Informe de Laboratorio 05

Tema: Django

Nota

Estudiante	Escuela	Asignatura
William Isaias Roque Quispe wroquequi@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III

Laboratorio	Tema	Duración
05	Django	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 8 junio 2023	Al 14 Junio 2023

1. Tarea

- Crea un blog sencillo en un entorno virtual utilizando la guía: `https://tutorial.djangogirls.org/es/django_start_project/`
- Especificar paso a paso la creación del blog en su informe.
- Crear un video tutorial donde realice las operaciones CRUD (URL public reproducible online)
- Adjuntar URL del video en el informe.

2. Equipos, materiales y temas utilizados

- Sistema Operativo
- GNU Vim.
- Python 3.11.3.
- Git
- Cuenta en GitHub con el correo institucional.
- Entorno virtual.
- Django 4.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/WilliamIsaiasRoque/Lab05-Pweb2.git`
- URL para el laboratorio 05 en el Repositorio GitHub.
- `https://github.com/rescobedoq/pw2/tree/main/labs/lab05`
- URL del video subido en Flip.
- `https://flip.com/groups/14644257/topics/36914916/responses/429073515/comments`

4. Actividades

4.1. Crear un directorio para el entorno virtual

- En este directorio inicializaremos el git y agregaremos un primer commit para probar.

Listing 1: Creando directorio de trabajo y accediendo en él

```
$ mkdir PWEB2lab/Lab05-Pweb2/
```

Listing 2: Dirijiéndonos al directorio de trabajo

```
$ cd PWEB2lab/Lab05-Pweb2/
```

Listing 3: Inicializando el repositorio Git y agregando remote add origin

```
$ git init
$ git remote add origin https://github.com/WilliamIsaiasRoque/Lab05-Pweb2.git
```

Listing 4: Añadiendo un README.md como primer push

```
$ echo "# Lab05-Pweb2" >> README.md
$ git add README.md
$ git commit -m "Initial commit"
$ git push -u origin main
```

4.2. Crear un entorno virtual en el directorio

Listing 5: Creando el directorio django_env y accediendo en esa carpeta

```
$ mkdir django_env
$ cd django_env
$ virtualenv -p python env
```

Listing 6: Activando el entorno virtual con source

```
$ source env/Scripts/activate
```

Listing 7: Instalando Django con pip

```
$ pip install Django
$ pip list
Package    Version
---------- -------
asgiref    3.7.2
Django     4.2.2
pip        23.1.2
sqlparse   0.4.4
```

Listing 8: Creando un proyecto Django llamado mysite

```
$ django-admin startproject mysite
```

4.3. Commits

Listing 9: Primero se crea una aplicación llamada blog

```
$ python manage.py startapp blog
```

- Los commits más importantes aparecerán en esta sección, al final se mostrarán unas capturas con todos los commits.
- Se ha modificado myenv/settings.py porque se agregó 'blog.apps.BlogConfig' para la correcta creación de la aplicación. También se cambió la zona horaria a 'America/Lima'

Listing 10: src/mysite/settings.py

```
"""
Django settings for mysite project.

Generated by 'django-admin startproject' using Django 4.2.2.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-69030o$q$)o6%#fs8au%5(6_j+3##-d%^=qw#n-#!nvihh95vo'

```

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog.apps.BlogConfig',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'mysite.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'mysite.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
```

```
    }
}


# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'America/Lima'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

- Se ha generado 'manage.py' cuando se creó el proyecto.

Listing 11: src/manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


```

```
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

- Se ha creado el modelo de Post en blog/apps.py y se ha importado a blog/admin.py

Listing 12: blog/apps.py

```
from django.apps import AppConfig


class BlogConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'blog'
```

Listing 13: blog/admin.py

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Listing 14: blog/models.py

```
from django.conf import settings
from django.db import models
from django.utils import timezone


class Post(models.Model):
    author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(
            default=timezone.now)
    published_date = models.DateTimeField(
            blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()
```

```
18
19      def __str__(self):
20          return self.title
```

- En el siguiente commit se aplicó makemigrations y migrate en el terminal, el código se generó automáticamente
- En mysite/urls.py se agregó 'include' a 'from django.urls import path'.

Listing 15: src/mysite/urls.py

```
1  """
2  URL configuration for mysite project.
3
4  The ‘urlpatterns‘ list routes URLs to views. For more information please see:
5      https://docs.djangoproject.com/en/4.2/topics/http/urls/
6  Examples:
7  Function views
8      1. Add an import: from my_app import views
9      2. Add a URL to urlpatterns: path('', views.home, name='home')
10 Class-based views
11     1. Add an import: from other_app.views import Home
12     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
13 Including another URLconf
14     1. Import the include() function: from django.urls import include, path
15     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
16 """
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('',include('blog.urls'))
23 ]
```

- Se he creado un html en blog/templates/blog/base.html y una hoja de estilo básica en blog/static/css/blog.css

Listing 16: src/blog/templates/blog/base.html

```
1  {% load static %}
2  <html>
3      <head>
4          <title>Blog Personalizado</title>
5          <link rel="stylesheet"
               href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
6          <link rel="stylesheet"
               href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
7          <link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext'
               rel='stylesheet' type='text/css'>
8          <link rel="stylesheet" href="{% static 'css/blog.css' %}">
9      </head>
10     <body>
11         <div class="page-header">
```

```
12              {% if user.is_authenticated %}
13              <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon
                    glyphicon-plus"></span></a>
14              {% endif %}
15              <h1><a href="/">Mi Blog Personal</a></h1>
16              <br>
17              <p class="name">ALUMNO: William Isaias Roque Quispe</p>
18          </div>
19          <div class="content container">
20              <div class="row">
21                  <div class="col-md-8">
22                  {% block content %}
23                  {% endblock %}
24                  </div>
25              </div>
26          </div>
27      </body>
28  </html>
```

Listing 17: src/blog/static/css/blog.css

```
1   h1 a, h2 a {
2       color: #FCA205;
3       font-family: 'Lobster';
4   }
5   .name {
6       color: #37cde1;
7       font-family: 'Comic Sans MS';
8       font-size: 12pt;
9       font-weight: bold;
10  }
11  .page-header {
12      background-color: #010106;
13      margin-top: 0;
14      padding: 20px 20px 20px 40px;
15  }
16
17  .page-header h1, .page-header h1 a, .page-header h1 a:visited, .page-header h1 a:active {
18      color: #ffffff;
19      font-size: 36pt;
20      text-decoration: none;
21      text-align: center;
22  }
23
24  .content {
25      margin-left: 40px;
26  }
27
28  h1, h2, h3, h4 {
29      font-family: 'Lobster', cursive;
30  }
31
32  .date {
33      color: #828282;
34  }
35
```

```
.save {
    float: right;
}

.post-form textarea, .post-form input {
    width: 100%;
}

.top-menu, .top-menu:hover, .top-menu:visited {
    color: #ffffff;
    float: right;
    font-size: 26pt;
    margin-right: 20px;
}

.post {
    margin-bottom: 70px;
}

.post h2 a, .post h2 a:visited {
    color: #000000;
}
```

- En el directorio raíz se agregó 'requirements.txt' para especificar las versiones utilizadas.

Listing 18: src/requirements.txt

```
asgiref   3.7.2
Django    4.2.2
pip       23.1.2
sqlparse  0.4.4
```

- Creación de un form inicial que permite crear nuevos posts.

Listing 19: src/blog/forms.py

```
from django import forms

from .models import Post

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        fields = ('title', 'text',)
```

- Se ha agreado post_list.html para la visualización de posts.

Listing 20: src/blog/templates/blog/post_list.html

```
{% extends 'blog/base.html' %}

```

```
{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.published_date }}
            </div>
            <h2><a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}</a></h2>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

- Creación de nuevos html para las operaciones CRUD.

Listing 21: src/blog/templates/blog/post_edit.html

```
{% extends 'blog/base.html' %}

{% block content %}
    <h2>Edit Post</h2>
    <form method="POST" class="post-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
    </form>
{% endblock %}
```

Listing 22: src/blog/templates/blog/post_detail.html

```
{% extends 'blog/base.html' %}

{% block content %}
    <div class="post">
        {% if post.published_date %}
            <div class="date">
                {{ post.published_date }}
            </div>
        {% endif %}
        {% if user.is_authenticated %}
        <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span
            class="glyphicon glyphicon-pencil"></span></a>
        <a class="btn btn-default" href="{% url 'post_delete' pk=post.pk %}"><span
            class="glyphicon glyphicon-trash"></span></a>
        {% endif %}
        <h2>{{ post.title }}</h2>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endblock %}
```

Listing 23: src/blog/templates/blog/post_delete.html

```
{% extends 'blog/base.html' %}

{% block content %}
    <h2>Delete post?</h2>
```

```
    <form method="POST" class="post-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Delete</button>
    </form>
{% endblock %}
```

- Modificaciones en urls.py y views.py para los html creados anteriormente.

Listing 24: src/blog/urls.py

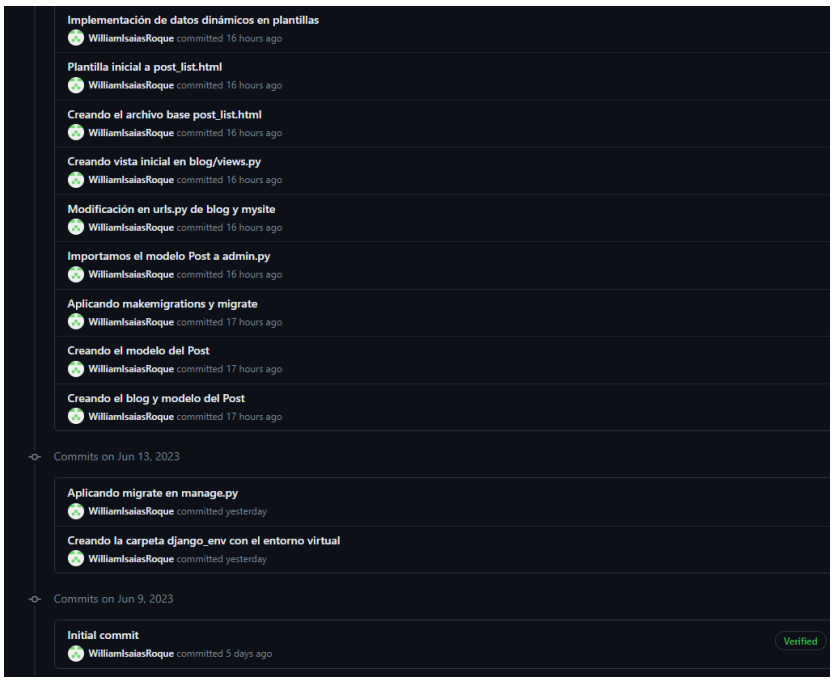
```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('post/new/', views.post_new, name='post_new'),
    path('post/<int:pk>/edit/', views.post_edit, name='post_edit'),
    path('post/<int:pk>/delete/', views.post_delete, name='post_delete'),
]
```

Listing 25: src/blog/views.py

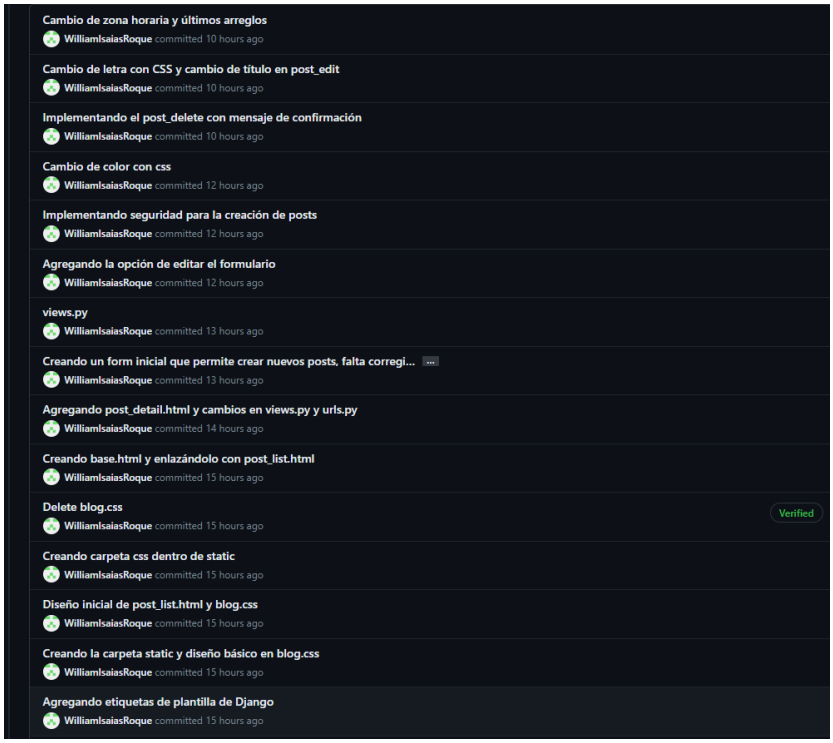
```
from django.shortcuts import render
from django.utils import timezone
from .models import Post
from django.shortcuts import render, get_object_or_404
from .forms import PostForm
from django.shortcuts import redirect

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {'posts': posts})

def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})

def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})

def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
```

```
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})

def post_delete(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        post.delete()
        return redirect('post_list')
    return render(request, 'blog/post_delete.html', {'post': post})
```

- Cabe aclarar que no se han agregado la mayoría de archivos de mysite, ya que no han sufrido ningún cambio en el archivo, tal es el caso de _init_.py , asgi.py y wsgi.py
- A continuación las capturas de todos los commits realizados en el repositorio GitHub



Primera parte de los commits realizados



Segunda parte de los commits realizados

5. Cuestionario

- ¿Cuál es un estándar de codificación para Python? Ejemplo: Para PHP en el proyecto Pear?

  PEP 8 es el estándar de codificación que es más aceptado para Python. Proporciona directrices para escribir código legible y consistente, como el uso de indentación de cuatro espacios, líneas de código limitadas a 79 caracteres, nombres de variables en minúsculas separados por guiones bajos, comentarios claros y concisos, importaciones ordenadas, espacios en blanco consistentes y convenciones de nombres.

- ¿Qué diferencias existen entre EasyInstall, pip, y PyPM?

  EasyInstall fue una herramienta anteriormente utilizada en Python para instalar paquetes y gestionar dependencias. Fue desarrollada por el proyecto setuptools y permitía descargar e instalar fácilmente paquetes desde PyPI. Sin embargo, EasyInstall ha sido reemplazado por pip, que se ha convertido en la herramienta estándar para la gestión de paquetes en Python debido a su mayor funcionalidad y soporte continuo. PyPM, por otro lado, es una herramienta específica de ActivePython que proporciona su propio sistema de gestión de paquetes para la distribución de paquetes de Python en ese entorno particular.

- En un proyecto Django que se debe ignorar para usar git. ¿Qué otros tipos de archivos se deberían agregar a este archivo?

  Archivos de configuración local: Si hay archivos de configuración local que contienen información sensible como claves secretas, contraseñas u otros datos específicos del entorno de desarrollo, es importante agregarlos al .gitignore para evitar su exposición accidental. Ejemplos comunes son settings_local.py o secrets.json.

- Utilice python manage.py shell para agregar objetos. ¿Qué archivos se modificaron al agregar más objetos?

  Al utilizar python manage.py shell en un proyecto Django para agregar objetos a través de la consola interactiva, los archivos que suelen modificarse son los archivos de código fuente de las aplicaciones, como models.py, views.py y forms.py, donde se agregan las definiciones de los nuevos objetos. Además, también pueden generarse archivos de migración en el directorio migrations, reflejando los cambios en la estructura de la base de datos. Estas modificaciones aseguran que los nuevos objetos y su lógica asociada se integren correctamente en el proyecto Django. Generalmente se modifican dos tipos de archivos:

  - - Archivos de código fuente: Tal es el caso de models.py, views.py o forms.py.
  - - Archivos de migraciones: Tienen el nombre de 00x_nombre_migracion.py, cada archivo de migración tiene un número secuencial.

6. Referencias

- `https://tutorial.djangogirls.org/es/django_start_project/`
- `https://docs.python.org/`
- `https://realpython.com/python-django-blog/`