

Informe de Programación Web - Django 02

Tema: Vistas en Django

Nota

Estudiantes	Escuela	Asignatura
Jhonatan Benjamin Mamani Céspedes jmamanices@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Práctica	Tema	Duración
02	Vistas en Django	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 4 Junio 2024	Al 8 Junio 2024

1. Tarea

- Seguir las diapositivas de DJango02, dentro de un proyecto git local.
- Deberá hacer un commit por cada paso.
- Responder a las preguntas de las diapositivas (incluya el enunciado de la diapositiva).
- Incrustar la captura de pantalla al inicio de su texto de respuesta) del siguiente comando: git log -graph -pretty=oneline -abbrev-commit -all
- Cada commit debe ser realizado con un mensaje descriptivo del paso de la diapositiva que estuvo siguiendo.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home v 23H2 64 bits
- VIM x64 v9.1.
- Visual Studio Code x64 v1.89.1
- Git v2.45.0.
- Cuenta en GitHub con el correo institucional.
- Python v3.12.3.
- Entorno virtual.
- Django v5.0.6.

3. URL de Repositorio Github

- URL del Repositorio GitHub en donde se realiza el proyecto
- <https://github.com/JBenjamin01/pw2-django>

4. Modificando el modelo Persona

4.1. Cambiando los tipos de campos

- Como inicio de esta segunda parte del proyecto Django, se cuestiona el tipo de campo que se tenía anteriormente en el modelo Persona con charField para la edad, por esto y por la validación respecto al tamaño del campo, se modifica el modelo con tipos de datos más apropiados para sus campos.
- Entonces abrimos el archivo de models.py y modificamos a Persona:

Listing 1: Ingresando a models.py

```
$ vim personas/models.py
```

Listing 2: Modificación del modelo Persona

```
from django.db import models

# Create your models here.
class Persona(models.Model):
    nombres = models.TextField(max_length = 100)
    apellidos = models.TextField(max_length = 100)
    edad = models.IntegerField(max_digits = 3)
```

- Al haber modificado directamente el modelo Persona, necesitamos aplicar las migraciones para efectuar el cambio en la base de datos, estos generan también un nuevo archivo en migrations:

Listing 3: Makemigrations y migrate

```
$ python manage.py makemigrations
$ python manage.py migrate
```

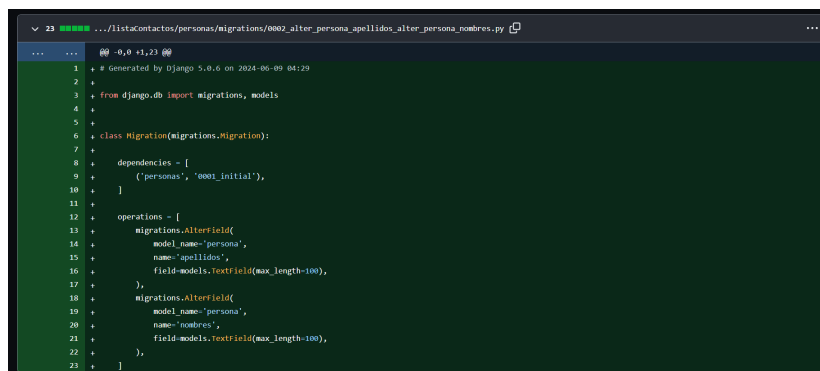


Figura 1: Archivo 0002_alter_persona_apellidos_alter_persona_nombres.py en GitHub

- Siguiendo las diapositivas, se crea un nuevo objeto desde el shell de Python:

Listing 4: Creación de un nuevo objeto desde el shell

```
>>> from personas.models import Persona
>>> Persona.objects.create(nombres="Jorge", apellidos="Gonzales", edad="18")
>>> exit()
```

- Posteriormente, se inicia el servidor :

Listing 5: Activación del servidor

```
$ python manage.py runserver
```

- Verificamos los cambios en el sitio de administración de Django:

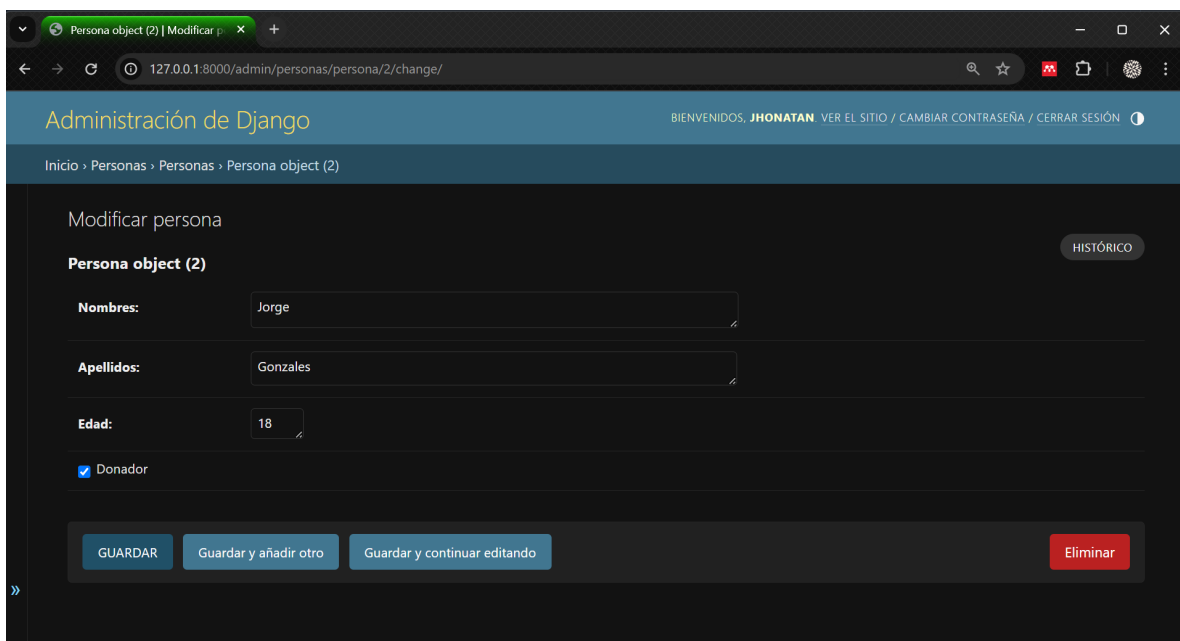


Figura 2: Verificación del objeto desde el Admin Site de Django

4.2. Cambiando el modelo Persona (Agregando un nuevo campo)

- Se quiere agregar un nuevo campo de tipo booleano que defina si la persona que se ingresa es donador o no, para esto se vuelve a modificar la clase Persona:

Listing 6: Modificación del modelo Persona

```
from django.db import models

# Create your models here.
class Persona(models.Model):
    nombres = models.TextField(max_length = 100)
    apellidos = models.TextField(max_length = 100)
    edad = models.IntegerField()
    donador = models.BooleanField()
```

- Al haber modificado de nuevo el modelo Persona, debemos volver a aplicar las migraciones:

Listing 7: Makemigrations

```
$ python manage.py makemigrations
```

- Sin embargo, antes de poder aplicarlas sale un aviso sobre el cambio que realizamos indicándonos que estamos intentando añadir un valor por defecto a las filas existentes, refiriéndose a los objetos que ya habíamos creado anteriormente:

```
You are trying to add a non-nullable field 'donador' to persona without a default  
t; we can't do that (the database needs something to populate existing rows).  
Please select a fix:  
  1) Provide a one-off default now (will be set on all existing rows with a null  
  value for this column)  
  2) Quit, and let me add a default in models.py  
Select an option: 1  
Please enter the default value now, as valid Python  
The datetime and django.utils.timezone modules are available, so you can do e.g.  
    timezone.now  
Type 'exit' to exit this prompt  
>>> True
```

Figura 3: Consola: Default para el campo donador

- Tal como se realiza en la consola de la imagen, solo seleccionamos 1 y como valor por defecto definimos a True para que todos los objetos en las filas de la base de datos se puedan actualizar correctamente:

Listing 8: Migrate y runserver

```
$ Select an option: 1  
$ Please enter the default value now, as valid Python  
$ The datetime and django.utils.timezone modules are available, so you can do  
  e.g. timezone.now  
$ Type 'exit' to exit this prompt  
>>> True
```

- Ejecutamos el migrate e iniciamos el servidor para verificarlo:

Listing 9: Makemigrations y migrate

```
$ python manage.py migrate  
$ python manage.py runserver
```

- Observamos el primer objeto y vemos que efectivamente, ahora tiene el campo donador marcado como True pese a no haber sido creado con este valor, nuestro procedimiento fue funcional:

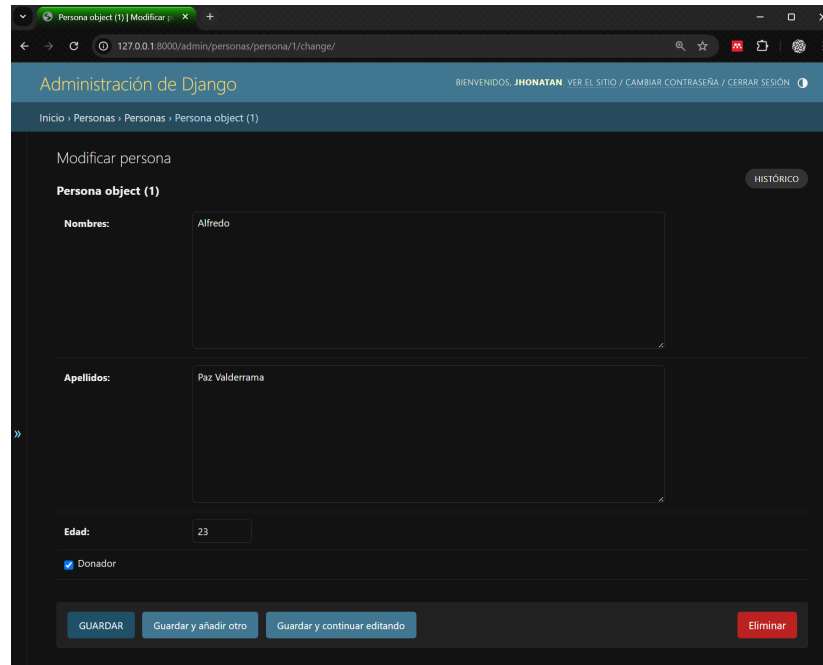


Figura 4: Objeto Persona 1

- Adicionalmente decidí implementar una forma de evitar el error con el uso de 'max_digits = 3' para validar las entradas de las edades, en este caso, la edad puede variar entre 0 y 120:

Listing 10: Validación de edad

```
from django.db import models
from django.core.exceptions import ValidationError

def validate_age(value):
    if value < 0 or value > 120:
        raise ValidationError('Edad debe estar entre 0 y 120.')
```

Create your models here.

```
class Persona(models.Model):
    nombres = models.TextField(max_length = 100)
    apellidos = models.TextField(max_length = 100)
    edad = models.IntegerField(validators=[validate_age])
    donador = models.BooleanField()
```

5. Personalizar la página de inicio

5.1. Creación de la app

- Para modificar esta página según las diapositivas se requiere crear una clase o función en la vista de una app, es así que se decide crear una app de prueba llamada 'inicio':

Listing 11: Creación de la app inicio

```
$ python manage.py startapp inicio
```

- Tras crear esta nueva app, al igual que se había realizado anteriormente con la app 'personas', esta también se debe agregar a las apps instaladas de la configuración del proyecto, entonces nos dirigimos a settings.py:

Listing 12: Ingresando a settings.py

```
$ vim listaContacto/settings.py
```

- Bajando hasta la parte de INSTALLED_APPS, agregamos la aplicación que acabamos de crear al final de la lista de la siguiente forma:

Listing 13: Aplicaciones del proyecto

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'personas',
    'inicio',
]
```

5.2. Creación de la vista

- Por otro lado, necesitamos crear la vista para devolver la respuesta que queremos darle a la página de inicio, por esto, modificamos el archivo de views de la aplicación recientemente creada:

Listing 14: Ingresando a views.py

```
$ vim inicio/views.py
```

- Modificamos el archivo agregando una nueva función para tener una vista sencilla que devuelve un saludo en HTML, y es un ejemplo clásico de una primera vista en un proyecto Django para asegurarse de que todo está funcionando correctamente:

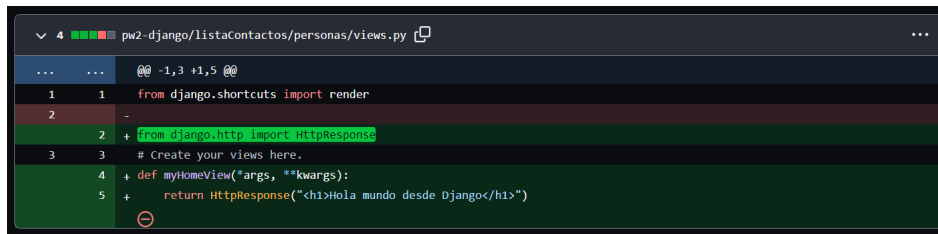
Listing 15: Respuesta HttpResponse de la vista myHomeView

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

# Create your views here.
def myHomeView(*args, **kwargs):
    return HttpResponse("<h1>Hola mundo desde Django</h1>")
```

- El propósito de la vista myHomeView que se creó es devolver una respuesta HTTP con un mensaje "Hola mundo desde Django" incrustado en etiquetas h1. Este es un ejemplo de una vista básica que se sigue en las diapositivas, se utilizó principalmente para comprobar que el marco de trabajo está configurado correctamente y que las vistas funcionan como se espera en la página de inicio.

- El commit realizado fué el siguiente:



```

pw2-django/listaContactos/personas/views.py
... -1,3 +1,5 @@
1 1 from django.shortcuts import render
2 -
2 + from django.http import HttpResponseRedirect
3 3 # Create your views here.
4 + def myHomeView(*args, **kwargs):
5 +     return HttpResponseRedirect("<h1>Hola mundo desde Django</h1>")

```

Figura 5: Commit de los cambios

5.3. Registro de la URL

- Una vez realizada la creación de la vista, debemos registrarlo en el archivo urls.py, de esta manera podremos acceder a la página desde el servidor con el url que definamos:

Listing 16: Ingresando a urls.py

```
$ vim inicio/urls.py
```

- Con esto vamos a asegurarnos de que la vista sea accesible a través de una URL en el proyecto Django, aquí se configura la URL sin especificar ingresar nada entre las comillas para que se aplique a la página de inicio:

Listing 17: Registro de la URL para la vista myHomeView

```

from django.contrib import admin
from django.urls import path
from inicio.views import myHomeView

urlpatterns = [
    path('', myHomeView, name='Pagina de Inicio'),
    path('admin/', admin.site.urls),
]

```

- Algo interesante que se encontró aquí fué la url de admin, en donde la parte de admin.site.urls se refiere al conjunto de URLs que Django ha predefinido para el sitio de administración. Estas URLs incluyen todas las rutas necesarias para manejar la interfaz de administración, como el inicio de sesión, la página principal del admin, las páginas para agregar, editar, eliminar modelos, etc.

5.4. Verificación de la URL

- Volviendo a lo que veníamos desarrollando, solo nos queda revisar que la URL de nuestra vista funcione correctamente, para esto volvemos a iniciar el servidor:

Listing 18: Inicio del servidor

```
$ python manage.py runserver
```

- Entonces, con el servidor activo, nos vamos a dirigir hacia la página de inicio principal de Django en `http://127.0.0.1:8000/`, aquí encontraremos lo siguiente:

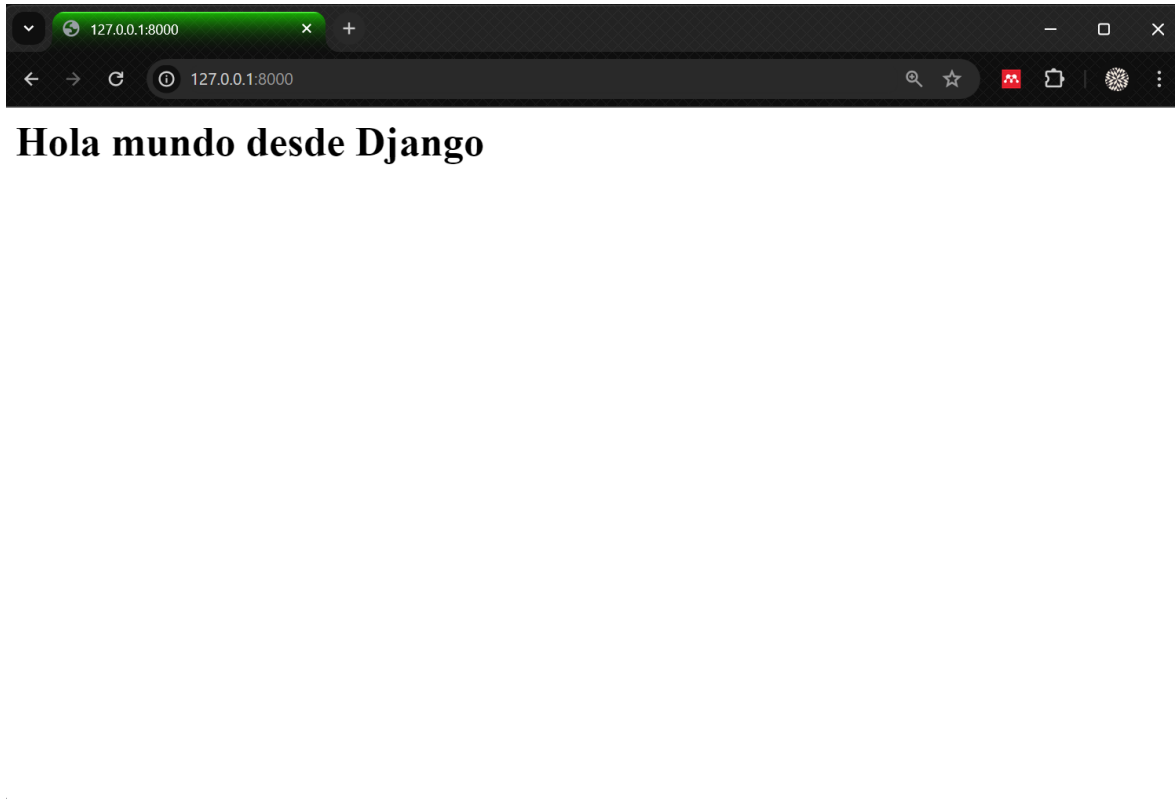


Figura 6: Página de Inicio con la nueva vista

- En donde comprobamos que funciona correctamente.

6. Peticiones y ruteos de URL

6.1. URL

- Definiendo un poco acerca del funcionamiento y relación entre las vistas y rutas, las vistas en Django son las que manejan peticiones HTTP y pueden responder de acuerdo con el método de la solicitud (GET, POST, etc.), en cuanto a los ruteos, el archivo `urls.py` mapea las URLs a las vistas correspondientes, permitiendo una navegación estructurada y lógica dentro de la aplicación según nosotros vayamos viendo convenientemente.
- Entonces, si decidimos crear una nueva URL para la misma vista no habría ningún problema mientras exista la vista y por lógica no tenga la misma ruta.
- Procedemos a abrir de nuevo el archivo de `urls.py` para agregar la nueva ruta:

Listing 19: Ingresando a `urls.py`

```
$ vim inicio/urls.py
```

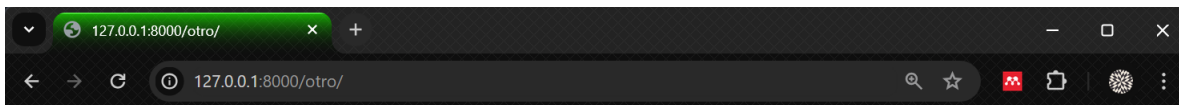

- Definimos la URL adicional con una nueva cadena que mapee una ruta con el siguiente complemento al final: otro/

Listing 20: Registro de la nueva URL para la vista myHomeView

```
from django.contrib import admin
from django.urls import path
from inicio.views import myHomeView

urlpatterns = [
    path('', myHomeView, name='Pagina de Inicio'),
    path('otro/', myHomeView, name='Pagina de Inicio'),
    path('admin/', admin.site.urls),
]
```

- Entonces, volviendo al servidor activo, si nos dirigimos a la nueva ruta `http://127.0.0.1:8000/otro/` se mostrará la misma vista de myHomeView:



Hola mundo desde Django

Figura 7: Visualización en la página enrutada /otro

6.2. Nuevas vistas

- Con lo anteriormente realizado, es posible desarrollar más vistas con diferentes lógicas de respuesta, siguiendo las diapositivas, se crea una nueva para la URL con la ruta de /otro/
- Nos dirigimos entonces a views.py de nuevo:

Listing 21: Ingresando a views.py

```
$ vim inicio/views.py
```

- Creamos la nueva vista anotherView:

Listing 22: Creación de la nueva vista anotherView

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

# Create your views here.
def myHomeView(*args, **kwargs):
    return HttpResponse("<h1>Hola mundo desde Django</h1>")
def anotherView(request):
    return HttpResponse("<h1>Solo otra pagina mas</h1>")
```

- Un dato a considerar aquí es la diferencia que manejan los argumentos que reciben las dos vistas, por ejemplo, en `anotherView` la vista recibe un solo argumento, `request`, que es una instancia de `HttpRequest`. Según la práctica con Django parece ser el método más común y estándar para definir vistas. También encontré que el objeto `request` contiene toda la información sobre la solicitud HTTP actual, incluyendo datos GET y POST, cookies, y meta información.
- Entonces, una vez realizada esa aclaración pasamos a la ruta, nos dirigimos entonces a `urls.py`:

Listing 23: Ingresando a `urls.py`

```
$ vim inicio/urls.py
```

- Y en este archivo, asignamos la nueva vista a la segunda ruta que habíamos creado antes:

Listing 24: Asignación de la vista `anotherView` a la ruta `/otro/`

```
from django.contrib import admin
from django.urls import path
from inicio.views import myHomeView

urlpatterns = [
    path('', myHomeView, name='Pagina de Inicio'),
    path('otro/', anotherView),
    path('admin/', admin.site.urls),
]
```

- Y la visualización en la página de la ruta definida ahora es la siguiente:

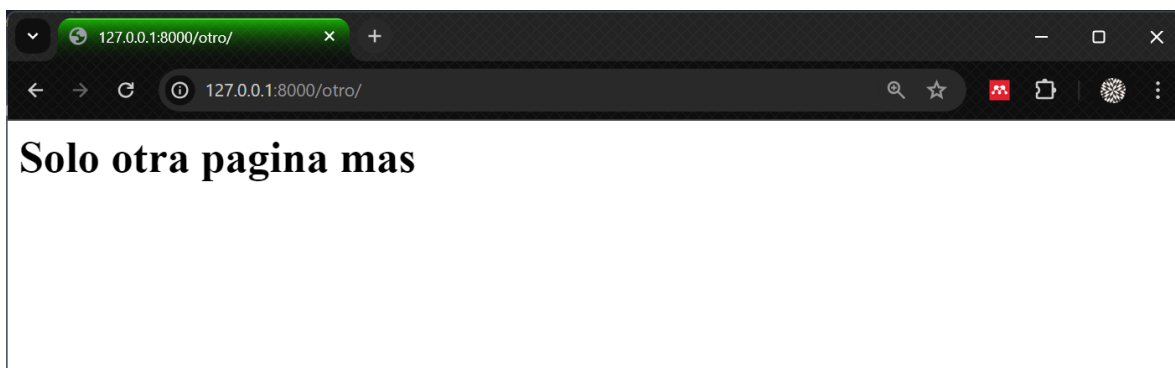


Figura 8: Nueva vista `anotherView` en la ruta `/otro`

6.3. Argumento de las funciones de vistas

- Como se había visto antes, las vistas reciben argumentos diferentes, en este caso, las diapositivas buscan imprimir los argumentos además del usuario que hace la solicitud.
- Nos dirigimos a `views.py`:

Listing 25: Ingresando a `views.py`

```
$ vim inicio/views.py
```

- Agregamos los print para imprimir estos valores:

Listing 26: Impresión de argumentos dentro de la función

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

# Create your views here.
def myHomeView(request, *args, **kwargs):
    print(args, kwargs)
    print(request.user)
    return HttpResponse("<h1>Hola mundo desde Django</h1>")
def anotherView(request):
    return HttpResponse("<h1>Solo otra pagina mas</h1>")
```

- Entonces, en nuestro servidor activo volveremos a ingresar a la página de inicio. Y en la consola se nos mostrará lo siguiente:

```
re\Programación Web 2\DJango\pw2-django\listaContactos> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 18, 2024 - 09:42:45
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.

() {}
jhonatan
[18/Jun/2024 09:42:49] "GET / HTTP/1.1" 200 32
```

Figura 9: Argumentos y usuario impresos en consola

- Como se observó, además de devolver la respuesta HTTP que muestra el mensaje "Hola mundo desde Django", dentro de la misma función, se imprimieron los valores de args y kwargs, así como el usuario que hizo la solicitud con request.user.
- Finalmente, si ingresamos desde el modo incógnito, la impresión del valor de usuario que realiza la solicitud cambia a anónimo:

```
() {}
jhonatan
[18/Jun/2024 09:42:49] "GET / HTTP/1.1" 200 32
() {}
AnonymousUser
[18/Jun/2024 09:53:15] "GET / HTTP/1.1" 200 32
```

Figura 10: Argumentos y usuario anónimo

7. Plantillas (Templates) de Django

7.1. Templates

- Los templates son la forma que tiene Django para generar código HTML, en las diapositivas se asignan algunas funciones simplificadas de esta forma:

- URL es la ruta
- Views es la lógica
- Template es el HTML

- Para diseñar nuestros primeros templates creamos un nuevo directorio:

Listing 27: Directorio templates

```
$ mkdir templates
```

- Creamos una nueva plantilla de HTML:

Listing 28: Creando el archivo home.html

```
$ vim templates/home.html
```

- Y desarrollamos la estructura de nuestro HTML básico:

Listing 29: Contenido de home.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Template Base</title>
  </head>
  <body>
    <h1>Hola Mundo desde Django!!</h1>
    <h2>Con Templates</h2>
  </body>
</html>
```

- Configuramos los TEMPLATES del archivo settings.py del proyecto

Listing 30: Ingresando a settings.py

```
$ vim listaContactos/settings.py
```

- Solo se modifica la lista de DIRS y también importamos el módulo os para que funcione:

Listing 31: Configurando templates

```
import os
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        .....
    }
]
```

- Por último nos dirigimos a views.py

Listing 32: Ingresando a views.py

```
$ vim listaContactos/views.py
```

- Y modificamos la respuesta de la función llamando a la plantilla HTML que acabamos de crear en templates:

Listing 33: Respuesta render del template home.html

```
def myHomeView(request, *args, **kwargs):
    print(args, kwargs)
    print(request.user)
    return render(request, "home.html", {}),
```

- La respuesta en la página de salida es la siguiente:

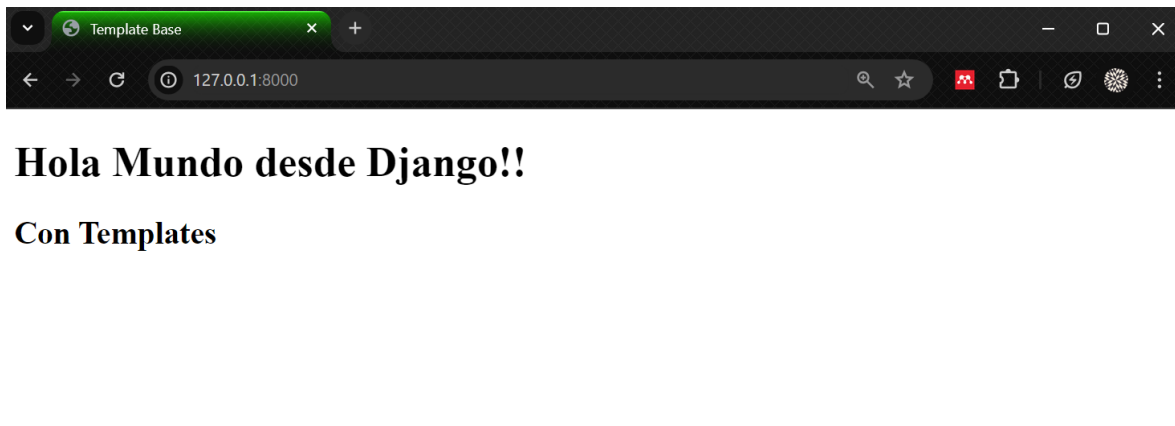


Figura 11: Página hecha con el template home.html

- Como se ha observado, los templates son archivos HTML que pueden contener una combinación de HTML estático y sintaxis del template de Django. Estos archivos suelen almacenarse en un directorio llamado templates dentro de las aplicaciones Django.

7.2. Django template language (DTL): Variables

- Una variable se considera como un valor según el contexto. Las variables se encuentran encerradas entre llaves dobles variable, request también puede ser usado como variable, ingresamos a la plantilla:

Listing 34: Ingresando a home.html

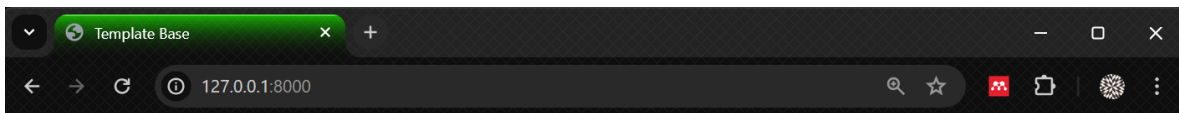
```
$ vim templates/home.html
```

- Las variables usadas son `{{ request.user }}` que muestra el usuario en una pag web Django y `{{ request.user.is_authenticated }}` que nos indica si está logeado.:

Listing 35: Agregación de variables de Django

```
<!DOCTYPE html>
<html>
  <head>
    <title>Template Base</title>
  </head>
  <body>
    <h1>Hola Mundo desde Django!!</h1>
    <h2>Con Templates</h2>
    {{ request.user }}
    <br>
    {{ request.user.is_authenticated }}
  </body>
</html>
```

- La forma en la que se muestran en la página es la siguiente:



Hola Mundo desde Django!!

Con Templates

jhonatan
True

Figura 12: Variables de request en la página principal

7.3. Django template language (DTL): Tags

- Los tags están encerrados entre llaves con signos de porcentaje `{% tag %}`
- Pueden actuar como una estructura de control, similar a un 'if' o un bucle 'for', puede funcionar como un bloque estructural, como las llaves en lenguajes como Java, y puede usarse para extraer contenido de una base de datos o proporcionar acceso a otros tags.
- Para probar el uso de los tags vamos a crear un nuevo html que contendrá la plantilla base:

Listing 36: Creando la plantilla base.html

```
$ vim templates/base.html
```

Listing 37: Incrustando tags block

```
<!DOCTYPE html>
<html>
  <head>
    <title>Codigo para estudiantes</title>
  </head>
  <body>
    <h1>ESTE ES UN TEXTO DE BASE</h1>
    {% block content %}
      Reemplazame.....
    {% endblock %}
  </body>
</html>
```

7.4. Tag Extends

- Contando con la plantilla base, se pueden definir nuevos y antiguos templates reusando algo de código, esto con el uso de extends para traer bloques de código, entramos a base.html y editamos el HTML que se usa en la vista (home.html) para que obtenga la plantilla de base.html:

Listing 38: Ingresando a home.html

```
$ vim templates/home.html
```

Listing 39: Incrustando tag extends a la plantilla de base

```
{% extends "base.html" %}
{% block content %}
<!DOCTYPE html>
<h2>Hola Mundo desde Django!!</h2>
<h3>Con Templates</h3>
{{ request.user }}
<br>
{{ request.user.is_authenticated }}
{% endblock %}
```

- Si se inicia el servidor e ingresamos a la página principal nos encontramos con el código del HTML heredado de la plantilla en base.html y la información agregada de home.html:

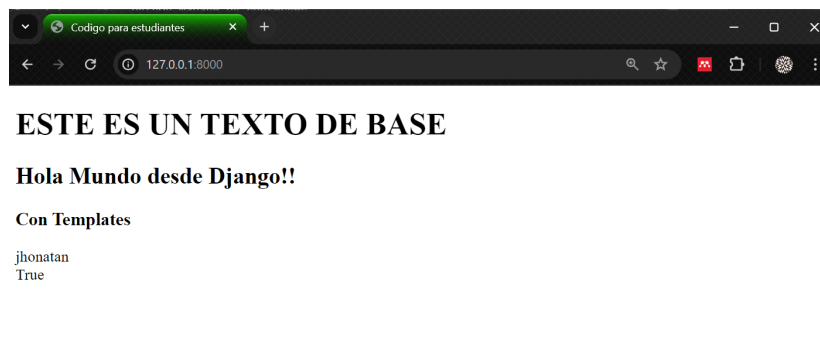


Figura 13: Tag extends en la página de inicio

7.5. Tag Include

- Este tag viene a ser la manera de incrustar código de otros templates a uno nuevo.
- Para probarlo, se crea un nuevo html llamado nav, el cual se incrustará en base.html:

Listing 40: Creando la plantilla nav.html

```
$ vim templates/nav.html
```

- Definimos el contenido de nav, ya que hace referencia a un navbar, creamos una lista:

Listing 41: Creando lista del navbar

```
<nav>
  <ul>
    <li>Home</li>
    <li>Primero</li>
    <li>Segundo</li>
    <li>Tercero</li>
  </ul>
</nav>
```

- En base.html solo agregamos un tag include al inicio de body:

Listing 42: Ingresando a base.html

```
$ vim templates/base.html
```

Listing 43: Incrustando el Tag include

```
<body>
{% include 'nav.html' %}
<h1>ESTE ES UN TEXTO DE BASE</h1>
{% block content %}
  Reemplazame.....
{% endblock %}
</body>
```

- El uso de estos tags es muy útil cuando se tienen partes de HTML que se repiten en varias páginas del sitio web. En lugar de copiar y pegar el mismo código en múltiples archivos HTML, sería posible crear un template separado para esa parte y luego incluirlo en las plantillas donde lo necesites con una sola línea.

- Finalmente, volviendo de nuevo al contenido de las diapositivas, al iniciar el servidor, en la página principal se nos muestra la lista que pusimos en el HTML de nav.html, ahora incrustado en base.html del cual extiende home.html:

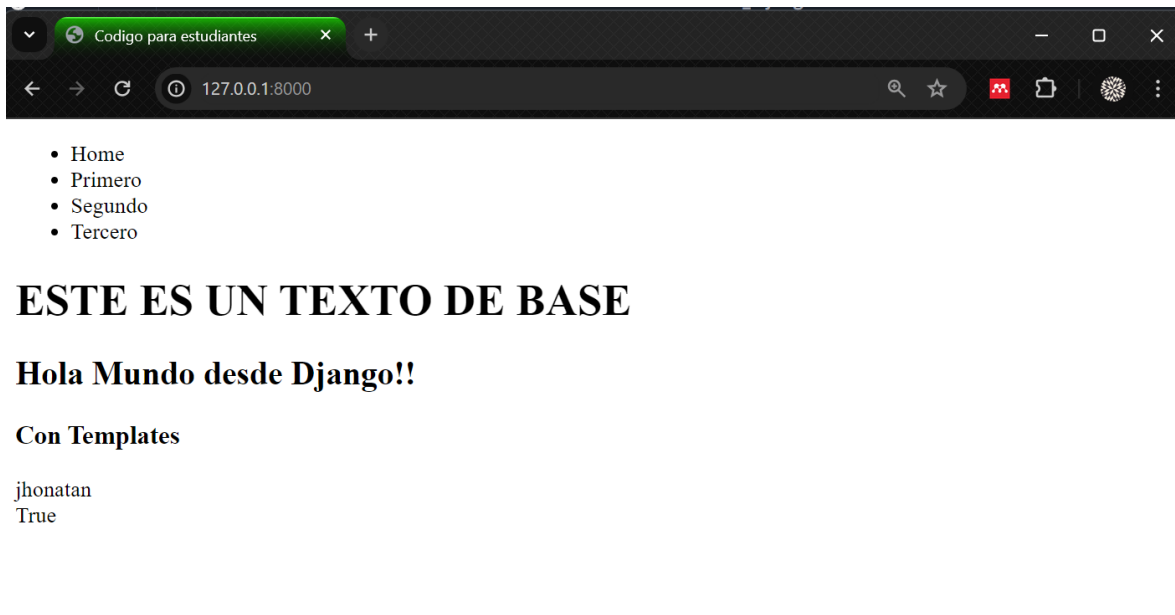


Figura 14: Tag include en la página principal

8. Captura de pantalla de commits

- Como último paso de la actividad se ejecuta este comando para obtener el historial de los commits realizados en la actividad y le se le toma una captura, para quitar los commits anteriores, se recortaron los realizados únicamente para la diapositiva de Django 02:

Listing 44: Historial de commits

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

```
(python_env) PS C:\Users\jhona\OneDrive\Documents\University\Universidad Nacional de San Agustín\2nd Year\Primer Semestre\Programación Web 2\Django\pw2-django> git log --graph --pretty=oneline --abbrev-commit --all
* 1c4c78c (HEAD -> main, origin/main, origin/HEAD) Uso del Tag Include para agregar un navbar en la plantilla base.html
* b88a5a3 Uso del Tag Extends para usar la plantilla base en home.html
* b01703e Uso del Tag Block en una plantilla nueva base.html
* 641ed56 Uso de las variables request.user y su autenticación
* a65ac88 Uso del primer Template home.html y la configuración básica
* 91a6500 Impresión de los argumentos ingresados en las funciones de las vistas
* 2466f9f Nueva vista anotherView asignada a la ruta con la cadena /otro/
* 2613657 Prueba de un segundo URL para la vista myHomeView
* ae0506e Registro de la nueva URL asignada con la vista para la nueva página de inicio
* 594256b Solución de la validación con una función que verifique la entrada de edad
* 57e413c Mejora del modelo de Persona con la validación de edad
* 7243d11 Corrección de las vistas en personas a las vistas en inicio
* 685b482 Creación de la app inicio
* 3259d9f Configuración en settings para permitir el acceso a la vista myHomeView.
* f937923 Creación de la vista para la página de inicio en Django.
* f69c8d5 Nuevo campo de entrada donador, y actualización del valor True por defecto desde shell.
* 09a8b84 Creación de una nueva entrada para Persona desde shell.
* ee0d757 Modificación del modelo Persona, valor máximo en la entrada de datos.
```

Figura 15: Captura del historial de commits

9. Estructura del trabajo

- El contenido que se entrega en esta práctica es el siguiente:


```
pw2-django/  
|--- Informe01  
| |--- img  
| | |--- Admin_login.png  
| | |--- Admin_site.png  
| | |--- Admin_site2.png  
| | |--- Captura.png  
| | |--- Commit1.png  
| | |--- Commit2.png  
| | |--- Commit3.png  
| | |--- Commit4.png  
| | |--- Django_site.png  
| | |--- logo_abet.png  
| | |--- logo_episunsa.png  
| | |--- logo_unsa.jpg  
| | |--- Persona1.png  
| |--- Informe_Django01.pdf  
| |--- Informe_Django01.tex  
|--- Informe02  
| |--- img  
| | |--- Anonymous.png  
| | |--- Argumentos.png  
| | |--- Captura.png  
| | |--- Commit1.png  
| | |--- DonadorDefault.png  
| | |--- Inicio.png  
| | |--- logo_abet.png  
| | |--- logo_episunsa.png  
| | |--- logo_unsa.jpg  
| | |--- Otro.png  
| | |--- Persona1.png  
| | |--- Persona2.png  
| | |--- Pregunta1.png  
| | |--- Pregunta2.png  
| | |--- Tag_extends.png  
| | |--- Tag_include.png  
| | |--- Template.png  
| | |--- URL2.png  
| | |--- Variables.png  
| | |--- Vista.png  
| |--- Informe_Django02.pdf  
| |--- Informe_Django02.tex  
|--- pw2-django  
| |--- listaContactos  
| | |--- listaContactos  
| | | |--- __init__.py  
| | | |--- asgi.py  
| | | |--- settings.py  
| | | |--- urls.py  
| | | |--- wsgi.py  
| | |--- inicio  
| | |--- migrations
```

```
| | | | |--- __init__.py
| | | | |--- __init__.py
| | | | |--- admin.py
| | | | |--- apps.py
| | | | |--- models.py
| | | | |--- tests.py
| | | | |--- views.py
| | | |--- personas
| | | |--- migrations
| | | | |--- 001_initial.py
| | | | |--- 0002_alter_persona_apellidos_alter_persona_nombres.py
| | | | |--- 0003_persona_donador.py
| | | | |--- 0004_alter_persona_edad.py
| | | | |--- 0005_alter_persona_edad.py
| | | | |--- __init__.py
| | | | |--- __init__.py
| | | | |--- admin.py
| | | | |--- apps.py
| | | | |--- models.py
| | | | |--- tests.py
| | | | |--- views.py
| | | |--- templates
| | | | |--- base.html
| | | | |--- home.html
| | | | |--- nav.html
| | | |--- db.sqlite3
| | | |--- manage.py
| | | |--- Comandos.txt
| | | |--- .gitignore
| | | |--- README.md
```

10. Preguntas:

10.1. ¿Qué pasa si añadimos un nuevo campo? los anteriores registros no lo tendrían, entonces ¿Con qué valor se actualizarán?

Viendo los cambios hechos de manera automática



```

((prueba) apaz src $ python manage.py shell
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 03:13:28)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from personas.models import Persona
>>> Persona.objects.create(nombre="Jorge", apellidos="Gonzales", edad=18)
<Persona: Persona object (3)>

```

- ¿Qué pasa si añadimos un nuevo campo? los anteriores registros no lo tendrían, entonces ¿Con qué valor se actualizarán?

Figura 16: Primera Diapositiva de Preguntas

- Cuando añadí un nuevo campo a un modelo en Django y ejecuté makemigrations, Django creó un archivo de migración para reflejar este cambio en la base de datos; como el campo no permitía valores nulos (en mi caso, un BooleanField), me pidió un valor por defecto para asignar a los registros existentes.
- Podía especificar este valor en el modelo directamente (por ejemplo, default=True) o también podía proporcionarlo durante el makemigrations.
- Al ejecutar migrate, Django actualizó todos los registros existentes, asignando el valor por defecto al nuevo campo, asegurando que cada registro tuviera un valor válido y manteniendo la consistencia de la base de datos. Aquí se dio el caso:

```

You are trying to add a non-nullable field 'donador' to persona without a default
t; we can't do that (the database needs something to populate existing rows).
Please select a fix:
  1) Provide a one-off default now (will be set on all existing rows with a null
value for this column)
  2) Quit, and let me add a default in models.py
Select an option: 1
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g.
    timezone.now
Type 'exit' to exit this prompt
>>> True

```

Figura 17: Consola: Default para el campo donador

10.2. ¿Cómo crearía un campo que sea obligatorio? ¿Cuáles de estos elementos afectarían a la base de datos? ¿Cuáles no?

Actualizando los campos antiguos

- Dejar a todos los campos antiguos con el mismo valor, es peligroso!
 - `null = True`, es legal que el campo se quede sin valor
 - `default=True`, el valor por defecto será `True`
 - `blank=True` significa que se puede dejar en blanco (no es requerido)
- ¿Cómo crearía un campo que sea obligatorio?
 - En el formulario los campos obligatorios aparecen en negrita (`blank = False`)
- ¿Cuáles de estos elementos afectarían a la base de datos? ¿Cuáles no?

<https://docs.djangoproject.com/en/3.2/ref/models/fields/>

Figura 18: Segunda Diapositiva de Preguntas

- Para crear un campo obligatorio en un modelo de Django, configuro el campo con `null=False` y `blank=False`.
- El parámetro `null=False` afecta a la base de datos porque asegura que la columna correspondiente no pueda contener valores nulos, mientras que `blank=False` es relevante para la validación de formularios en Django y no afecta la estructura de la base de datos.
- Otros parámetros que impactan directamente la base de datos incluyen `default`, `unique=True`, y `primary_key=True`, ya que estos establecen valores predeterminados, restricciones de unicidad y claves primarias, respectivamente.

Listing 45: Implementación de la lógica con `null` y `blank`

```
from django.db import models

class Persona(models.Model):
    nombre = models.CharField(max_length=100)
    edad = models.IntegerField()
    email = models.EmailField(null=False, blank=False) # Campo obligatorio
```

- En contraste, aquí, los parámetros como `blank=False`, `verbose_name`, y `help_text` no afectan la estructura de la base de datos, sino que mejoran la validación y documentación en los formularios y el panel de administración de Django.
- Por ejemplo, al definir un campo `email` en el modelo `Persona` con `null=False`, `blank=False`, y `unique=True`, garantizo que cada registro tenga un valor de email no nulo, no vacío y único, lo cual impacta tanto la base de datos como la lógica de validación en Django.

- Esta sería la implementación:

Listing 46: Implementación de la lógica de validación con unique

```
from django.db import models

class Persona(models.Model):
    nombre = models.CharField(max_length=100)
    edad = models.IntegerField()
    email = models.EmailField(null=False, blank=False, unique=True,
                              help_text="Por favor, ingrese un email valido") # Campo obligatorio y
    unico
```

11. Rúbricas

11.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

11.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple 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 los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: 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 3: 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	3	
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 están 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		19	

12. Referencias

- <https://www.w3schools.com/django/>
- <https://docs.djangoproject.com/es/5.0/intro/tutorial01/>
- <https://github.com/mdn/django-locallibrary-tutorial/blob/main/.gitignore>
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/development_environment
- https://tutorial.djangogirls.org/es/django_views/
- https://www.w3schools.com/django/django_template_tags.php