Informe de Django 3 Templates

Tema: Django Templates

Nota

Estudiante	Escuela	Asignatura
Victor Gonzalo Maldonado Vilca vmaldonadov@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Tarea	Tema	Duración
07	Django Templates	2 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 16 de mayo de 2024	Al 15 de junio de 2024

1. Introducción

Las plantillas en Django son archivos HTML que permiten generar contenido dinámico al integrar código Python. Se almacenan en el directorio templates, se extienden usando herencia, y se manipulan con etiquetas y variables de Django para mostrar datos dinámicos y aplicar filtros. Son fundamentales para crear interfaces web dinámicas y flexibles en aplicaciones Django.

2. Objetivos

- Generar contenido HTML dinámico basado en datos de la aplicación.
- Reutilizar código mediante la creación de plantillas base y herencia.
- Mantener la separación entre la lógica y la presentación en las aplicaciones web.
- Crear interfaces interactivas utilizando etiquetas, variables y filtros.
- Optimizar el desarrollo al agilizar el manejo de la presentación de datos en Django.

3. Tarea

- En esta tarea deberá seguir las diapositivas de DJango03, dentro de un proyecto git local.
- Deberá hacer un commit por cada paso y deberá usar branch para hacer sus experimentos o pruebas.

- La entrega de la tarea consistirá de texto con las respuestas a las preguntas de las diapositivas (incluya el enunciado de la diapositiva) y una captura de pantalla (deberá incrustar la captura de pantalla al inicio de su texto de respuesta) del siguiente comando:

```
git log --graph --pretty=oneline --abbrev-commit --all
```

- NO INCLUYA commits de tareas pasadas.
- Cada commit debe ser realizado con un mensaje descriptivo del paso de la diapositiva que estuvo siguiendo

4. Entregables

- Informe hecho en Latex.
- **texto:** Respuestas a las preguntas de las diapositivas
- Captura de pantalla del siguiente comando:

```
git log --graph --pretty=oneline --abbrev-commit --all
```

- **Extra:** URL - Repositorio GitHub

5. Equipos, materiales y temas utilizados

- Sintaxis de plantillas en Django.
- Herencia de plantillas y bloques.
- Contexto de plantillas y vistas en Django.
- Uso avanzado de plantillas.

6. Respuestas a las Preguntas dejadas en las Diapositivas

1. **Pruebe qué ocurre si no incluímos el campo “donador” en el formulario¿De qué se trata este error?**
   Respuesta: Este error trata de que si el campo 'donador' es requerido en el modelo Persona y no se incluye en el formulario al intentar guardar un objeto de Persona con este formulario, se producirá un error de validación. Debido a que Django espera que todos los campos requeridos sean proporcionados al crear o actualizar una instancia del modelo.
2. **¿Cómo se puede solucionar?**
   Respuesta: La solución más obvia es volver a agregar el campo en el formulario. Otra solución es establecer un valor por defecto usando 'default=True' o 'default=False'. Recordemos que el valor por defecto cambiará dependiendo del tipo de dato que sea el campo, en este caso, es booleano.
3. **Del ejemplo anterior vemos que la vista se llama primero con GET y luego con POST ¿En qué momento se hace llamada GET y en qué momento se hace la llamada POST?**
   Respuesta: La llamada GET se realiza al enviar el formulario con 'method=GET', mientras que la llamada POST se realiza al enviar el formulario con 'method=POST' y el token CSRF. Pero su principal diferencia es que el método GET se usa comúnmente para solicitar recursos o enviar datos de manera visible en la URL, mientras que el método POST se usa para enviar datos de manera más segura y sin mostrarlos directamente en la URL.

7. Captura de Pantalla

```
Usuario@DESKTOP-G0S7RR6 MINGW64 /D/UNSA/pw2/teoria/Django2/src (master)
$ git log --graph --pretty=oneline --abbrev-commit --all
* b3dbee5 (HEAD -> master, origin/master) Finalizacion de la actividad Django3 templates, donde se h
izo uso de las plantillas, ademas de filtros, tag if y for, incluso variables predefinidas
* 3fa03f1 ¿En qué momento se hace llamada GET y en qué momento se hace la llamada POST?: La llamada
GET se realiza al enviar el formulario con 'method=GET', mientras que la llamada POST se realiza al
enviar el formulario con 'method=POST' y el token CSRF. Pero su principal diferencia es que el métod
o GET se usa comúnmente para solicitar recursos o enviar datos de manera visible en la URL, mientras
 que el método POST se usa para enviar datos de manera más segura y sin mostrarlos directamente en l
a URL.
* adff393 Modificando vista searchForHelp para procesar la búsqueda y obtener el nombre de búsqueda
desde el formulario POST.
* 38ca009 Uso del tag csrf_token en el formulario debido a que se cambio el method de GET a POST, re
cordemos que el tag anteriormente mencionado hace que nuestra página web sea más segura. Recordemos
que Django no te permite el uso del method POST sino se usa csrf_token
* 07c2586 Cambio de action en el formulario de ser a la documentación de Django a ser search, además
 cambio en la vista de searcForHelp donde se imprime tanto el metodo GET como POST
* d3e9cd4 Asociamos la url 'search/' a la vista searchForHelp, para poder visualizar la pantilla al
activar el servidor e ir a la ruta respetiva
* 07a1eb0 Se creo una nueva vista searchForHelp que nos renderiza a la plantilla search.html haciend
o uso de la funcion render
* f873cf0 Se creo plantilla search.html donde se implemento un formulario de búsqueda con método GET
 para buscar en la documentación de Django.
* 8c5e01d ¿Cómo se puede solucionar?: La solución más obvia es volver a agregar el campo en el formu
lario. Otra solución es establecer un valor por defecto usando 'default=True' o 'default=False'. Rec
ordemos que el valor por defecto cambiará dependiendo del tipo de dato que sea el campo, en este cas
o, es booleano.
* 0a20116 ¿De qué trata este error?: Este error trata de que si el campo 'donador' es requerido en e
l modelo Persona y no se incluye en el formulario al intentar guardar un objeto de Persona con este
formulario, se producirá un error de validación. Debido a que Django espera que todos los campos req
ueridos sean proporcionados al crear o actualizar una instancia del modelo.
* 1b72501 Se elimino campo Donador del formulario salio un error, llegado a este punto se respondera
n las preguntas correspondientes
* 3a47da1 En el archivo urls.py del proyecto se añadio la ruta para la vista personaCreateView en ur
lpatterns, donde la ruta es 'agregar/'
* 6303cfd Se creo plantilla HTML (personasCreate.html) para la creación de personas usando PersonaFo
rm con protección CSRF y método POST, uso de etiqueta <form>
```

Figura 1: Primera Parte commits

```
* 8abe126 Implementando vista para la creación de personas con validación de formulario (uso de is_v
alid()) y renderización a la plantilla personasCreate.html
* a2b68c6 Se creo archivo forms.py donde se implemento el formulario PersonaForm para el modelo Pers
ona con campos específicos (nombre, apellidos, edad, email, donador, dni y telefono)
* dec5605 En views.py de la app personas se cambio la plantilla de uso de test.html a description.ht
ml, donde se va a generar una respuesta HTML.
* b261e7a Corrigiendo error sintactico en el diccionario se uso el = en ves de :, además en test.htm
l se modifico el extraer la informacion traida de las vistas, debido que ahora la vista envia un obj
eto entero
* 1cdf634 Cambiando diccionario de las vistas ahora se envia el objeto entero a la plantilla corresp
ondiente
* 24a37a1 Corrigiendo errores en views.py de la app personas, se hizo la importacion de personaTestV
iew, además se coloco una ',' en la funcion path, debido a que era un error sintactico
* cc3a9bb Implementando estructura html (template) en personas/test.html, uso de etiquetas if donde
se verificara si la edad es mayor a 40, se imprimira mensajes dependiendo de la edad
* d357142 Se agrego nuevas Url y vistas en urlpatterns, en este caso Ruta para la vista de prueba de
 persona ('persona/'): personaTestView, además se cambio el nombre de la vista de personaView a pers
onaTestView
* b1998d1 Definiendo vista personaView en views.py donde se obtendra informacion de la base de datos
 y se contendra en los elementos del diccionarios (nombre y edad), la información se enviara a la pl
antilla test/html
* 5f6257e Uso de filtros, en el template se hizo uso de filtros como upper y divisibleby, donde uppe
r convierte el texto en mayúscula y divisibleby returna verdadero si el elemento es divisible por al
gun numero dado 'dividibleby:2'
*   5e4642c Solucionando conflictos del merge con la rama pruebas, en la rama pruebas se realizo pru
ebas respectivas usando el tag if
|\
| * e7843f9 (origin/pruebas, pruebas) Prueba siguiente donde se uso opradores <= y >= donde se ve qu
e funciona correctamente, se imprime mensajes cuando el elemento del diccionario es menor o igual a
44 de igual manera si es mayor o igual
| * 45c0a27 Haciendo prueba usando operadores != y < en el tag if, donde se observo que funciona cor
rectamente el uso de estos operadores
* | d33dd96 Uso del tag if, ya que el DTL tambien posee condiconales, se uso para verificar si el it
em de la lista es igual a 33 o si es mayor que 50, en los dos casos se imprimirá un respectivo mensa
je
* | ccc9d3d Solucionando conflictos merge en los archivos home.html y settings.py, en la rama prueba
s se hzo uso de variables predefinidas del tag for
|\|
```

Figura 2: Segunda Parte commits

```
|\|
| * d144818 Prueba de vatiable predefinida forloop.parentloop lo que hace es dar una referencia al o
bjeto forloop del bucle exterior.
| * 8b67794 Prueba de la variable predefinida forloop.last donde se observa que se imprime Verdadero
 si este es el ultimo ciclo del bucle.
| * 02fa059 Prueba de la variable predefinida forloop.first donde se observa que se imprime Verdader
o si este es el primer ciclo del bucle.
| * cf24827 Haciendo prueba con la variable predefinida forloop.revcounter0 donde es simillar a la
anterior variable solo que esta vez toma en cuenta el numero 0
| * 12c2e95 Haciendo prueba con la variable predefinida forloop.revcounter se puede observar que el
conteo es de orden inverso iniciando desde 3 y teminando en 1
| * ba5a8dc Haciendo prueba con la variable predefinida forloop.counter0 se puede observar que todos
 el coteo de elementos inicia desde 0
* | 20e4cca Uso de la variable predefinidad 'forloop.counter' del tag for en el template home.html,
donde se observa que a se conto los elementos de la lista apareciendo el numero a lado de cada eleme
nto
* | 9915c7b Uso del tag for para poder recorrer la lista enviada del diccionario, y poder visualizar
la mediante el template debido al uso de la etiqueta <li></li>
* | 426b84d Se añadio nuevo elemento en el diccionario, una lista para pode concluir que no solo se
pueden enviar datos simples al template
* | c2055e3 Continuando con Django 3 - templates, se modifico la estructura html para que se pueda v
isualizar las variables enviadas, debido a que los elementos del diccionario encontrado en vistas so
n variables en la plantilla
* | c33adde Modificando vistas, para que se pueda enviar variables del contexto a una plantilla, se
creo un diccionario y se envio a la funcion reader
```

Figura 3: Tercera Parte commits

8. URL de Repositorio Github

- URL del Repositorio GitHub.
- `https://github.com/Victor-Gonzalo-Maldonado-Vilca/Django2.git`

9. Metodología

Debido a que es un proyecto continuo - La metodología ya esta empleada en las actividades de Django1 y Django2

10. Desarrollo del trabajo

Continuamos a partir del proyecto realizado en Django 2

10.1. Modelos – Aplicación personas

El modelo `Persona` en Django define los atributos que se utilizarán para almacenar información sobre una persona en la base de datos. Cada atributo tiene un tipo de datos específico y ciertas reglas asociadas:

- `nombre`: Se trata de un campo de texto que puede almacenar hasta 100 caracteres y es obligatorio.
- `apellidos`: Otro campo de texto que también puede contener hasta 100 caracteres y debe ser completado.
- `edad`: Este campo guarda un número entero que representa la edad de la persona.
- `email`: Aquí se almacena la dirección de correo electrónico de la persona, con un límite de 100 caracteres.
- `donador`: Un campo booleano que indica si la persona es donante o no. Por defecto, se establece como `False` si no se especifica lo contrario.
- `dni`: Se utiliza para guardar el número de documento nacional de identidad (DNI) de la persona, aunque este campo puede quedar vacío si no se proporciona.
- `telefono`: Este campo guarda el número de teléfono de la persona y también puede quedar sin completar si no se desea proporcionar esta información.

Estos atributos permiten almacenar información relevante sobre una persona de manera organizada en la base de datos de la aplicación Django. Después se harán las migraciones correspondientes.

Listing 1: Modelo Persona

```
class Persona(models.Model):
  nombre   = models.CharField(max_length = 100, blank=False)
  apellidos = models.CharField(max_length = 100, blank=False)
  edad     = models.IntegerField()
  email    = models.CharField(max_length = 100)
  donador  = models.BooleanField(default=False)
  dni      = models.CharField(max_length = 8, null=True)
  telefono = models.CharField(max_length = 9, null=True)
```

10.1.1. Examinando los objetos del Modelo

```
(lab08) PS D:\UNSA\PW2\Teoria\Django2\src\listaContactos> python manage.py shell
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from personas.models import Persona
>>> Persona.objects.all()
<QuerySet [<Persona: Persona object (1)>, <Persona: Persona object (2)>, <Persona: Persona object (3
)>, <Persona: Persona object (4)>, <Persona: Persona object (5)>, <Persona: Persona object (6)>, <Pe
rsona: Persona object (7)>, <Persona: Persona object (8)>, <Persona: Persona object (9)>]>
>>> obj = Persona.objects.get(id = 2)
>>> dir(obj)
['DoesNotExist', 'MultipleObjectsReturned', '__class__', '__delattr__', '__dict__', '__dir__', '__do
c__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '_
_init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook
__', '__weakref__', '_check_column_name_clashes', '_check_constraints', '_check_db_table_comment', '
_check_default_pk', '_check_field_name_clashes', '_check_fields', '_check_id_field', '_check_index_t
ogether', '_check_indexes', '_check_local_fields', '_check_long_column_names', '_check_m2m_through_s
ame_relationship', '_check_managers', '_check_model', '_check_model_name_db_lookup_clashes', '_check
_ordering', '_check_property_name_related_field_accessor_clashes', '_check_single_primary_key', '_ch
eck_swappable', '_check_unique_together', '_do_insert', '_do_update', '_get_FIELD_display', '_get_ex
or_references', '_get_field_value_map', '_get_next_or_previous_by_FIELD', '_get_next_or_previous_in_
order', '_get_pk_val', '_get_unique_checks', '_meta', '_perform_date_checks', '_perform_unique_check
s', '_prepare_related_fields_for_save', '_save_parents', '_save_table', '_set_pk_val', '_state', '_v
alidate_force_insert', 'adelete', 'apellidos', 'arefresh_from_db', 'asave', 'check', 'clean', 'clean
_fields', 'date_error_message', 'delete', 'dni', 'donador', 'edad', 'email', 'from_db', 'full_clean'
, 'get_constraints', 'get_deferred_fields', 'id', 'nombre', 'objects', 'pk', 'prepare_database_save'
, 'refresh_from_db', 'save', 'save_base', 'serializable_value', 'telefono', 'unique_error_message',
'validate_constraints', 'validate_unique']
```

Figura 4: Examinar objetos del Modelo

10.2. Vistas – Aplicación personas

Estos son tres views en Django. personaTestView muestra los detalles de una persona según su ID. personaCreateView procesa un formulario para crear una nueva persona. searchForHelp busca ayuda mediante un formulario de búsqueda y muestra los resultados.

```python
from django.shortcuts import render
from .models import Persona
from .forms import PersonaForm

# Create your views here.

def personaTestView(request):
    obj = Persona.objects.get(id = 1)
    context = {
        'objeto': obj,
    }
    return render(request, 'personas/description.html', context)

def personaCreateView(request):
    form = PersonaForm(request.POST or None)
    if form.is_valid():
        form.save()
        form = PersonaForm()

    context = {
        'form': form
    }
```

```
        return render(request, 'personas/personasCreate.html', context)

    def searchForHelp(request):
        print(request)
        if request.method == 'POST':
            nombre = request.POST.get('q')
            print(nombre)
        context = {}
        return render(request, 'personas/search.html', context)
```

10.3. Vistas – Aplicación inicio

Estos son dos views básicos en Django. myHomeView renderiza una plantilla llamada "home.htmlçon un contexto que incluye un texto, un número y una lista. Mientras tanto, anotherView devuelve directamente una respuesta HTTP con un título en formato HTML.

```
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.

def myHomeView(request, *args, **kwargs):
    myContext = {
        'myText': 'Esto es sobre nosostros',
        'myNumber': 123,
        'myList': [33, 44, 55],
    }
    return render(request, "home.html", myContext)

def anotherView(request):
    return HttpResponse('<h1>Solo otra pagina</h1>')
```

10.4. URLs – proyecto listaContactos

Esto configura las rutas URL para las diferentes vistas de tu aplicación Django:

- La primera ruta, `'admin/'`, permite acceder al panel de administración de Django a través de la URL `/admin/`.
- La segunda ruta, `''` (página de inicio), está asociada a la vista `myHomeView` definida en `inicio.views`. Esta ruta representa la página principal de tu aplicación y se accede a través de la URL base del sitio.
- La tercera ruta, `'another/'`, está asociada a la vista `anotherView` de `inicio.views`. Esta ruta permite acceder a una página adicional dentro de tu aplicación mediante la URL `/another/`.
- La cuarta ruta, `'persona/'`, está asociada a la vista `personaTestView` de `personas.views`. Al acceder a esta URL (`/persona/`), se mostrará la información de una persona específica.
- La quinta ruta, `'agregar/'`, está asociada a la vista `personaCreateView` de `personas.views`. Esta ruta permite agregar una nueva persona a través de la URL `/agregar/`.
- La sexta ruta, `'search/'`, está asociada a la vista `searchForHelp` de `personas.views`. Esta ruta se utiliza para buscar ayuda dentro de la aplicación y se accede mediante la URL `/search/`.

```
from django.contrib import admin
from django.urls import path
from inicio.views import myHomeView, anotherView
from personas.views import personaTestView, personaCreateView, searchForHelp

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', myHomeView, name='Pagina de Inicio'),
    path('another/', anotherView, name='otro'),
    path('persona/', personaTestView, name='testViewPersona'),
    path('agregar/', personaCreateView, name='createPersona'),
    path('search/', searchForHelp, name='buscar'),
]
```

11. Templates

11.1. Plantilla home.html

El siguiente código define una página HTML que utiliza plantillas de Django. La página se extiende de otra plantilla llamada `base.html`, lo que significa que hereda su estructura y estilos, y luego agrega contenido adicional en un bloque específico llamado `content`. Dentro del bloque `content`, se incluyen varios elementos HTML:

- Un encabezado de nivel 2 (`<h2>`) que muestra el texto "Hola Mundo desde Django".
- Un encabezado de nivel 3 (`<h3>`) con el texto Çon Templates".
- Un párrafo (`<p>`) que muestra el contenido de la variable `myText` en mayúsculas y truncado a 16 caracteres, seguido por el valor de la variable `myNumber`.
- Una lista desordenada (`<ul>`) generada mediante un bucle `for`, donde cada elemento de la lista se obtiene de la variable `myList`.
- Cada elemento de la lista muestra su posición utilizando `forloop.counter` y el valor de la variable incrementado en 2 usando `|add:2`.
- Se incluyen condiciones `if`, `elif` y `else` dentro del bucle para agregar texto adicional según el valor de cada elemento en `myList`.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Template Base</title>
  </head>
  <body>
    {% extends 'base.html' %}
    {% block content %}
    <h2>Hola Mundo desde Django</h2>
    <h3>Con Templates</h3>
    <p>
      {{ myText|upper|truncatechars:"16" }},
      {{ myNumber }}
    </p>
    <ul>
```

```
        {% for myItem in myList %}
          <li>{{forloop.counter}} - {{ myItem|add:2 }}
          {% if myItem == 33 %}
            - Este numero es magico
          {% elif myItem|divisibleby:2 %}
            - Este numero es par
          {% elif myItem > 50 %}
            - Este numero es mayor que 50
          {% endif %}
          </li>
        {% endfor %}
      </ul>
      {% endblock %}
    </body>
  </html>
```

EJECUCIÓN:

- Home
- Primera
- Segunda
- Tercera
- Cuarta
- Quinta
- Sexta

Mi Página web usando Django

Hola Mundo

DJANGO

Este es un texto de Base

Hola Mundo desde Django

Con Templates

ESTO ES SOBRE N..., 123

- 1 - 35 - Este numero es magico
- 2 - 46 - Este numero es par
- 3 - 57 - Este numero es mayor que 50

Figura 5: Ejecución

11.2. Plantilla description.html – templates/personas/

Este código HTML es una plantilla de Django que extiende otra plantilla llamada 'base.html' y define un bloque de contenido. Dentro de este bloque, se muestra el nombre y la edad de un objeto utilizando variables de Django (objeto.nombre y objeto.edad). Además, incluye una condición para mostrar un mensaje basado en la edad del objeto: si la edad es mayor que 40, muestra "debe cuidarse del colesterol", de lo contrario, muestra "puede comer lo que quiera".

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>test</title>
  </head>
  <body>
    {% extends 'base.html' %}
    {% block content %}
    <h1>{{objeto.nombre}}</h1>
    <h2>
      {% if objeto.edad > 40 %}
      debe cuidarse del colesterol
      {% else %}
      puede comer lo que quiera
      {% endif %}
    </h2>
    {% endblock %}
  </body>
</html>
```

EJECUCIÓN:

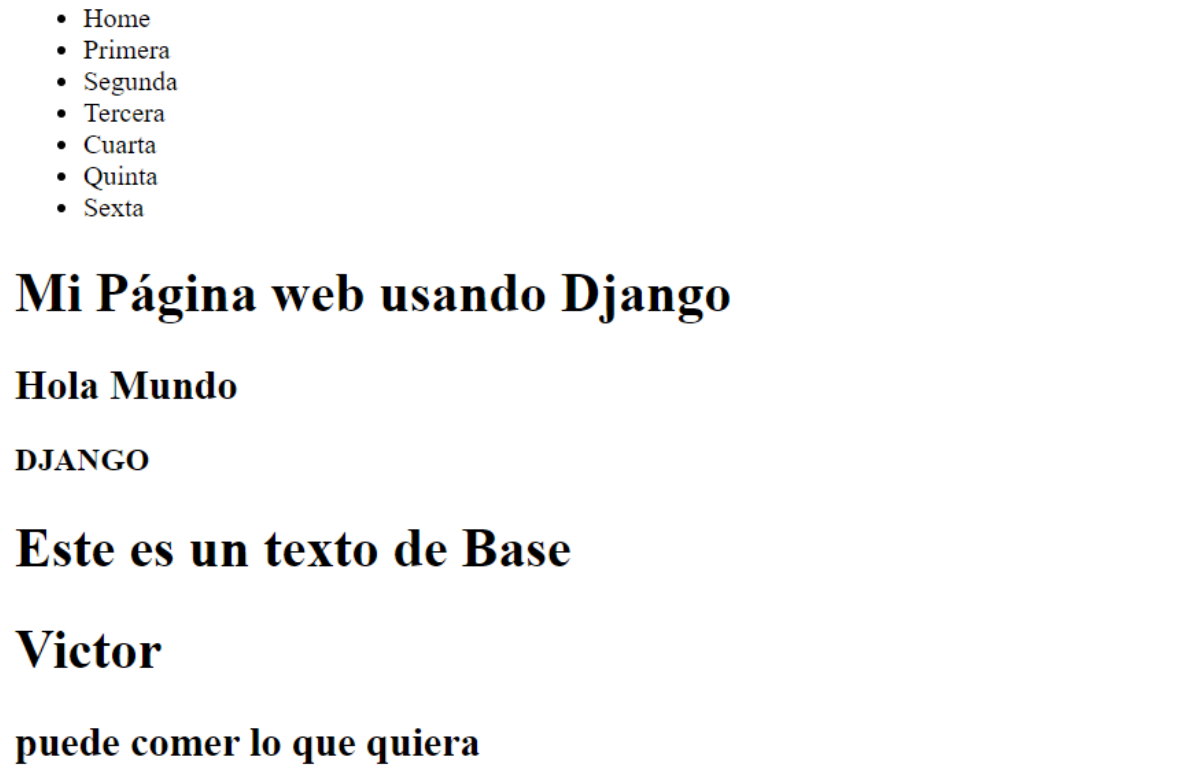

Figura 6: Ejecución

11.3. Plantilla personasCreate.html – templates/personas/

Este código HTML es una plantilla de Django que se utiliza para crear un nuevo objeto de tipo Persona. La plantilla extiende otra plantilla llamada 'base.html' y define un bloque de contenido donde se encuentra un formulario. Este formulario utiliza el método POST y está protegido contra ataques CSRF mediante el token 'csr ftoken'. El formulario se renderiza utilizando el objeto 'form.as p', que genera los campos del formulario como párrafos. Finalmente, hay un botón de tipo submit con el texto 'Grabar' para enviar el formulario y crear el objeto Persona.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Create Persona</title>
  </head>
  <body>
    {% extends 'base.html' %}
    {% block content %}
```

```
        <form method='POST'> {% csrf_token %}
          {{ form.as_p }}
          <input type='submit' value='Grabar'/>
        </form>
        {% endblock %}
      </body>
    </html>
```

EJECUCIÓN:

- Home
- Primera
- Segunda
- Tercera
- Cuarta
- Quinta
- Sexta

Mi Página web usando Django

Hola Mundo

DJANGO

Este es un texto de Base

Nombre:

Apellidos:

Edad:

Email:

Dni:

Telefono:

Grabar

Figura 7: Ejecución

11.4. Plantilla search.html – templates/personas/

Esta plantilla HTML forma parte de un formulario de búsqueda en Django. Está diseñada para integrarse con otra plantilla llamada 'base.html' y presenta un bloque de contenido que incluye el formulario. Utiliza el método POST para enviar datos y asegura la protección contra ataques CSRF con un token especial. Dentro del formulario, se encuentra un campo de texto para ingresar el término de búsqueda y un botón para activar la búsqueda.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>search</title>
  </head>
  <body>
    {% extends 'base.html' %}
    {% block content %}
    <form action='' method='POST'> {% csrf_token %}
      <input type='text' name='q' placeholder="buscar"/>
      <input type='submit' value='Buscar'/>
    </form>
    {% endblock %}
  </body>
</html>
```

EJECUCIÓN:

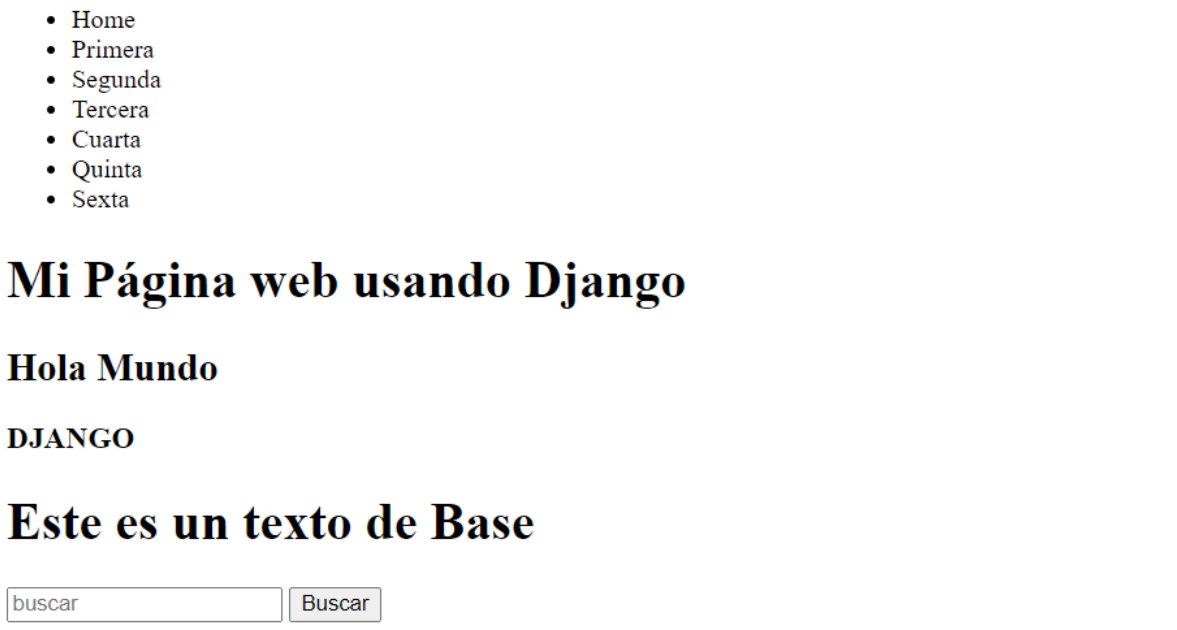

Figura 8: Ejecución

```
(lab08) PS D:\UNSA\PW2\Teoria\Django2\src\listaContactos> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 13, 2024 - 16:28:18
Django version 5.0.6, using settings 'listaContactos.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

<WSGIRequest: POST '/search/'>
Victor
[13/Jun/2024 16:28:27] "POST /search/ HTTP/1.1" 200 1161
```

Figura 9: servidor - POST

```
(lab08) PS D:\UNSA\PW2\Teoria\Django2\src\listaContactos> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 13, 2024 - 16:30:42
Django version 5.0.6, using settings 'listaContactos.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

<WSGIRequest: GET '/search/?q=Victor'>
[13/Jun/2024 16:30:45] "GET /search/?q=Victor HTTP/1.1" 200 1038
```

Figura 10: servidor - GET

11.5. Ejecución del servidor

Este siguiente comando inicia el servidor local de Django. Una vez ejecutado, el servidor estará disponible por defecto en la dirección `http://127.0.0.1:8000/`. Desde esta dirección, se puede acceder a las diferentes vistas y funcionalidades del proyecto Django.

```
python manage.py runserver
```

11.6. Uso de GitHub

11.6.1. Usuario

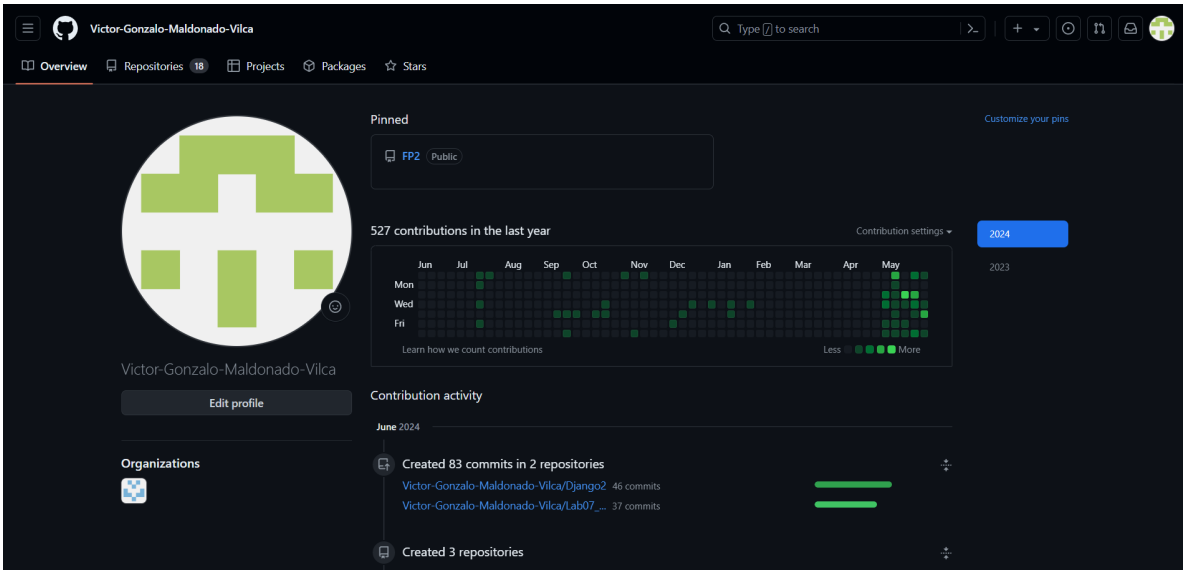

Figura 11: Usuario

11.6.2. Registro de cambios en mi código

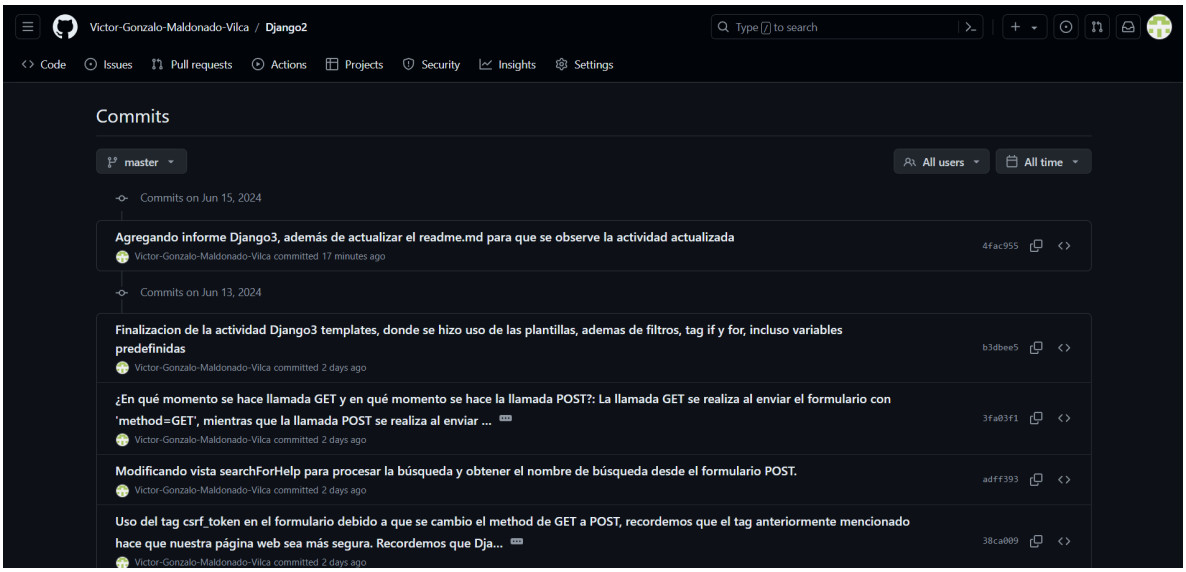

Figura 12: Commits

11.6.3. Uso de Ramas

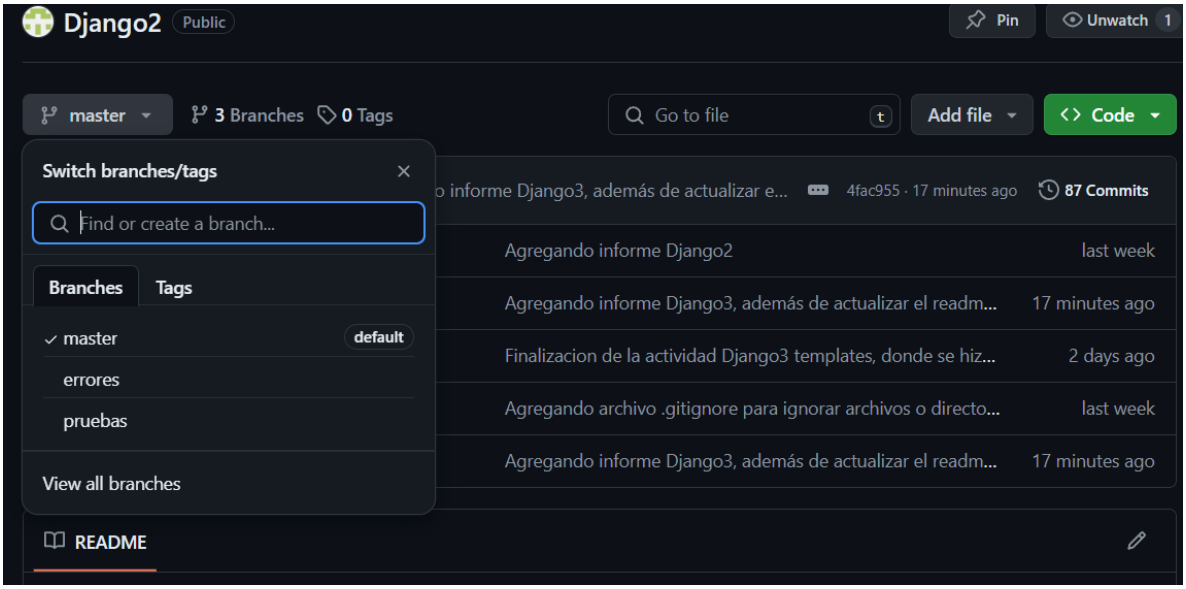

Figura 13: Ramas

11.6.4. Repositorio

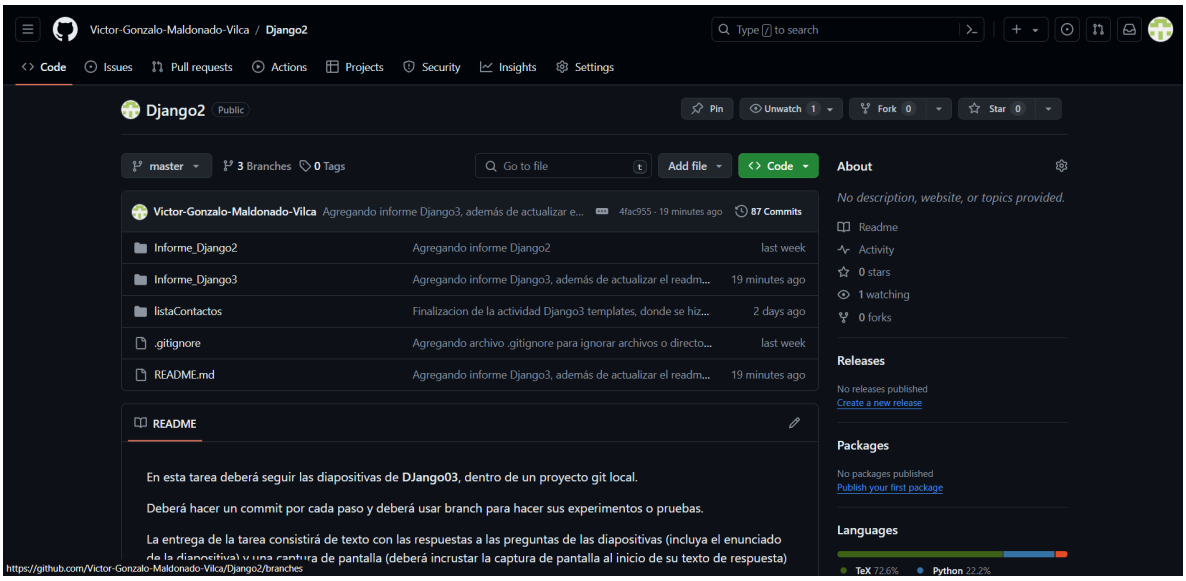

Figura 14: Repositorio

11.6.5. Proyecto compartido con el profesor de github

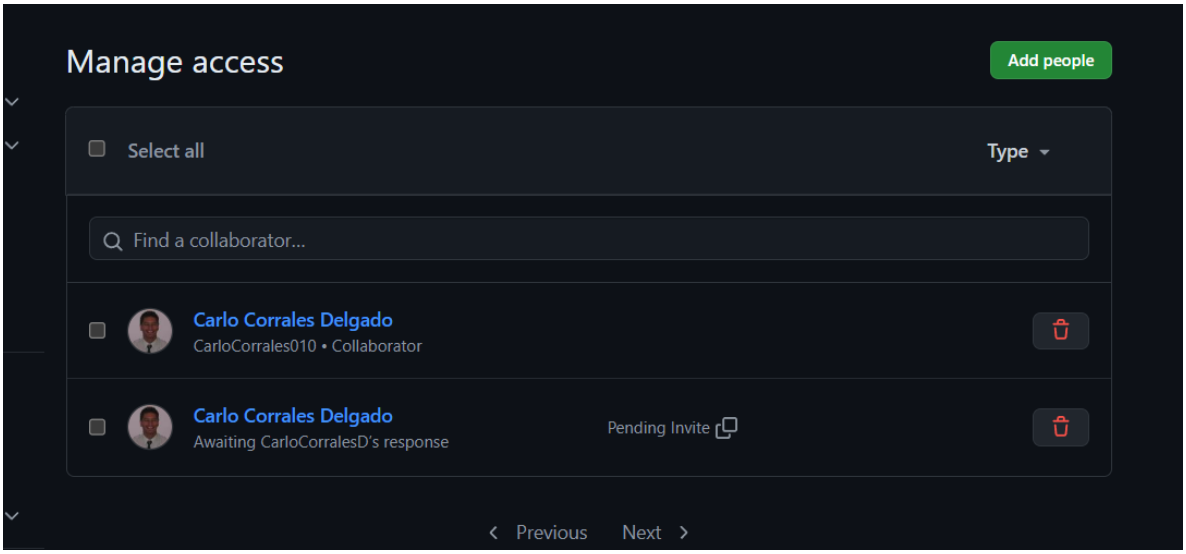

Figura 15: Compartir con el Docente

12. Recomensaciones

- Asegurarse siempre de incluir todos los campos requeridos en los formularios de Django para evitar errores de validación.
- Utilizar valores por defecto de manera cuidadosa y coherente, especialmente en campos booleanos u otros tipos de datos sensibles.

13. Conclusiones

- El uso efectivo de plantillas en Django permite una optimización significativa del frontend de las aplicaciones web. Al separar la lógica de presentación del código Python, se facilita el mantenimiento y la modificación de la interfaz de usuario, lo que resulta en una experiencia más agradable para los usuarios finales.
- Las plantillas en Django proporcionan una gran flexibilidad y capacidad de reutilización de componentes de interfaz. Esto permite construir interfaces dinámicas y personalizadas de manera eficiente, aprovechando la capacidad de Django para gestionar de forma efectiva la lógica de negocio y la presentación de datos en las aplicaciones web.

13.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 2: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		20	

14. Referencias

- https://docs.djangoproject.com/en/5.0/
- https://docs.github.com/es
- https://git-scm.com/doc