": "El formato de la fecha no es correcto."})

def home(request):
    return render(request, 'home.html')
```

Figura 5. Creación de las funciones Tareas completas, Tareas creadas y el Inicio.



```
def signout(request):  
    logout(request)  
    return redirect('home')  
  
def signin(request):  
    if request.method == 'GET':  
        return render(request, 'signin.html', {"form": AuthenticationForm})  
    else:  
        user = authenticate(  
            request, username=request.POST['username'], password=request.POST['password']  
        )  
        if user is None:  
            return render(request, 'signin.html', {"form": AuthenticationForm, "error": "Invalid credentials"})  
        login(request, user)  
        return redirect('tasks')  
  
def task_detail(request, task_id):  
    if request.method == 'GET':  
        task = get_object_or_404(Task, pk=task_id, user=request.user)  
        form = TaskForm(instance=task)  
        return render(request, 'task_detail.html', {'task': task, 'form': form})  
    else:  
        try:  
            task = get_object_or_404(Task, pk=task_id, user=request.user)
```

Figura 6. Creación de los detalles de la tarea, inicio de sesión y cierre de sesión.

```
def complete_task(request, task_id):  
    task = get_object_or_404(Task, pk=task_id, user=request.user)  
    if request.method == 'POST':  
        task.datecompleted = timezone.now()  
        task.save()  
        return redirect('tasks')  
  
def delete_task(request, task_id):  
    task = get_object_or_404(Task, pk=task_id, user=request.user)  
    if request.method == 'POST':  
        task.delete()  
        return redirect('tasks')
```

Figura 7. Creación de las funciones eliminar tareas y completar tarea.



Base de datos

Para el uso de las bases de datos tenemos la herramienta ORM de Django que es una implementación del concepto de mapeo de objeto relacional (ORM). Una de las características más poderosas de Django es su Mapeador Relacional de Objetos (ORM), que le permite interactuar con su base de datos, como lo haría con instrucciones SQL (Structured Query Language).

Esto nos permite manejar los datos de manera sencilla y rápida al estar integrados en nuestro proyecto, esto se realiza mediante las migraciones de las clases que implementamos como tablas con atributos, para ello debemos ir dentro de task en el archivo Models.py

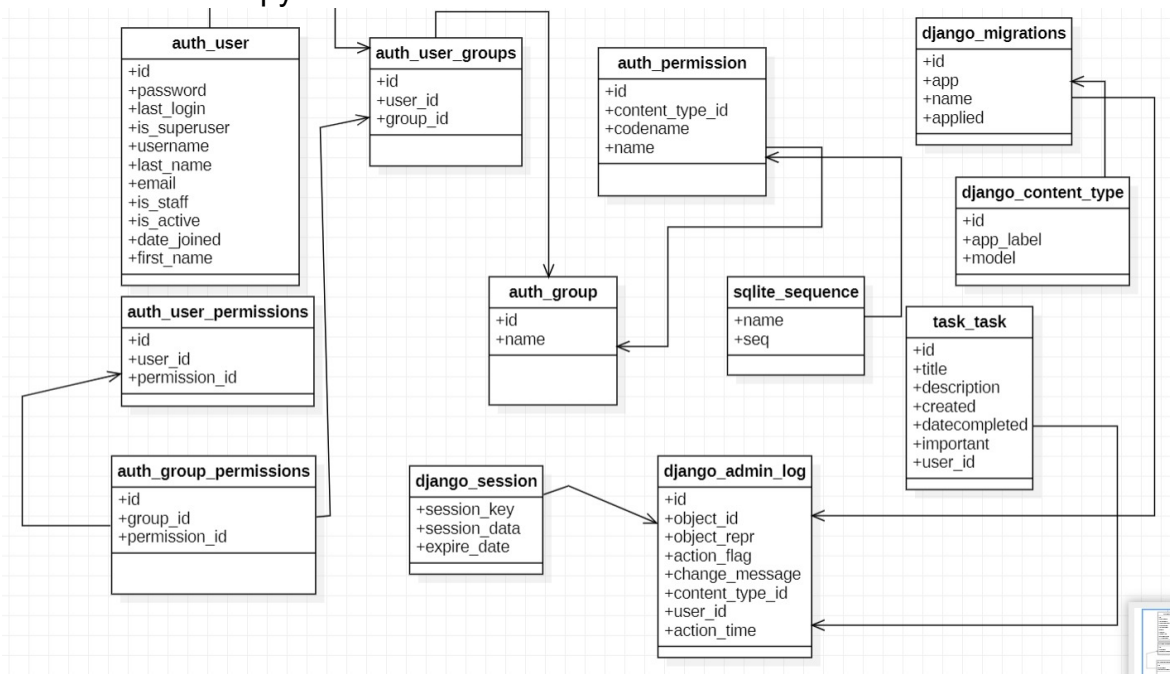


Figura 8. Diagrama de clases.

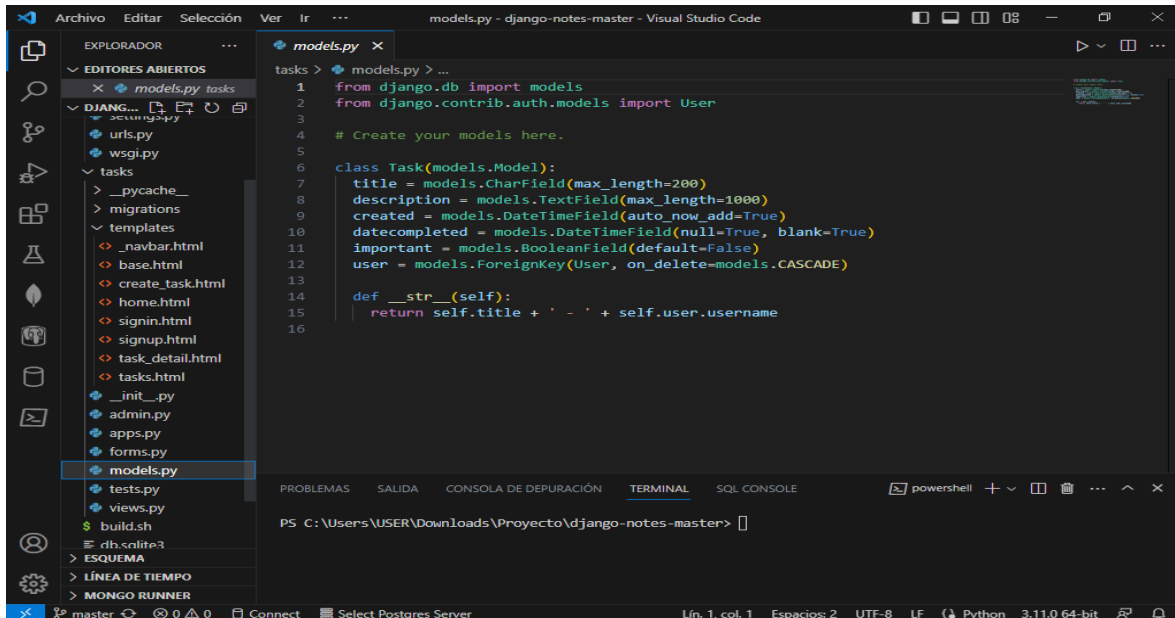


Figura 9. Models.py

Las migraciones son cómo almacena Django cambios a sus modelos (y por tanto el esquema de base de datos) - son simplemente los archivos en el disco. Puedes leer la migración para su nuevo modelo si se quiere; que es el archivo migrations/0001_initial.py

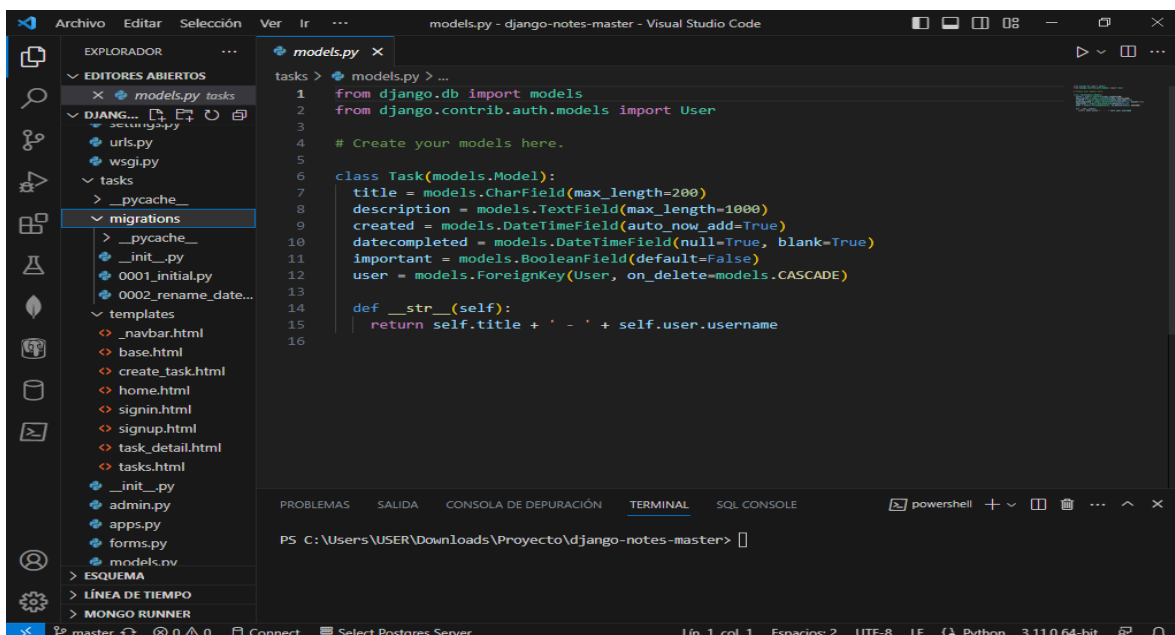


Figura 10. Migraciones



Esto mediante el comando:

python manage.py makemigrations

Este comando nos permite realizar las migraciones ya mencionadas y mencionar los cambios realizados a la base de datos y con esto podemos realizar un mejor modelado de la base de datos.

auth_group			
id	name		

auth_group_permissions		
id	group_id	permission_id

auth_permission			
id	content_type_id	codename	name

auth_user										
id	password	last_login	is_superuser	username	last_name	email	is_staff	is_active	date_joined	first_name

auth_user_groups		
id	user_id	group_id

Figura 11. Tabla de la base de datos.

auth_user_permissions		
id	user_id	permission_id

django_admin_log						
id	object id	object repr	action flag	change message	content_type_id	user_id

django_content_type		
id	app_label	model

django_migrations			
id	app	name	applied

Figura 12. Tablas de la base de datos



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

django_session						
session_key	session_data				expire_date	

sqlite_sequence	
name	seq

task_task						
id	title	description	created	datecompleted	important	user_id

Figura 13. Tablas de la base de datos.

```
67     },
68 },
69 ]
70
71 WSGI_APPLICATION = 'djangocrud.wsgi.application'
72
73 # Database
74 # https://docs.djangoproject.com/en/4.1/ref/settings/#databases
75
76 DATABASES = {
77     'default': {
78         'ENGINE': 'django.db.backends.sqlite3',
79         'NAME': BASE_DIR / 'db.sqlite3',
80     }
81 }
82
83 # Password validation
84 # https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators
85
86 AUTH_PASSWORD_VALIDATORS = [
87     {
88         'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityVa
89
90
```

Figura 14. Setting Base de datos



Como podemos observar en el archivo de settings.py de nuestro proyecto ya tenemos implementada la conexión, esta conexión puede ser modificada para que se utilice el manejador de base de datos de nuestra conveniencia dependiendo de nuestras necesidades ya sean relaciones o no relacionales.

En este caso nosotros cambiamos esta base de datos por una de postgresql la cual nos permitirá subirla a un sistema de hosting en la nube.

```
settings.py
67
68
69
70
71 WSGI_APPLICATION = 'djangocrud.wsgi.application'
72
73
74 # Database
75 # https://docs.djangoproject.com/en/4.1/ref/settings/#databases
76
77 DATABASES = {
78     'default': {
79         'ENGINE': 'django.db.backends.sqlite3',
80         'NAME': BASE_DIR / 'db.sqlite3',
81     }
82 }
83
84 # Password validation
85 # https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators
86
87 AUTH_PASSWORD_VALIDATORS = [
88     {
89         'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityVa...
```

Figura 15. Modificación de la base de datos en el archivo settings.py

Y en este caso usaremos esta clase creada de Task la cual nos ayudara a crear un formulario.

```
forms.py
1 from django.forms import ModelForm
2 from .models import Task
3
4 class TaskForm(ModelForm):
5     class Meta:
6         model = Task
7         fields = ['title', 'description', 'important']
```

Figura 16. Forms.py



URL del proyecto.

Estas funciones deben estar declaradas dentro del archivo urls.py dentro de nuestro proyecto, esto debido a que en caso contrario nos lanzara el error de que no existe la ruta.

```
14 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15
16 from django.contrib import admin
17 from django.urls import path
18 from tasks import views
19
20 urlpatterns = [
21     path('', views.home, name='home'),
22     path('admin/', admin.site.urls),
23     path('signup/', views.signup, name='signup'),
24     path('tasks/', views.tasks, name='tasks'),
25     path('tasks_completed/', views.tasks_completed, name='tasks_completed'),
26     path('logout/', views.signout, name='logout'),
27     path('signin/', views.signin, name='signin'),
28     path('create_task/', views.create_task, name='create_task'),
29     path('tasks/<int:task_id>', views.task_detail, name='task_detail'),
30     path('tasks/<int:task_id>/complete', views.complete_task, name='complete_task'),
31     path('tasks/<int:task_id>/delete', views.delete_task, name='delete_task'),
32 ]
33
```

Figura 17. Adición de las rutas con la importación de las views de task.



Templates.

Teniendo las rutas es necesario crear los archivos html y para ello en la carpeta de nuestra app task, crearemos la carpeta donde guardaremos nuestras plantillas.

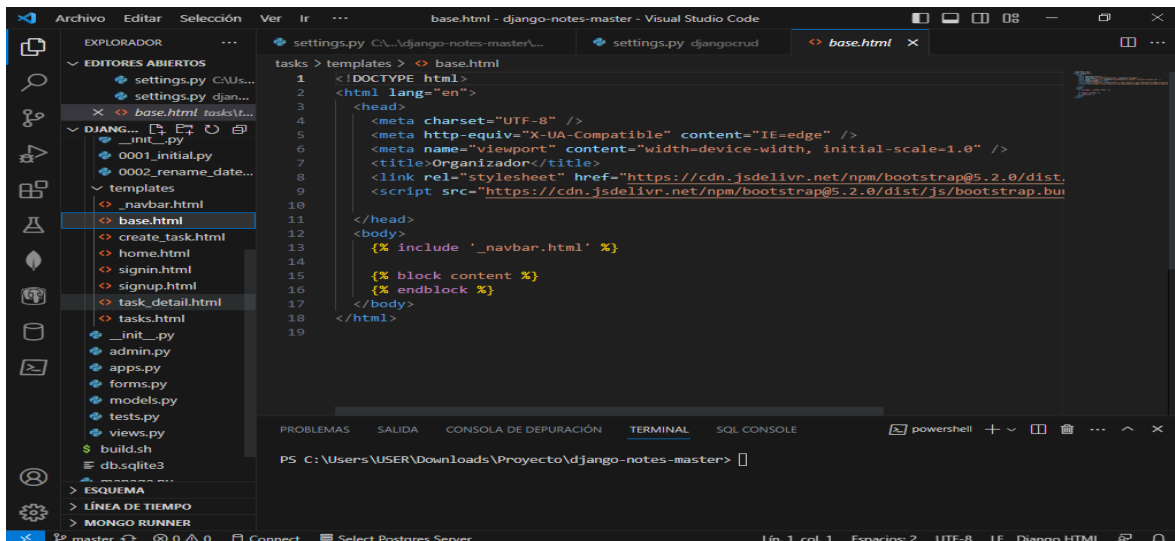


Figura 18. Carpeta templates.

Crearemos la base que no observa en la figura 18 esto con el fin de optimizar el código al solo usar una base de página que cambia el contenido mediante la necesidad de nuestro proyecto, mostrando formulario, listas o texto sin hacer una redundancia en el código.

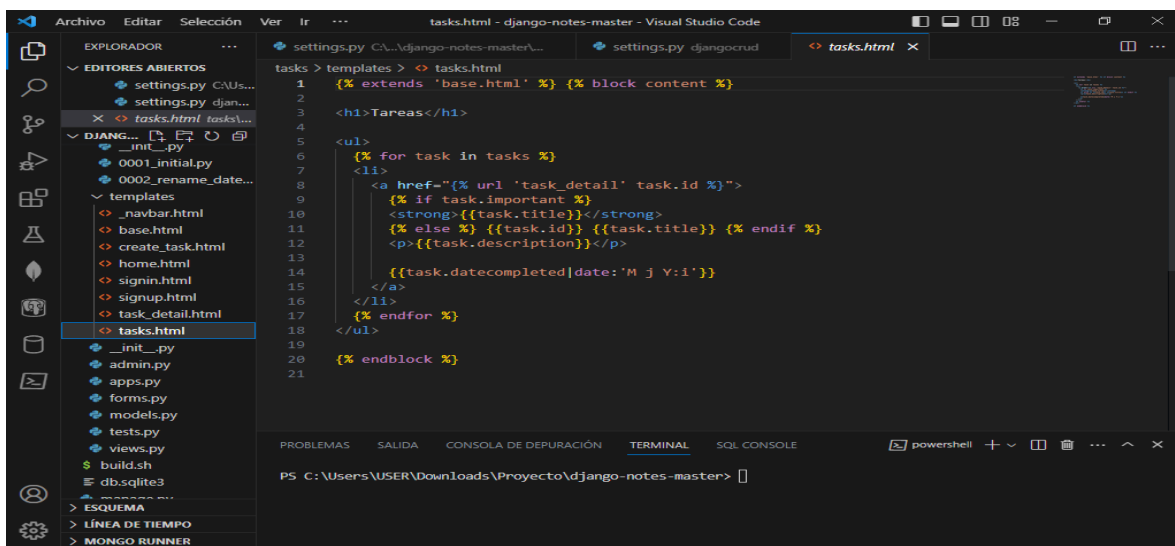


Figura 19. Task.html



Como podemos observar el html task tiene una extensión del html base el cual como se comentó, solo cambia el contenido de la página dejando una base establecida.

```
1 <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
2   <div class="container">
3     <a class="navbar-brand" href="/">Organizacion</a>
4     <button
5       class="navbar-toggler"
6       type="button"
7       data-bs-toggle="collapse"
8       data-bs-target="#navbarNav"
9       aria-controls="navbarNav"
10      aria-expanded="false"
11      aria-label="Toggle navigation"
12    >
13      <span class="navbar-toggler-icon"></span>
14    </button>
15    <div class="collapse navbar-collapse" id="navbarNav">
16      <ul class="navbar-nav ms-auto">
17        <li class="nav-item">
18          <a href="/" class="nav-link">Inicio</a>
19        </li>
20        {% if user.is_authenticated %}
21        <li class="nav-item">
22          <a href="{% url 'tasks_completed' %}" class="nav-link">
23            Tareas Completadas</a>
24        </li>
25      </ul>
26    </div>
27  </div>
28</nav>
```

Figura 20. Barra de navegación.

```
1 {% extends "base.html" %} {% block content %}
2
3 <div class="container">
4   <h1>Inicio</h1>
5 </div>
6
7 {% endblock %}
8
```

Figura 21. Home.html



Usando la base de datos, nosotros creamos un formulario el cual es declarado en el html como en el ejemplo de task_details,

```
tasks > templates > task_detail.html
9  <form method="POST">
10  { csrf_token %}
11  {{ form.as_p }}
12
13  <button>
14  Actualizar Tarea
15  </button>
16 </form>
17
18 <form action="{% url 'complete_task' task.id %}" method="POST">
19 { csrf_token %}
20 <button>
21 Completar
22 </button>
23 </form>
24
25 <form action="{% url 'delete_task' task.id %}" method="POST">
26 { csrf_token %}
27 <button>
28 Borrar Tarea
29 </button>
30 </form>
31
32 {% endblock %}
```

Figura 22. Task_details html (Formulario).

```
tasks > templates > create_task.html
1  {% extends 'base.html' %}
2
3  {% block content %}
4  <h1>Crear Tarea</h1>
5
6  {{ error }}
7
8  <form action="/create_task/" method="POST">
9  { csrf_token %}
10  {{ form.as_p }}
11
12  <button>Guardar</button>
13  </form>
14  {% endblock %}
```

Figura 23. Create Task.



Figura 24. Singin.

Figura 23. Singup.



Teniendo todos los templates, se utiliza el framework Bootstrap 5, una librería que nos dará estilo, de esta manera tendremos un esquema de la aplicación similar a como se vera la aplicación de una manera mas interactiva al nosotros tener un diseño el cual seguimos.

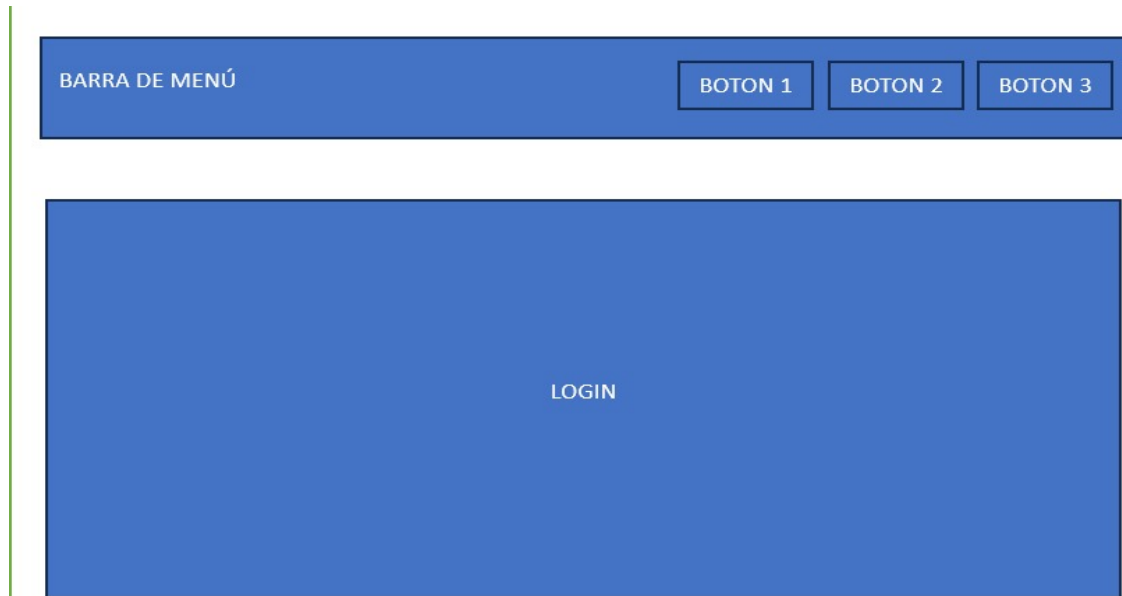


Figura 24. Maquetado login.

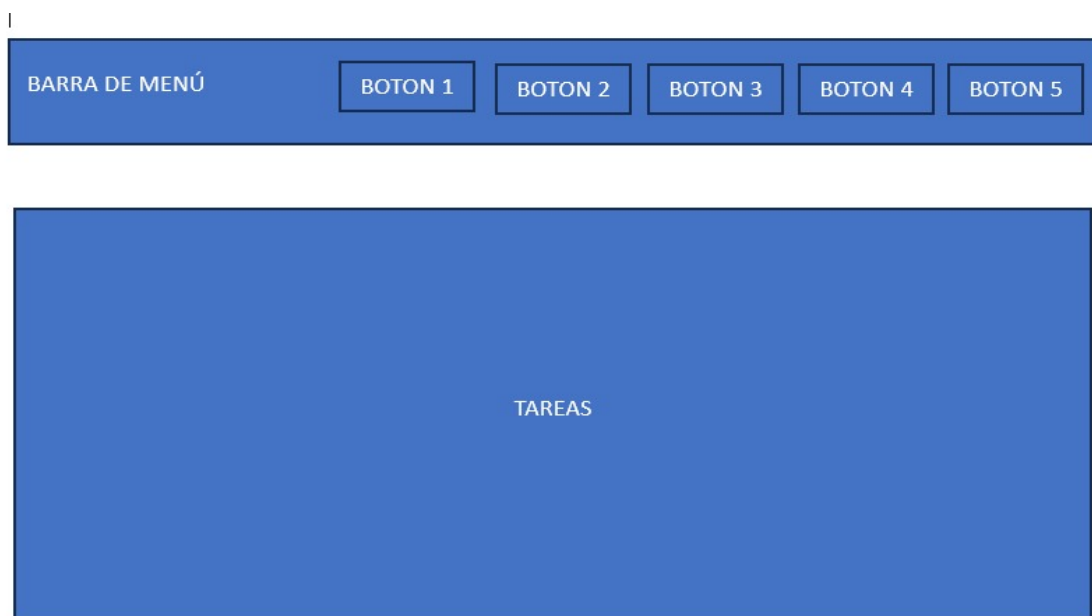


Figura 25. Maquetado tareas.



Figura 25. Maquetada lista de tareas.

Servidor Local.

Para realizar las pruebas y ver si se obtuvo el resultado esperado, en la consola powershell que nos ofrece visual studio code, dentro de la carpeta del proyecto, utilizamos el siguiente comando:

Python manage.py runserver

Compilara el proyecto y nos arrojará un link del localhost que nos permite observar que el puerto usado sería el puerto 8000 con la ip 127.0.0.1.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  SQL CONSOLE  python + v [icon] ... ^ x

PS C:\Users\USER\Downloads\Nueva carpeta\django-notes-master> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 17, 2023 - 19:40:28
Django version 4.1.3, using settings 'djangocrud.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

Lector de pantalla optimizado [icon] [icon]
```

Figura 26. Terminal y runserver.



**UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN**



**Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática**

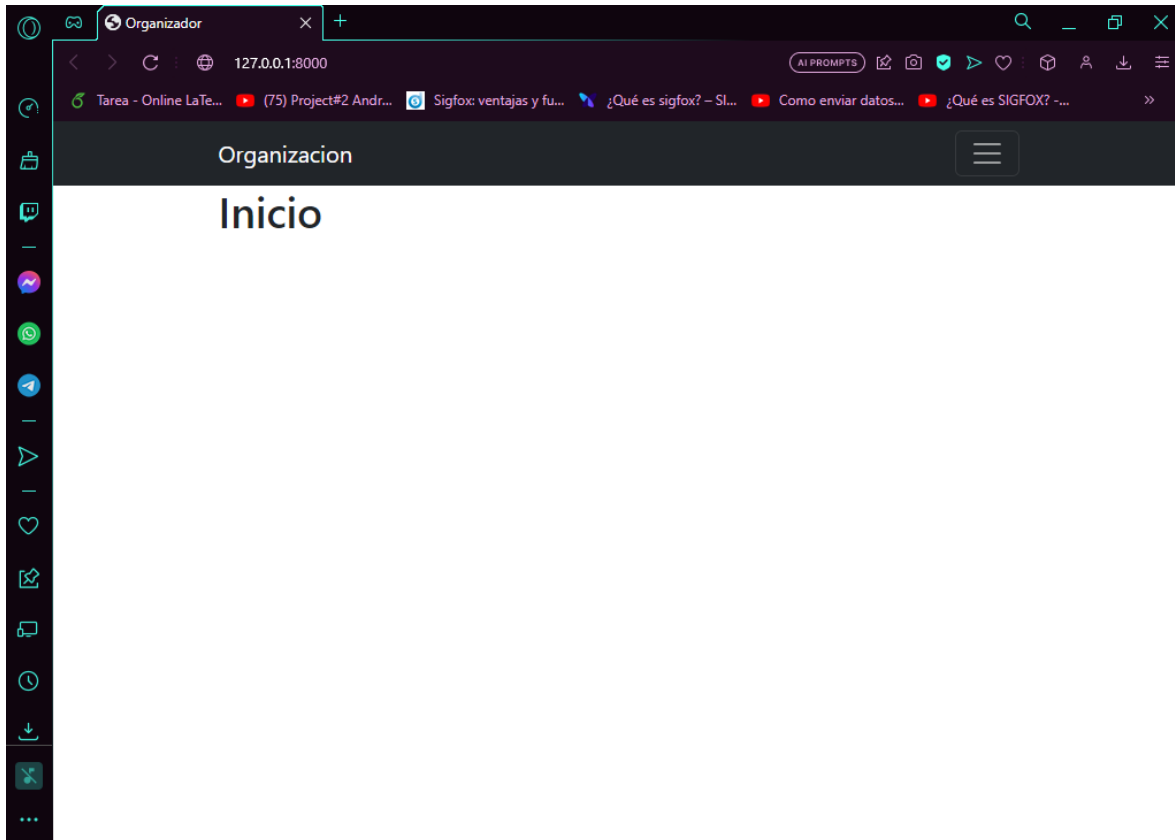


Figura 27. Inicio.

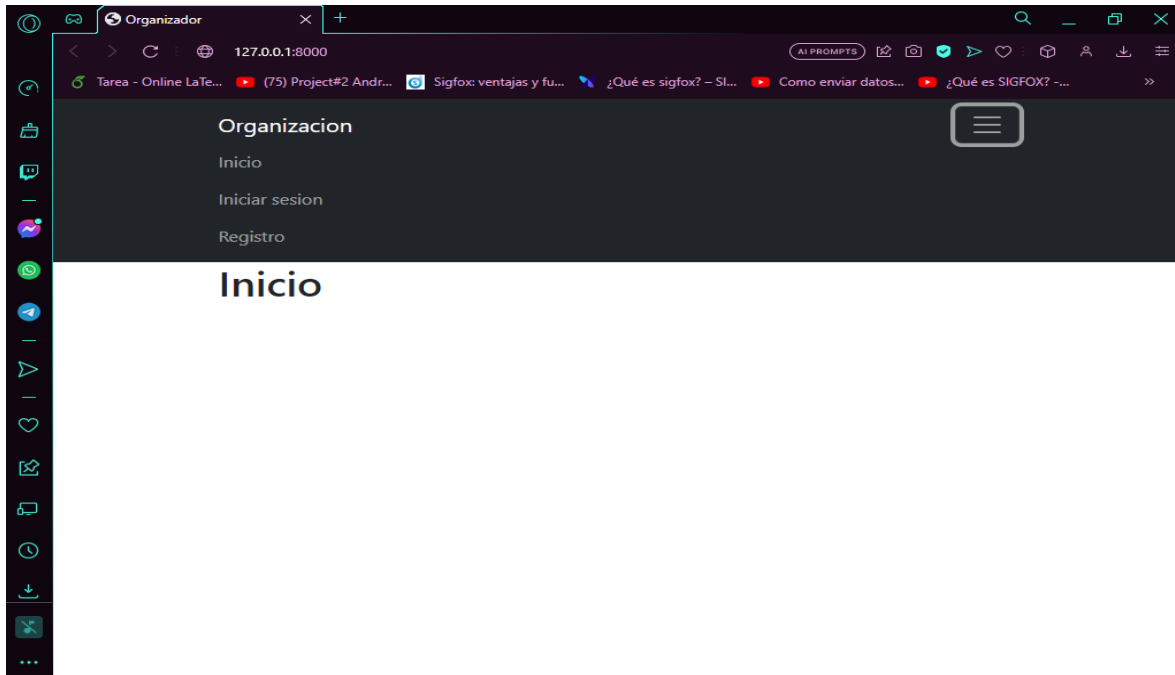


Figura 28. Barra de navegación.

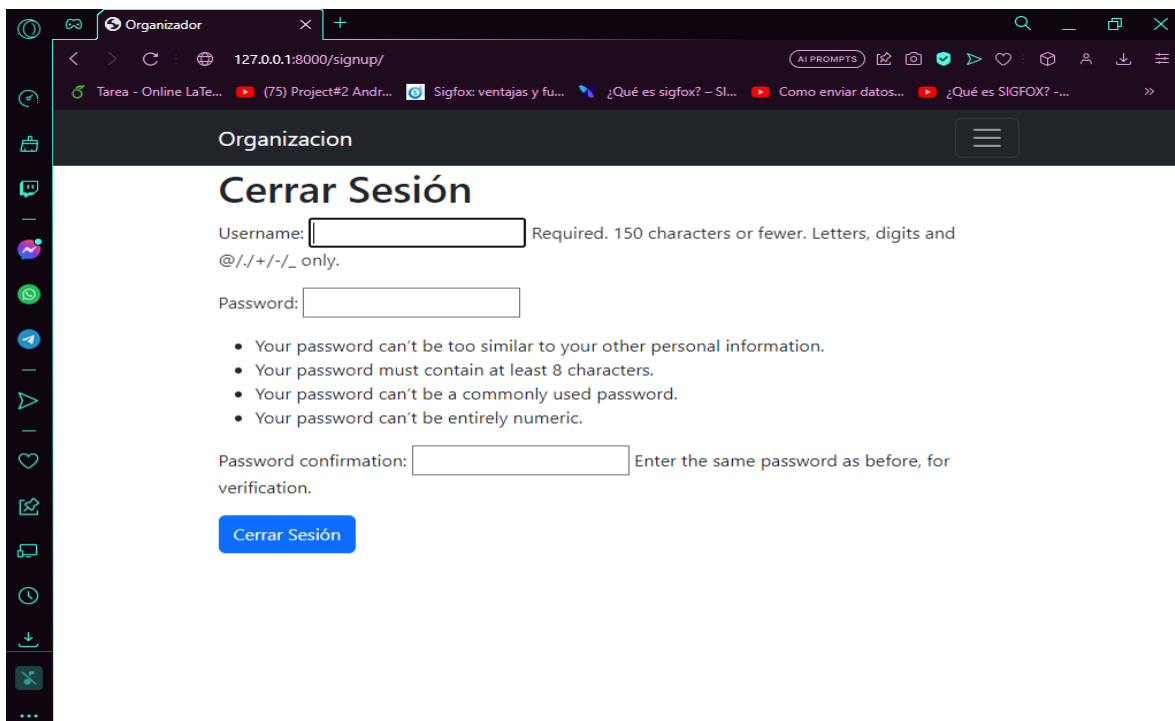


Figura 29. Registro.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

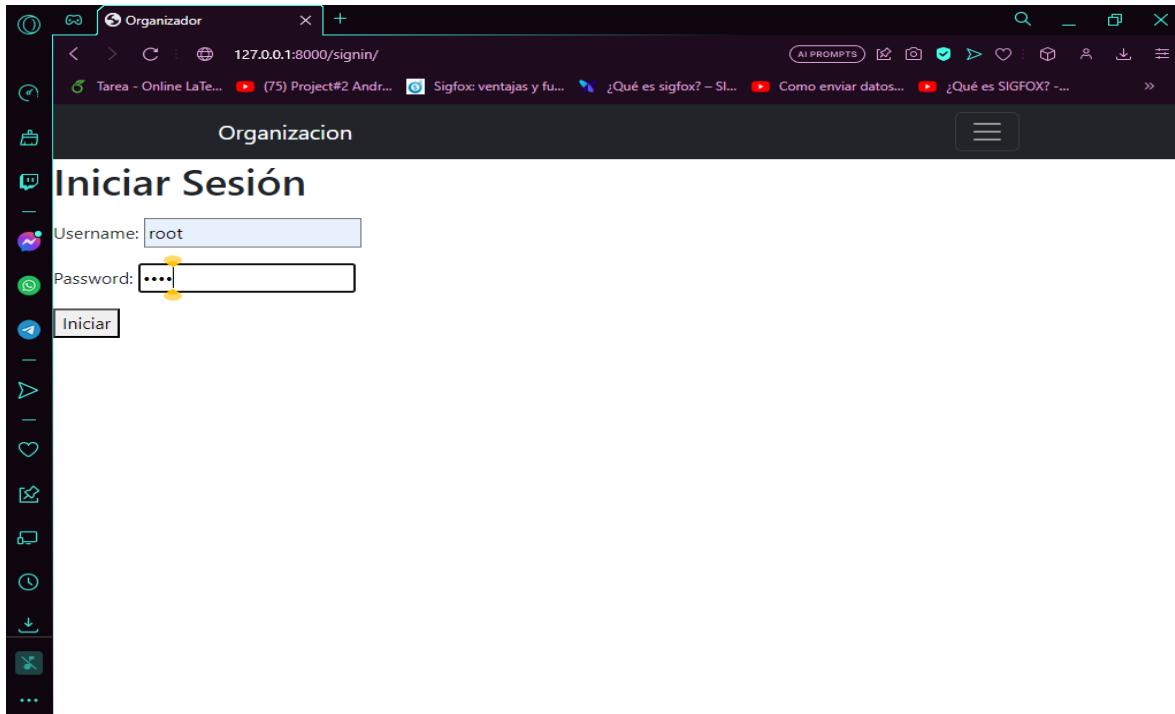


Figura 30. Inicio de sesión.

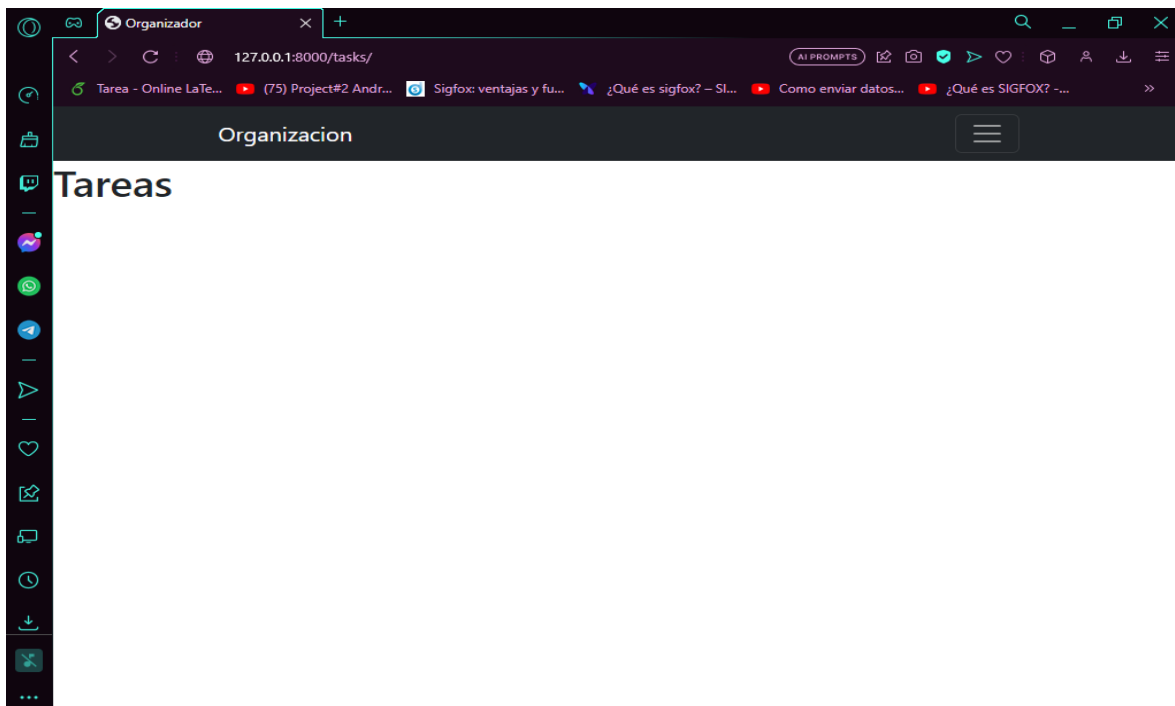


Figure 31. Tareas.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

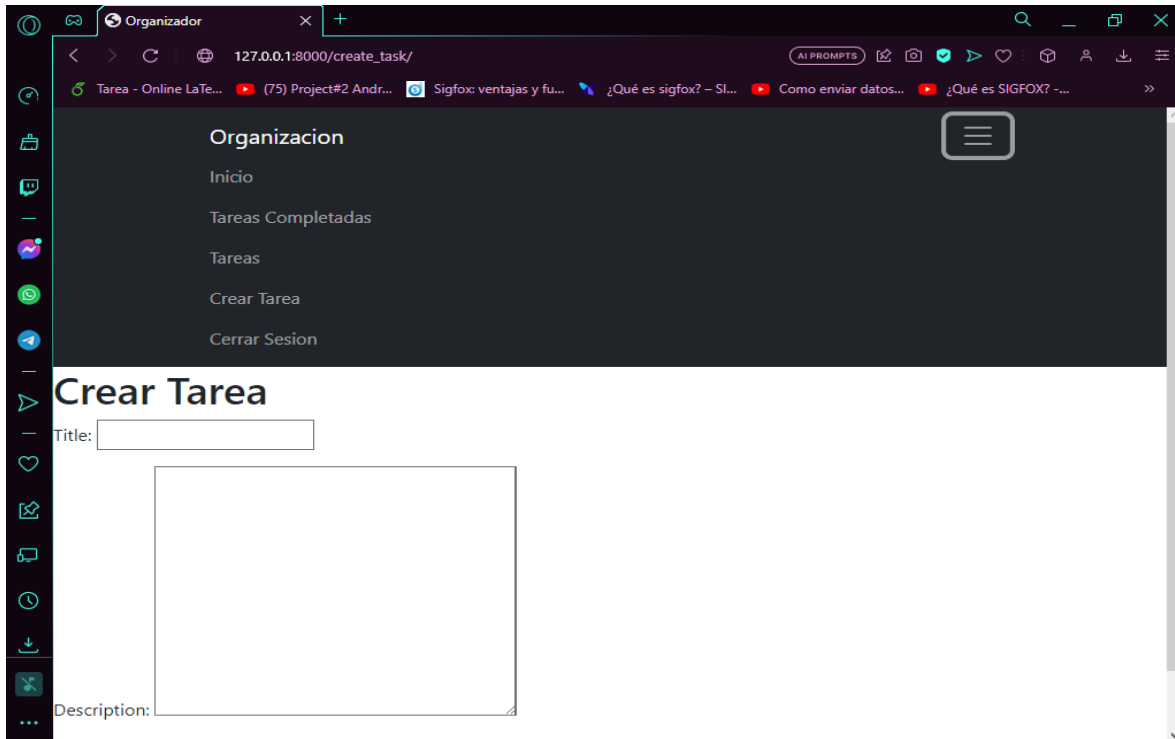


Figura 32. Barra de navegación.

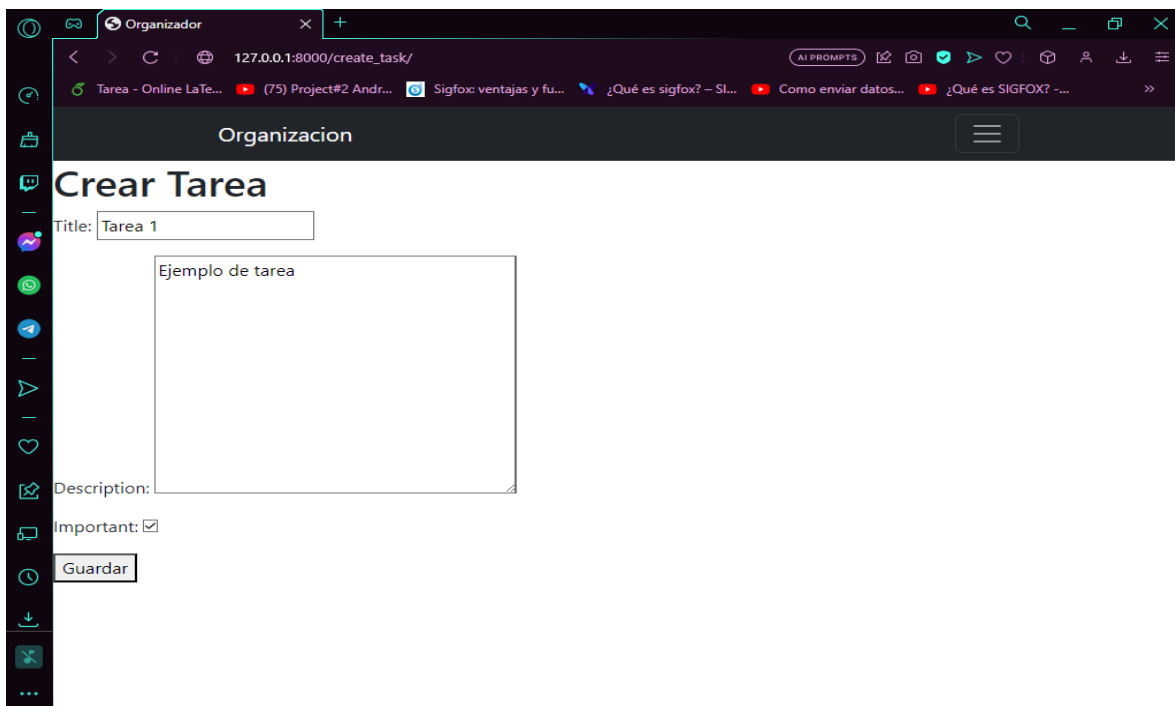


Figura 33. Crear Tarea.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

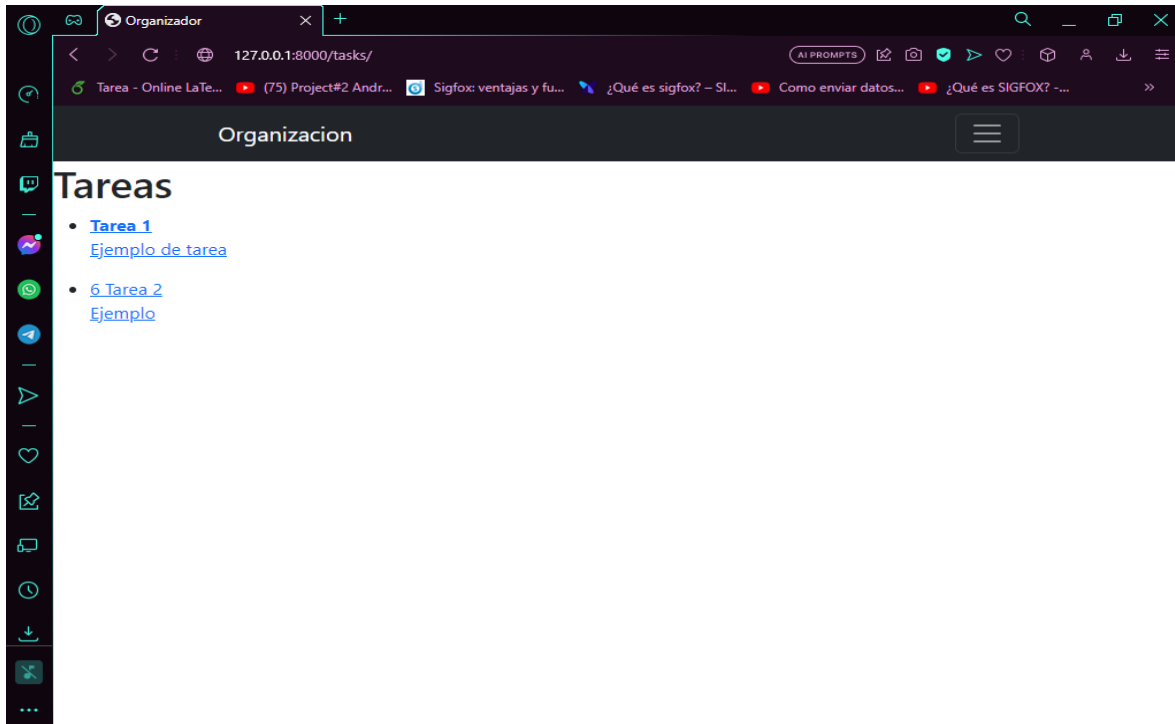


Figura 34. Lista de Tareas.

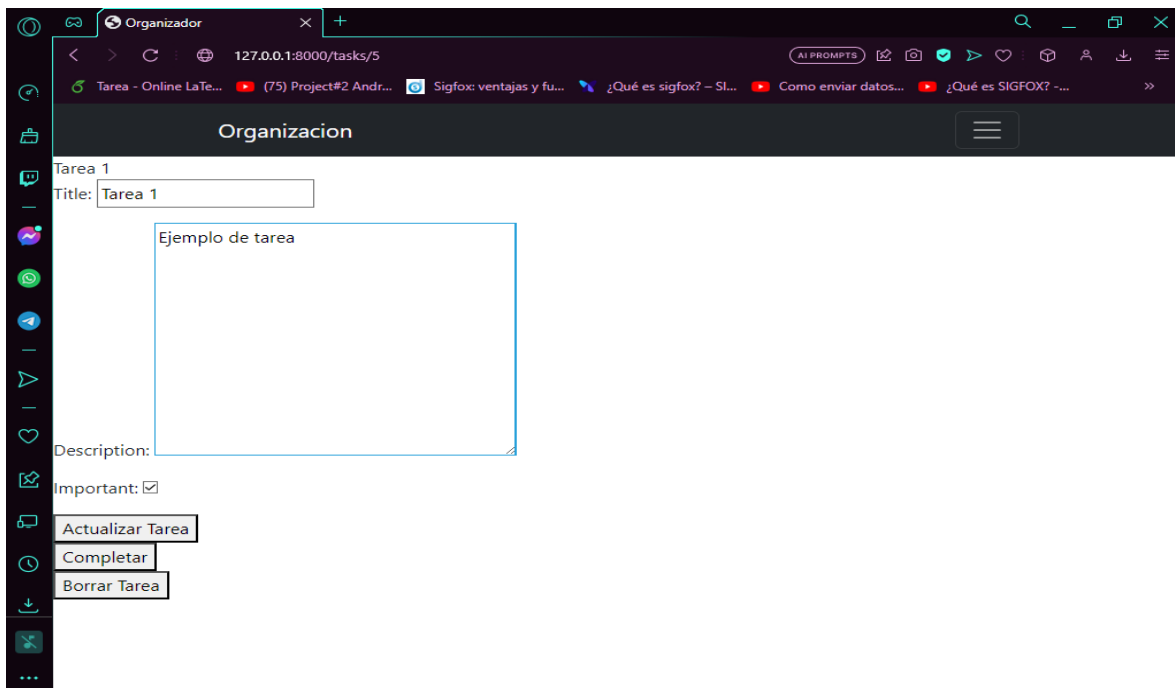


Figura 35. Implementación de CRUD.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

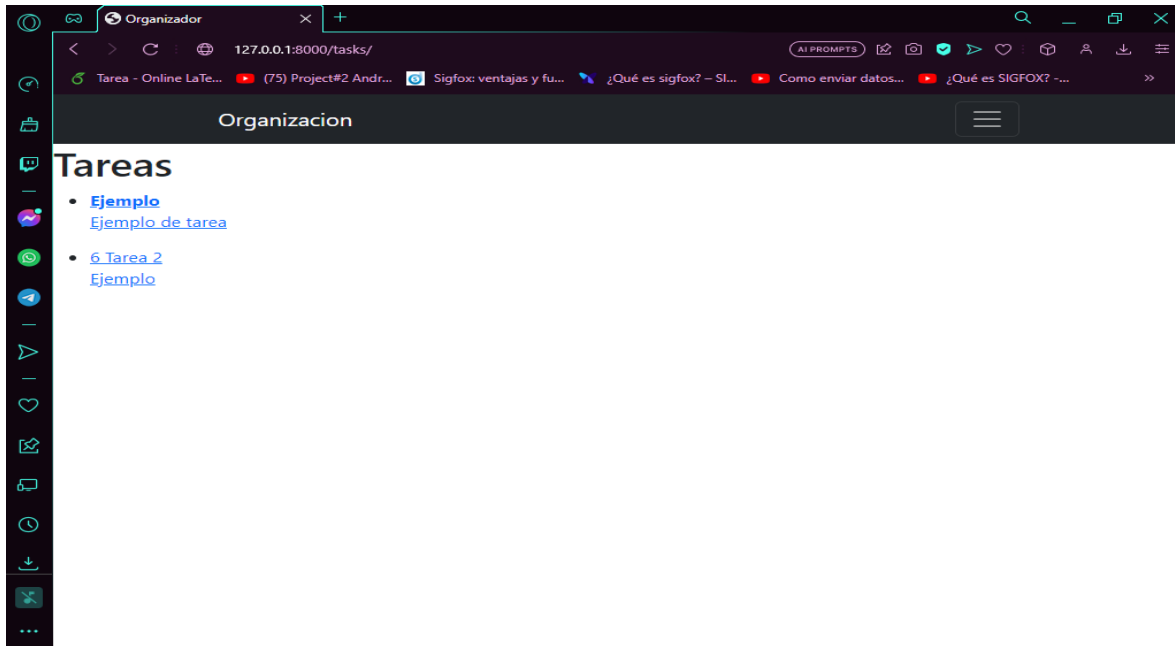


Figura 36. modificación de tarea.

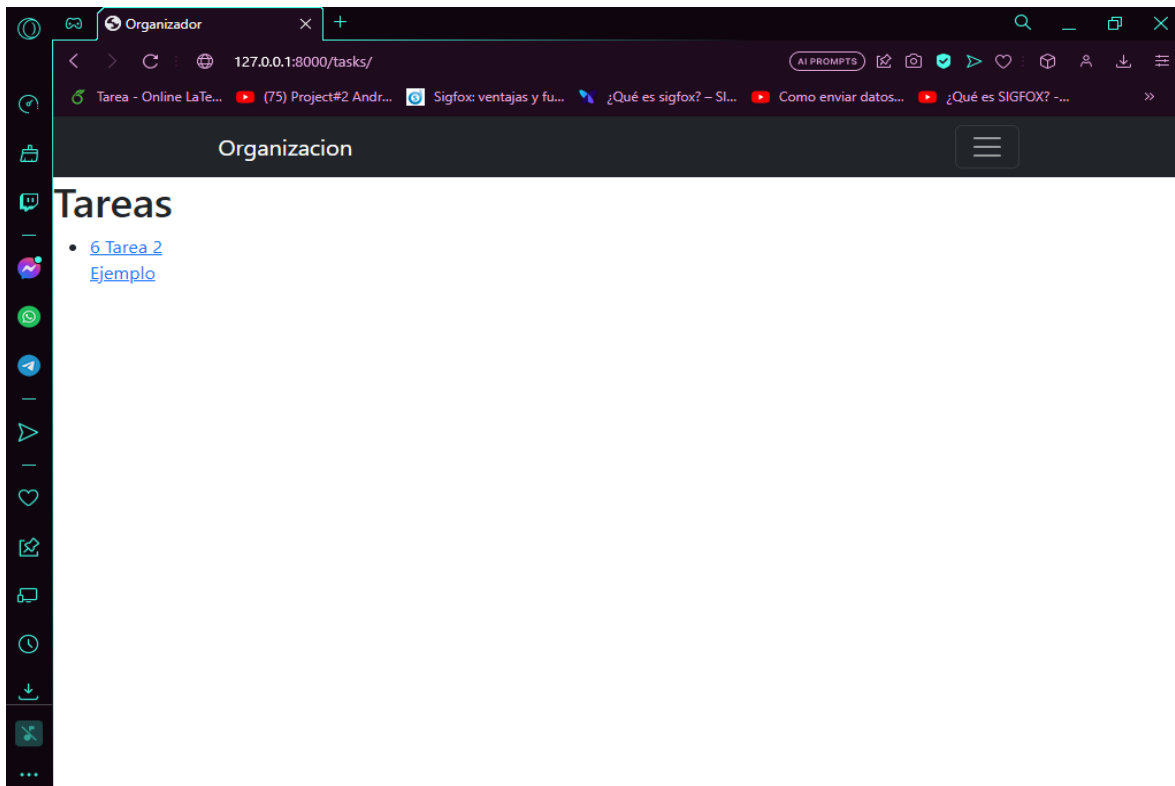


Figura 37. eliminación de Tarea.

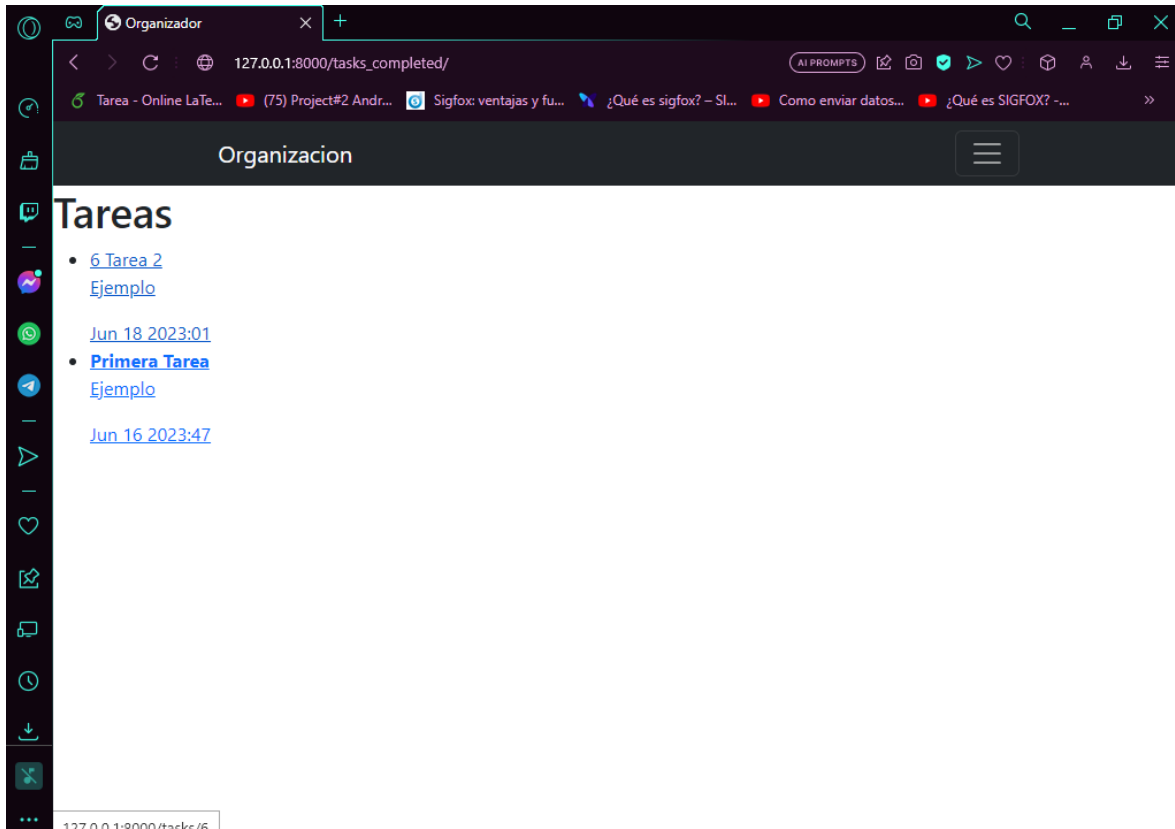


Figure 38. Tareas completadas.

Servidor externo.

El renderizado en la nube es un tipo de proceso de edición que utiliza servidores remotos para renderizar los gráficos. La ventaja del renderizado en la nube es que libera los recursos de tu ordenador local y puede proporcionar una experiencia de renderizado más rápida.

Antes de implementar cualquier aplicación seria en un entorno de producción, debe asegurarse de que esté correctamente protegida y configurada. La documentación de Django proporciona una lista de verificación de implementación útil, que seguiremos en este paso.

Ingeniería web
1er Departamental

Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

Abra `django-notes-master/settings.py` y busque la declaración de la configuración `SECRET_KEY`. No queremos almacenar secretos de producción en el código fuente, por lo que los buscaremos en una variable de entorno que crearemos más adelante:

```
import os
import dj_database_url

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get('SECRET_KEY', default='your secret key')
```

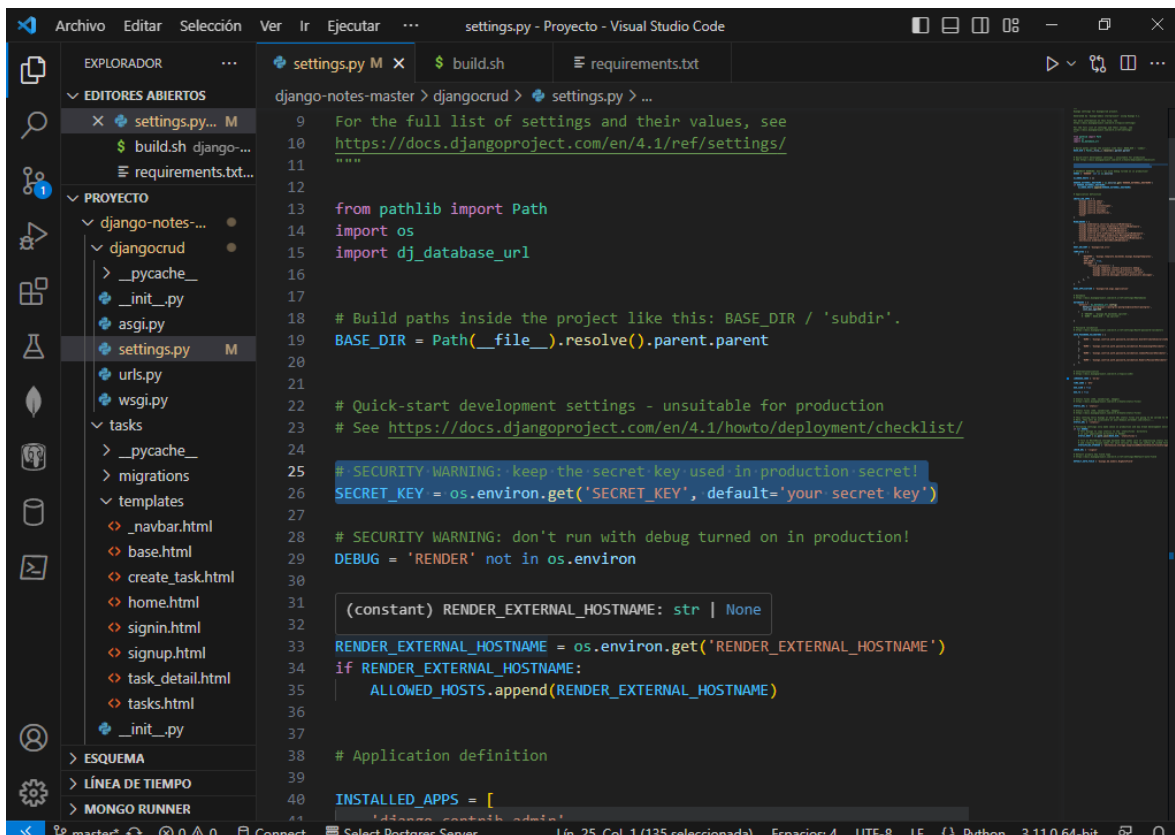
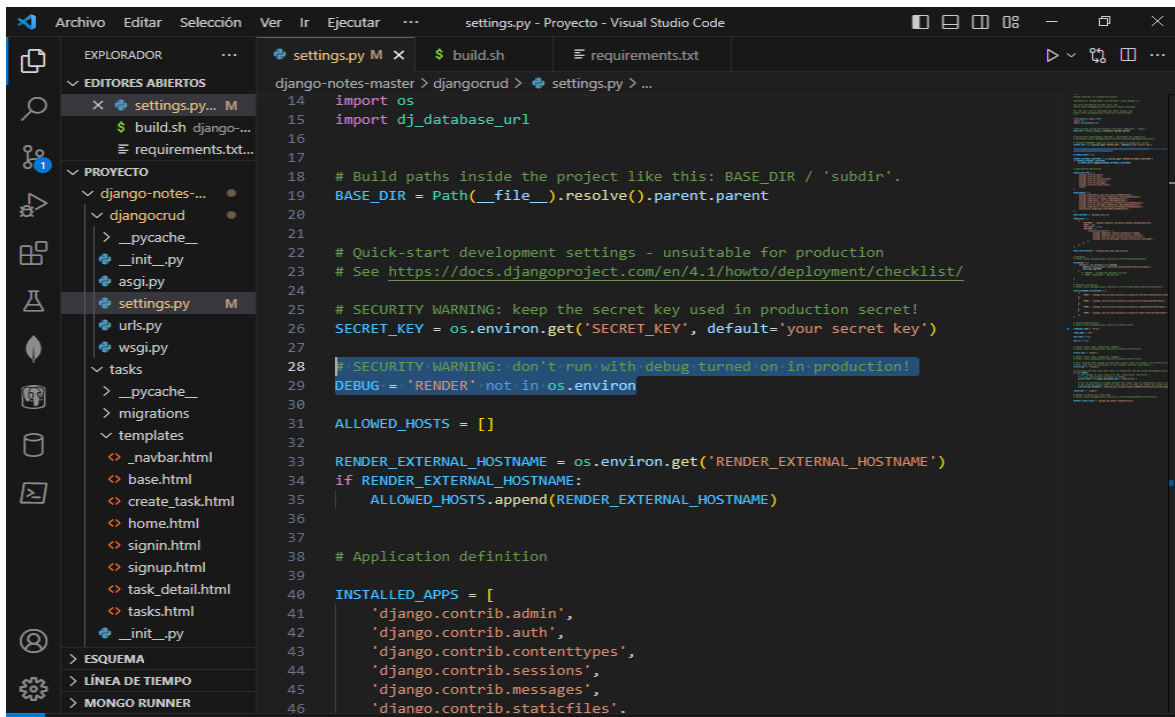


Figura 39. Modificación en setting.



Busque la declaración de la configuración DEBUG. Esta configuración nunca debe establecerse en True en un entorno de producción. Puede detectar si está ejecutando Render comprobando si la variable de entorno RENDER está presente en el entorno de la aplicación.

DEBUG = 'RENDER' not in os.environ



```
14 import os
15 import dj_database_url
16
17
18 # Build paths inside the project like this: BASE_DIR / 'subdir'.
19 BASE_DIR = Path(__file__).resolve().parent.parent
20
21
22 # Quick-start development settings - unsuitable for production
23 # See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/
24
25 # SECURITY WARNING: keep the secret key used in production secret!
26 SECRET_KEY = os.environ.get('SECRET_KEY', default='your secret key')
27
28 # SECURITY WARNING: don't run with debug turned on in production!
29 DEBUG = 'RENDER' not in os.environ
30
31 ALLOWED_HOSTS = []
32
33 RENDER_EXTERNAL_HOSTNAME = os.environ.get('RENDER_EXTERNAL_HOSTNAME')
34 if RENDER_EXTERNAL_HOSTNAME:
35     ALLOWED_HOSTS.append(RENDER_EXTERNAL_HOSTNAME)
36
37
38 # Application definition
39
40 INSTALLED_APPS = [
41     'django.contrib.admin',
42     'django.contrib.auth',
43     'django.contrib.contenttypes',
44     'django.contrib.sessions',
45     'django.contrib.messages',
46     'django.contrib.staticfiles',
```

Figura 40. Render.

Cuando DEBUG = False, Django no funcionará sin un valor adecuado para ALLOWED_HOSTS. Puede obtener el nombre del host de su servicio web de la variable de entorno RENDER_EXTERNAL_HOSTNAME, que Render configura automáticamente.



```
django-notes-master > djangocrud > settings.py > ALLOWED_HOSTS
22 # Quick-start development settings - unsuitable for production
23 # See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/
24
25 # SECURITY WARNING: keep the secret key used in production secret!
26 SECRET_KEY = os.environ.get('SECRET_KEY', default='your secret key')
27
28 # SECURITY WARNING: don't run with debug turned on in production!
29 DEBUG = 'DEBUG' not in os.environ
30
31 ALLOWED_HOSTS = []
32
33 RENDER_EXTERNAL_HOSTNAME = os.environ.get('RENDER_EXTERNAL_HOSTNAME')
34 if RENDER_EXTERNAL_HOSTNAME:
35     ALLOWED_HOSTS.append(RENDER_EXTERNAL_HOSTNAME)
36
37
38 # Application definition
39
40 INSTALLED_APPS = [
41     'django.contrib.admin',
42     'django.contrib.auth',
43     'django.contrib.contenttypes',
44     'django.contrib.sessions',
45     'django.contrib.messages',
46     'django.contrib.staticfiles',
47     'tasks'
48 ]
49
50 MIDDLEWARE = [
51     'django.middleware.security.SecurityMiddleware',
52     'django.contrib.sessions.middleware.SessionMiddleware',
53     'django.middleware.common.CommonMiddleware',
54     'django.middleware.csrf.CsrfViewMiddleware'
```

Figura 41. Nombre del host.

Para mayor comodidad, agregaremos el paquete DJ-Database-URL, que nos permite especificar bases de datos en Django usando cadenas de conexión. Las bases de datos de procesamiento proporcionan automáticamente cadenas de conexión en su panel de control, que luego proporcionaremos a nuestro servicio web a través de la variable de entorno DATABASE_URL. También necesitaremos agregar psycopg2 al proyecto.

Pip install dj-database-url psycopg2-binary

En django-notes-master /settings.py, busque la declaración de la configuración de BASES DE DATOS y modifíquela para que tenga el siguiente aspecto:



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

```
1 """
2 Django settings for djangocrud project.
3
4 Generated by 'django-admin startproject' using Django 4.1.
5
6 For more information on this file, see
7 https://docs.djangoproject.com/en/4.1/topics/settings/
8
9 For the full list of settings and their values, see
10 https://docs.djangoproject.com/en/4.1/ref/settings/
11 """
12
13 from pathlib import Path
14 import os
15 import dj_database_url
16
17 # Build paths inside the project like this: BASE_DIR / 'subdir'.
18 BASE_DIR = Path(__file__).resolve().parent.parent
19
20 # Quick-start development settings - unsuitable for production
21 # See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/
22
23 # SECURITY WARNING: keep the secret key used in production secret!
24 SECRET_KEY = os.environ.get('SECRET_KEY', default='your secret key')
25
26 # SECURITY WARNING: don't run with debug turned on in production!
27 DEBUG = 'RENDER' not in os.environ
28
29 ALLOWED_HOSTS = []
30
31 # RENDER_EXTERNAL_HOSTNAME = os.environ.get('RENDER_EXTERNAL_HOSTNAME')
32
33 # Select Postgres Server
```

Figura 42. Librería dj_database_url

```
72
73
74
75
76
77
78
79
80
81
82
83 # Database
84 # https://docs.djangoproject.com/en/4.1/ref/settings/#databases
85
86 DATABASES = {
87     'default': dj_database_url.config(
88         default='postgresql://postgres:postgres@localhost/postgres',
89         conn_max_age=600
90     ),
91     'ENGINE': 'django.db.backends.sqlite3',
92     'NAME': BASE_DIR / 'db.sqlite3',
93 }
94
95 # Password validation
96 (constant) AUTH_PASSWORD_VALIDATORS = list(dict[str, str]) password_validators
97
98 AUTH_PASSWORD_VALIDATORS = [
99     {
100         'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityVa
101     },
102     {
103         'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator'
104     },
105     {
106         'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator'
107     },
108     {
109         'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator'
110     },
111 ]
112
113 # Select Postgres Server
```

Figura 43. Modificación de la base de datos.



Archivos estáticos

Los sitios web generalmente necesitan servir archivos adicionales como imágenes, JavaScript y CSS. En Django, estos archivos se conocen como archivos estáticos y proporciona un módulo dedicado para recopilarlos en un solo lugar para servir en producción.

El módulo incorporado solo admite mover archivos de un lugar a otro, confiando en servidores web como Apache o NGINX para entregarlos a los usuarios finales. En Render, el servidor web orientado a Internet se proporciona de forma predeterminada y necesitamos una forma de alojar archivos estáticos usándolo. En este paso, configuraremos WhiteNoise, que es una solución muy popular para este problema. Las siguientes instrucciones son una breve descripción general del procedimiento descrito en la documentación de WhiteNoise.

Agregue WhiteNoise como dependencia (agregar compatibilidad con Brotli es opcional, pero se recomienda):

Pip install whitenoise[brotli]

Abra django-notes-master /settings.py, busque la lista MIDDLEWARE y agregue el middleware WhiteNoise justo después de SecurityMiddleware:

```
46 'django.contrib.staticfiles',
47 'tasks'
48 ]
49
50 MIDDLEWARE = [
51     'django.middleware.security.SecurityMiddleware',
52     'django.contrib.sessions.middleware.SessionMiddleware',
53     'django.middleware.common.CommonMiddleware',
54     'django.middleware.csrf.CsrfViewMiddleware',
55     'django.contrib.auth.middleware.AuthenticationMiddleware',
56     'django.contrib.messages.middleware.MessageMiddleware',
57     'django.middleware.clickjacking.XFrameOptionsMiddleware',
58     'whitenoise.middleware.WhiteNoiseMiddleware',
59 ]
60
61 (constant) ROOT_URLCONF = Literal['djangocrud.urls']
62
63 ROOT_URLCONF = 'djangocrud.urls'
64
65 TEMPLATES = [
66     {
67         'BACKEND': 'django.template.backends.django.DjangoTemplates',
68         'DIRS': [],
69         'APP_DIRS': True,
70         'OPTIONS': {
71             'context_processors': [
72                 'django.template.context_processors.debug',
73                 'django.template.context_processors.request',
74                 'django.contrib.auth.context_processors.auth',
75                 'django.contrib.messages.context_processors.messages',
76             ],
77         },
78     },
79 ]
```

Figura 44. Middleware.



Busque la sección donde se configuran los archivos estáticos. Aplicar las siguientes modificaciones:

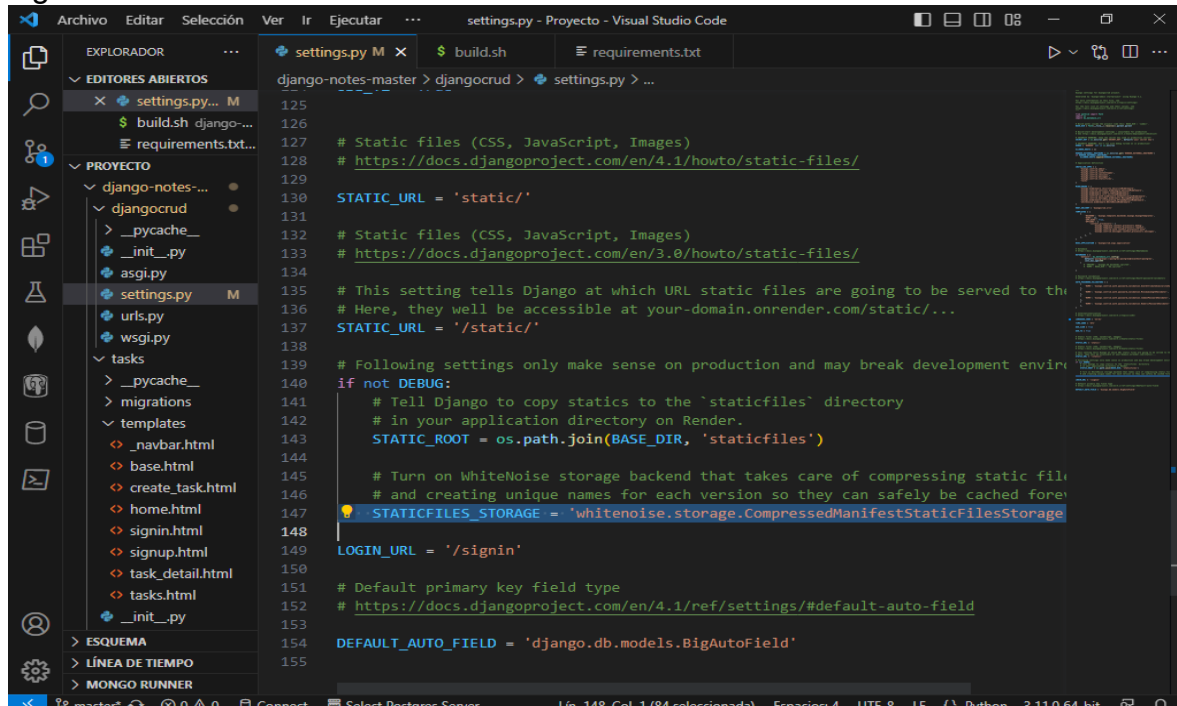


Figura 45. Archivos estáticos.

Crear un script de compilación

Necesitamos ejecutar una serie de comandos para construir nuestra aplicación. Podemos lograr esto con un script de compilación. Cree un script llamado build.sh en la raíz de su repositorio:

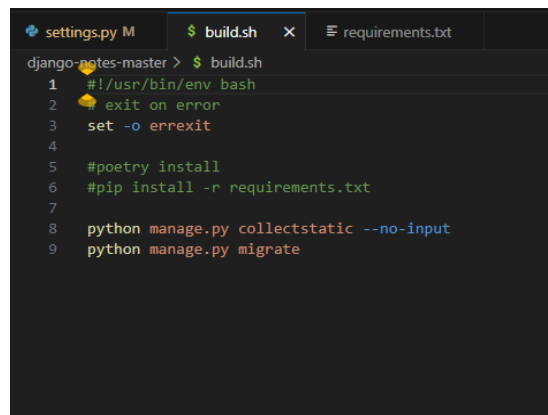


Figura 46. Build.sh



Implementación manual

Cree una nueva base de datos PostgreSQL en Render. Tenga en cuenta la URL de la base de datos interna de su base de datos; Lo necesitarás más tarde.

Cree un nuevo servicio web, apuntándolo a su repositorio de aplicaciones (otorgue permiso a Render para acceder a él si aún no lo ha hecho).

Seleccione Python para el tiempo de ejecución y establezca las siguientes propiedades:

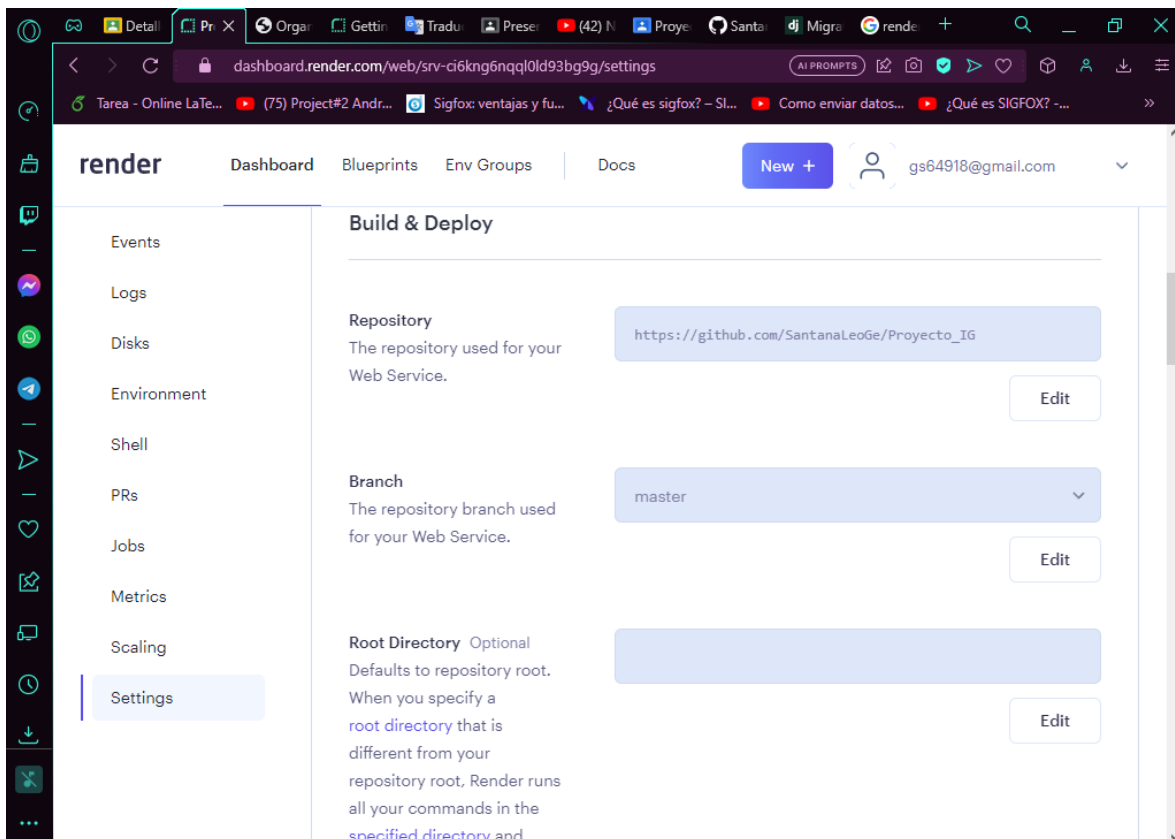


Figura 47. Modificación de variables en render.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

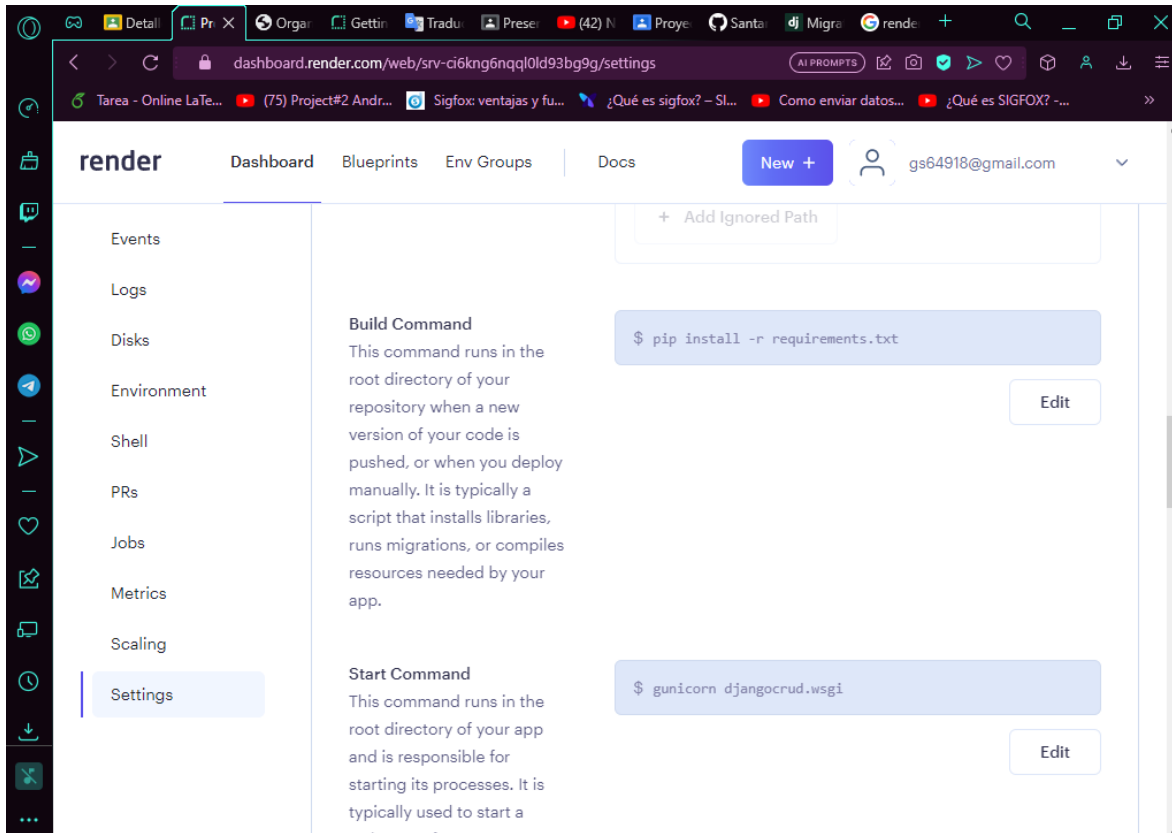


Figura 48. Modificación de variables en render.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

Nos redirecciona a la ventana de eventos donde se muestran las acciones realizadas.

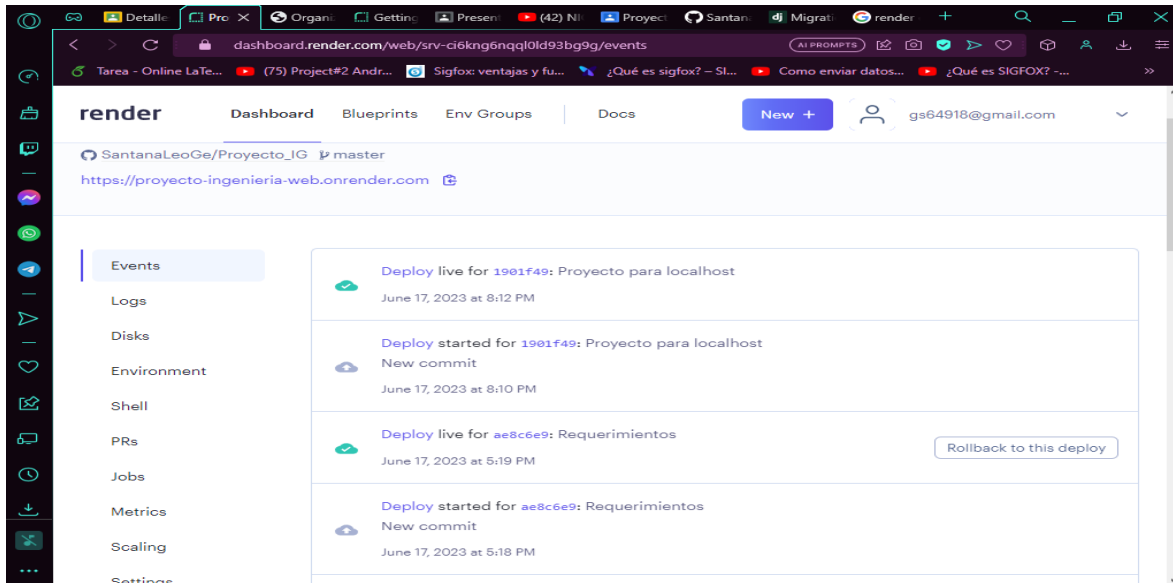


Figure 49. Eventos render

Al dar click en Despoly tendremos la información de nuestra página.

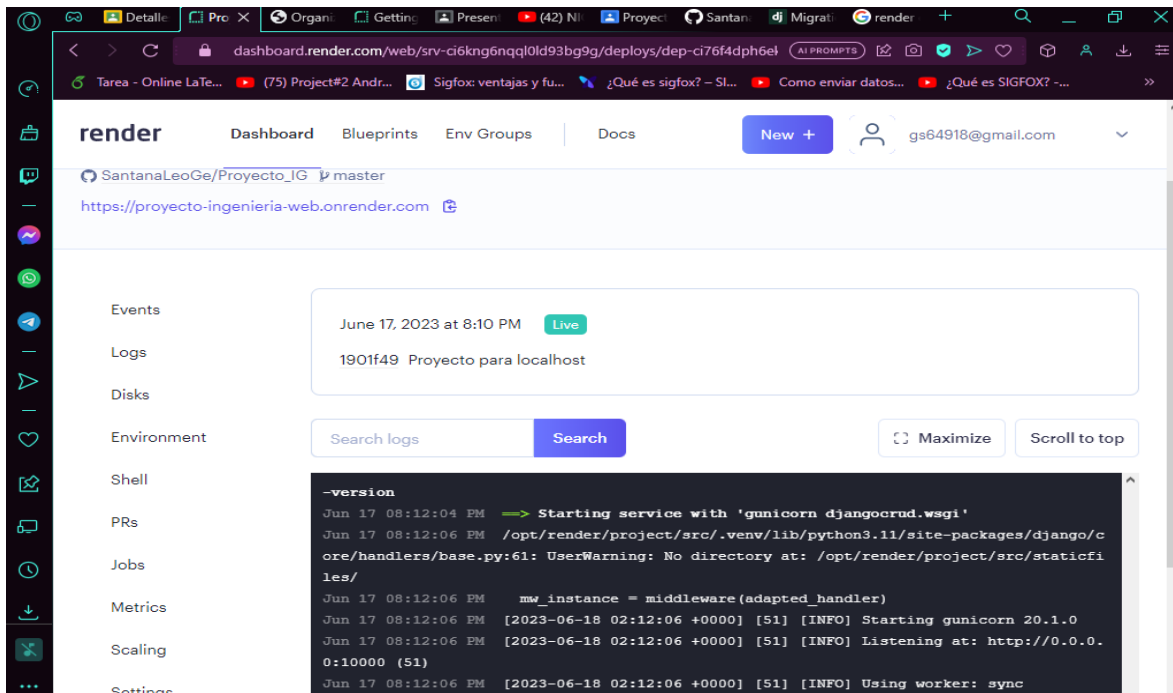


Figura 50. Servidor levantado.



UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS – IPN



Ingeniería web
1er Departamental
Profr. POLANCO MONTELONGO FRANCISCO ANTONIO
Academia de Telemática

Al entrar en el link que nos envía, podremos ver nuestra pagina web en línea con un servidor remoto.

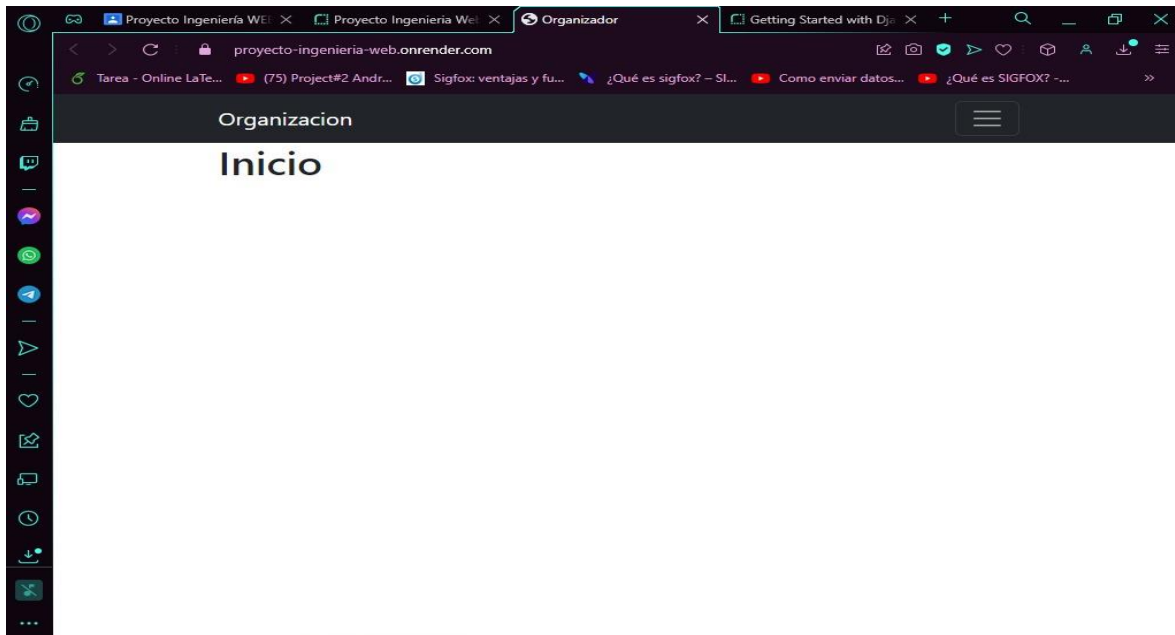


Figura 51. Inicio.

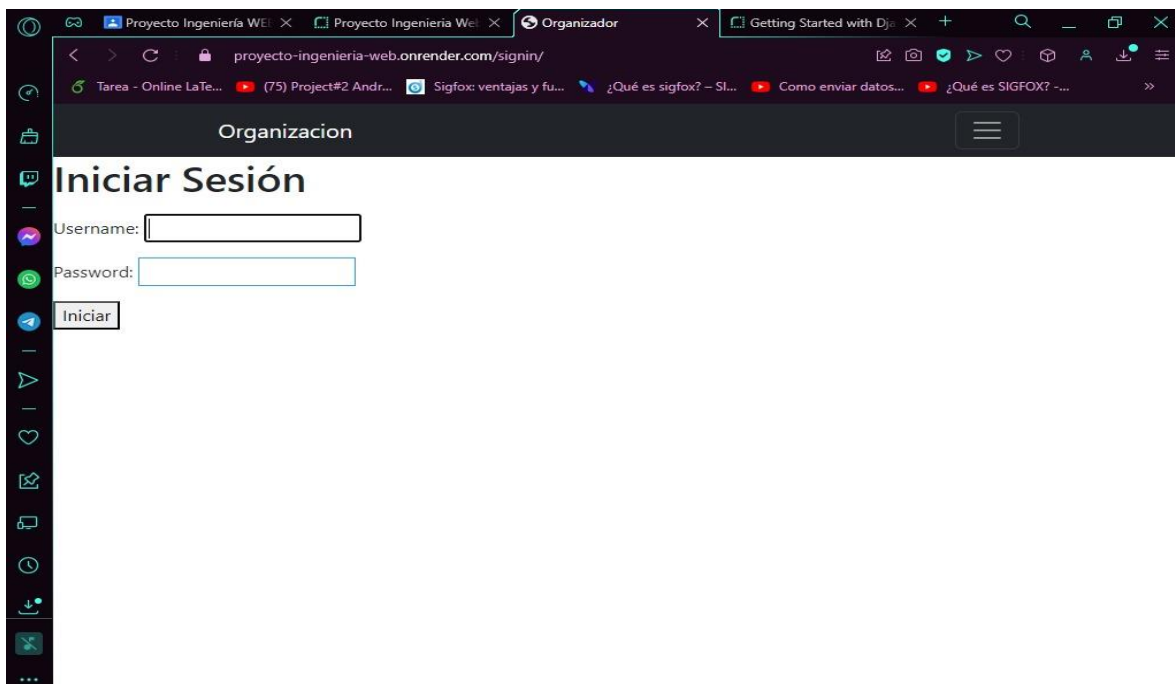


Figura 52. Inicio de sesión.



Figure 53. Registro.

Así ya esta lista la pagina web con un servicio de hosting en la nube, que permite ser visualizada.

Link: <https://proyecto-ingenieria-web.onrender.com>