



Práctica 6.

Modelos de datos y panel de Django



Objetivo

Crear un modelo de datos para registrar casas de resguardo y mostrar su información desde Django.

PASOS

1

Comprende qué es un modelo

Un modelo describe la información que guardará la aplicación. Empezaremos con casas de resguardo; después los otros equipos podrán adaptar la estructura.



2

Escribe el modelo en casas/models.py

Utiliza nombres claros. Los datos reales de las casas deberán ser revisados con las personas responsables de la comunidad.

casas > models.py

```

from django.db import models

class Casa(models.Model):
    nombre = models.CharField(max_length=120)
    descripcion = models.TextField()
    ubicacion = models.CharField(max_length=180)

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

3

Genera y aplica las migraciones

RESULTADO ESPERADO

Django confirma la creación y aplicación de la migración sin errores.

TERMINAL - PowerShell

```

python manage.py makemigrations casas
python manage.py migrate
  
```



Práctica 6.

Panel de administración

4

Registra el modelo en el administrador

Abre `casas/admin.py`. Al registrar Casa, Django permite agregar y editar registros desde su panel.

`casas > admin.py`

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

```
admin.site.register(Casa)
```

5

Crea una cuenta de administración

Ejecuta el comando y sigue las instrucciones de la terminal. La contraseña no debe compartirse ni subirse a GitHub.

TERMINAL · PowerShell

```
python manage.py createsuperuser
python manage.py runserver
```

6

Agrega un registro de prueba

Entra a `http://127.0.0.1:8000/admin/`, inicia sesión, abre Casas y agrega una casa ficticia de prueba. No inventes información real de la comunidad.

Administración de Django

IMPORTANTE

El administrador es para capturar contenido; no es la página pública que verán los visitantes.

[Guardar]



Práctica 6.

Mostrar información real

7

Consulta los registros desde una vista

En `casas/views.py`, consulta los objetos guardados y envíalos a la plantilla. La ruta `casas/` debe apuntar a esta vista.

`casas > views.py`

```
from django.shortcuts import render
from .models import Casa

def inicio(request):
    casas = Casa.objects.all()
    return render(request, "casas/inicio.html",
                  {"casas": casas})
```

8

Presenta los datos en la plantilla

En `templates/casas/inicio.html` usa un ciclo `for` para mostrar las casas. No escribas información ficticia como si fuera verificada.

`templates > casas > inicio.html`

```
{% extends "base.html" %}
{% block content %}
<h1>Casas de resguardo</h1>
{% for casa in casas %}
  <h2>{{ casa.nombre }}</h2>
  <p>{{ casa.descripcion }}</p>
{% empty %}
  <p>Aún no hay registros publicados.</p>
{% endfor %}
{% endblock %}
```

AUTOEVALUACIÓN

☐ Creé el modelo Casa. ☐ Apliqué migraciones. ☐ Registré el modelo en admin. ☐ Mostré registros con una plantilla.

9

Publica tus cambios en tu rama de `GitHub`.



Práctica 7.

Formularios y validación de datos



Objetivo

Construir un formulario Django para registrar datos de prueba con validaciones y protección CSRF.

PASOS

1

¿Para qué sirve un formulario?

Un formulario recibe información del usuario y Django comprueba si es válida antes de guardarla. El equipo Casas lo utilizará como ejemplo.

FLUJO DE TRABAJO

Navegador → Formulario → Validación → Modelo Casa → Base de datos.

2

Crea casas/forms.py

ModelForm genera campos a partir del modelo Casa creado en la práctica 6. Solo se incluyen los campos necesarios.

casas > forms.py

```
from django import forms
from .models import Casa

class CasaForm(forms.ModelForm):
    class Meta:
        model = Casa
        fields = ["nombre", "descripcion", "ubicacion"]
```

3

Agrega una regla de validación

CONSEJO

Usa registros ficticios durante las pruebas. Los datos comunitarios deben validarse antes de publicarse.

```
def clean_nombre(self):
    nombre = self.cleaned_data["nombre"].strip()
    if len(nombre) < 3:
        raise forms.ValidationError("Mínimo 3 caracteres.")
    return nombre
```



Práctica 7.

Procesar y proteger el formulario

4

Programa la vista para GET y POST

GET muestra el formulario vacío. POST procesa lo enviado. Si los datos son correctos, se guardan y se redirige al listado.

casas > views.py (añadir a las vistas existentes)

```
from django.shortcuts import render, redirect
from .forms import CasaForm

def nueva_casa(request):
    if request.method == "POST":
        form = CasaForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect("inicio_casas")
    else:
        form = CasaForm()
    return render(request, "casas/formulario.html",
                  {"form": form})
```

5

Agrega la ruta del formulario

Conserva la ruta del listado y agrega la nueva. La ruta nueva debe declararse en casas/urls.py.

casas > urls.py

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

SEGURIDAD

No elimines {% csrf_token %} del formulario: Django lo usa para proteger las solicitudes POST. En una aplicación real, restringe esta vista a personal autorizado.



Práctica 7.

Plantilla, pruebas y evidencias

6

Diseña la plantilla HTML

Crea templates/casas/formulario.html. Hereda el diseño general y presenta los campos y mensajes de error de Django.

```
templates > casas > formulario.html
```

```
{% extends "base.html" %}
{% block content %}
<h1>Registrar casa de resguardo</h1>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Guardar</button>
</form>
{% endblock %}
```

7

Comprueba la página en el navegador

Ejecuta el servidor y abre /casas/nueva/. Intenta guardar un nombre de dos letras: debe aparecer el mensaje de validación.

```
TERMINAL
```

```
python manage.py check
python manage.py runserver
```

```
127.0.0.1:8000/casas/nueva/
```

AUTOEVALUACIÓN

☐ Creé un ModelForm. ☐ Validé el nombre. ☐ Usé csrf_token. ☐ Probé un registro válido y otro inválido.

Nombre:

Ubicación:



Práctica 8.

Imágenes, audio y contenido multimedia



OBJETIVO

Integrar imágenes y audios autorizados en Django con rutas correctas, accesibilidad y pruebas.

PASOS

1

Distingue static de media

Static contiene archivos propios del sitio: CSS, imágenes y audios. Media se utiliza para archivos que suben los usuarios. Aquí usaremos static.

ORGANIZACIÓN

static/imagenes/lengua/ y static/audio/lengua/

2

Organiza los recursos autorizados

Coloca una fotografía en la carpeta imagenes/lengua y un audio MP3 en audio/lengua. Usa nombres cortos y sin espacios.

EXPLORADOR DE VS CODE

```
static/  
  imagenes/  
    lengua/  
      comunidad.jpg  
  audio/  
    lengua/  
      saludo.mp3
```

3

Comprueba la configuración

En config/settings.py debe existir `STATIC_URL = "static/"`.

CUIDADO

Usa únicamente imágenes y audios con autorización y validación comunitaria.



Práctica 8.

Insertar imágenes y audio en HTML

4

Carga la etiqueta static

En templates/lengua/inicio.html agrega load static después de extends. Inserta la imagen con una descripción alt.

templates > lengua > inicio.html

```
{% extends "base.html" %}
{% load static %}
{% block content %}
<h1>Lengua ñötho</h1>

{% endblock %}
```

5

Agrega un reproductor de audio

Dentro del bloque content, antes de endblock, coloca el reproductor con controles y texto de apoyo.

Fragmento HTML para agregar a la plantilla

```
<figure>
  <figcaption>Escucha la pronunciación:</figcaption>
  <audio controls>
    <source src="{% static 'audio/lengua/saludo.mp3' %}"
      type="audio/mpeg">
    Tu navegador no reproduce audio.
  </audio>
</figure>
```

ACCESIBILIDAD

Añade transcripción verificada del audio y una descripción alt útil para la imagen.



Práctica 8.

Estilos, comprobación y entrega

6

Ajusta imágenes y audio con CSS

Agrega reglas responsivas para evitar que los recursos sobrepasen el ancho de la pantalla.

static > css > estilos.css

```
img { max-width: 100%; height: auto; }  
audio { width: 100%; max-width: 420px; }
```

7

Comprueba el resultado en Chrome

Ejecuta Django y abre /lengua/. La imagen debe mostrarse y el audio debe reproducirse.

TERMINAL Y NAVEGADOR

```
python manage.py check  
python manage.py runserver  
http://127.0.0.1:8000/lengua/
```

8

Corrige problemas de carga

Si una imagen no aparece, revisa nombre, extensión y ruta static. Si el audio falla, verifica que el archivo exista y tenga formato compatible.

9

Entrega las evidencias

Incluye capturas del navegador, estructura de carpetas, HTML y CSS, más constancia de autorización de los recursos utilizados.

AUTOEVALUACIÓN

☐ Organicé archivos. ☐ Usé load static. ☐ Probé imagen y audio. ☐ Añadí alt y transcripción.

Práctica 9.

Integración final, pruebas y entrega

OBJETIVO

Integrar las cinco aplicaciones Django, comprobar su funcionamiento y preparar una entrega verificable del proyecto.

1. Integra las cinco aplicaciones

Los equipos Core, Casas, Turismo, Lengua y Videojuego deben comprobar que sus páginas usan el menú y la plantilla base compartida.

LISTA DE INTEGRACIÓN

Core: portada y menú. Casas: listado y formulario. Turismo: contenido turístico. Lengua: imágenes y audio. Videojuego: actividad educativa.

2. Actualiza el repositorio

El responsable de integración revisa las solicitudes de cambio, resuelve conflictos y sincroniza la rama principal. No sobrescribas el trabajo de otros equipos.

TERMINAL · Repositorio compartido

```
git status  
  
git switch main  
  
git pull origin main  
  
python -m pip install -r requirements.txt
```

3. Comprueba la instalación

Activa el entorno virtual y ejecuta las comprobaciones desde la carpeta que contiene manage.py.

TERMINAL · Proyecto Django

```
python manage.py check  
  
python manage.py migrate  
  
python manage.py runserver
```

Práctica 9.

Plan de pruebas funcionales

4. Revisa las rutas y el menú

Abre las cinco secciones y comprueba que sus enlaces funcionen. Registra los errores 404, 500 o enlaces incorrectos.

RUTA	RESULTADO ESPERADO
/	Portada y menú
/casas/	Listado de casas
/turismo/	Contenido turístico
/lengua/	Imagen y audio
/videojuego/	Acceso a actividad

5. Prueba el formulario y la accesibilidad

Registra datos ficticios válidos e inválidos en /casas/nueva/. Comprueba los mensajes de error, el texto alternativo de las imágenes y los controles del audio.

BITÁCORA DE INCIDENCIAS

Registra por cada problema: ruta, descripción, responsable, corrección aplicada y resultado de la nueva prueba.

6. Ejecuta las comprobaciones técnicas

Ejecuta los comandos siguientes. Si el equipo creó pruebas automatizadas, revisa también sus resultados.

TERMINAL • Verificación

```
python manage.py check
```

```
python manage.py test
```

CRITERIOS DE ACEPTACIÓN

Las cinco secciones abren; el menú funciona; el formulario valida; los recursos multimedia cargan; no quedan errores críticos pendientes.

Práctica 9.

Evidencias y presentación final

7. Prepara el paquete de entrega

Incluye el enlace a GitHub, README con instrucciones, requirements.txt, capturas de las cinco secciones y bitácora de pruebas.

ARCHIVOS Y CARPETAS DE ENTREGA

```
README.md

requirements.txt

config/ core/ casas/ turismo/

lengua/ videojuego/ templates/ static/

.gitignore
```

8. Demuestra el funcionamiento

Cada equipo presenta su aportación, la función en vivo y las evidencias de prueba. Explica qué problema resolvió y cómo se integra con las demás aplicaciones.

GUION PARA LA DEMOSTRACIÓN

1. Problema comunitario. 2. Aportación del equipo. 3. Función en vivo. 4. Evidencia de prueba. 5. Mejoras pendientes.

9. Distingue versión local y publicación

La práctica concluye con una versión funcional local y su repositorio. Publicar en Internet requiere configurar alojamiento, seguridad, variables de entorno y archivos estáticos.

10. Valida el contenido comunitario

Antes de difundir el sitio, solicita revisión de fotografías, pronunciaciones y referencias culturales a la instructora y participantes autorizados.

AUTOEVALUACIÓN DEL ESTUDIANTE

☐ Mi sección funciona. ☐ Revisé enlaces. ☐ Participé en las pruebas. ☐ Documenté correcciones. ☐ Colaboré en la presentación.