

 Tema de la sesión:	Práctica 1. Creación del entorno virtual Django		 Fecha:	
¿Qué hicimos hoy?	Se preparó el entorno de Python para desarrollar un sitio con Django.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Python: lenguaje de programación con el que está construido Django. Antes de empezar se comprueba la versión instalada con <code>py --version</code>. • venv (entorno virtual): carpeta aislada con su propio Python y sus propios paquetes. Permite que cada proyecto use sus versiones sin afectar a otros. Cuando está activo, la terminal muestra (entorno). • pip: instalador de paquetes de Python. Con <code>pip install django</code> se descarga Django dentro del entorno virtual activo. • Django: framework web: ofrece una estructura lista (rutas, vistas, plantillas y base de datos) para construir sitios sin empezar desde cero. • Intérprete en VS Code: debe ser el Python del entorno virtual para que el editor reconozca Django.	<pre># Terminal, dentro de E:\Django py -3.14 --version py -3.14 -m venv entorno entorno\Scripts\activate.bat # La terminal ahora muestra (entorno) python -m pip install django python -m django --version # Probar (después de crear el proyecto) python manage.py runserver</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. Crear la carpeta E:\Django y abrirla en la terminal. 2. Verificar Python con <code>py -3.14 --version</code>. 3. Crear el entorno: <code>py -3.14 -m venv entorno</code>. 4. Activarlo con <code>entorno\Scripts\activate.bat</code> y confirmar que aparece (entorno). 5. Instalar Django con pip y comprobar la versión con <code>python -m django --version</code>. 6. Abrir VS Code y elegir el intérprete del entorno virtual.	• La terminal muestra Python 3.14.x y la versión de Django instalada. • Aparece (entorno) al inicio de la línea de comandos. • VS Code usa el intérprete del entorno virtual. • Con runserver, el navegador muestra la página de bienvenida de Django (el cohete).	Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 1. Creación del entorno virtual Django

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 2. Creación de proyecto, aplicación y estructura		 Fecha:	
¿Qué hicimos hoy?	Se organizó el proyecto webproject y la aplicación app1.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Proyecto (webproject): contenedor principal del sitio. Guarda la configuración global en settings.py, urls.py, wsgi.py y asgi.py. Se crea con django-admin startproject. • Aplicación (app1): módulo que agrupa una función concreta del sitio: vistas, modelos y plantillas. Un proyecto puede tener varias apps. • manage.py: herramienta para ejecutar tareas: runserver, startapp, migrate, check, shell. • settings.py: configuración del proyecto: apps instaladas (INSTALLED_APPS), base de datos, plantillas, idioma y archivos estáticos. Cada app nueva se registra aquí. • migrate: crea en db.sqlite3 las tablas iniciales de Django (usuarios, sesiones, administración).	<pre># Terminal con (entorno) activo django-admin startproject webproject . python manage.py startapp app1 # webproject/settings.py INSTALLED_APPS = ["django.contrib.admin", # ... apps de Django ... "app1", # registrar la app nueva] python manage.py check python manage.py migrate</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. Con el entorno activo, ejecutar django-admin startproject webproject . 2. Revisar los archivos creados: manage.py, settings.py, urls.py y wsgi.py. 3. Crear la aplicación con python manage.py startapp app1. 4. Agregar 'app1' a INSTALLED_APPS en settings.py. 5. Ejecutar check y migrate para validar y crear las tablas iniciales.	• Existen manage.py, la carpeta webproject y la carpeta app1. • 'app1' aparece en INSTALLED_APPS de settings.py. • check responde «System check identified no issues». • migrate muestra cada migración inicial con OK y se crea db.sqlite3.	Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 2. Creación de proyecto, aplicación y estructura

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 3. Texto y URL		 Fecha:	
¿Qué hicimos hoy?	Se respondieron solicitudes de navegador con vistas Python y rutas URL.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Vista: función que se escribe en <code>views.py</code> de la aplicación (<code>app1</code>). Recibe el parámetro <code>request</code> y regresa una respuesta, por ejemplo <code>def saludar(request):</code>. • HttpResponse: se importa con <code>from django.http import HttpResponse</code> y regresa un texto al navegador, como "Hola a todos" o "Adiós". Puede llevar etiquetas como <code>h1</code> o <code>h2</code>, pero solo para practicar: el HTML correcto va en plantillas. • urls.py y path(): se importan las vistas (<code>from app1 import views</code>) y se crea una ruta por función: <code>path(' ', views.saludar)</code> abre en la página principal y <code>path('despedirse/', views.despedirse)</code> en <code>despedirse/</code>. • Nombre exacto de la ruta: si se escribe <code>/despedir/</code> en lugar de <code>/despedirse/</code>, Django marca error porque esa ruta no existe; la diagonal final también cuenta. • Guardar y actualizar: cada cambio se guarda con <code>Ctrl + S</code> y se recarga el navegador con <code>python manage.py runserver</code> activo.	<pre># app1/views.py from django.http import HttpResponse def saludar(request): return HttpResponse("<h1>Hola a todos</h1>") def despedirse(request): return HttpResponse("Adiós") # webproject/urls.py from django.contrib import admin from django.urls import path from app1 import views urlpatterns = [path("admin/", admin.site.urls), path("", views.saludar), path("despedirse/", views.despedirse),]</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. En <code>app1/views.py</code> importar <code>HttpResponse</code>. 2. Crear la función <code>saludar(request)</code> que regresa "Hola a todos". 3. En <code>urls.py</code> importar las vistas y agregar <code>path(' ', views.saludar)</code>. 4. Guardar (<code>Ctrl + S</code>) y ejecutar <code>python manage.py runserver</code>. 5. Abrir <code>127.0.0.1:8000</code> y verificar el texto; probar <code>h1/h2</code>. 6. Crear <code>despedirse</code>, su ruta <code>despedirse/</code> y probarla.	• En <code>127.0.0.1:8000</code> aparece «Hola a todos». • Al usar <code>h1</code> o <code>h2</code> dentro de <code>HttpResponse</code>, el texto cambia de tamaño. • En <code>/despedirse/</code> aparece «Adiós». • Una ruta mal escrita (<code>/despedir/</code>) muestra error 404 con la lista de rutas válidas.	Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 3. Texto y URL

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 4. Django include		 Fecha:											
¿Qué hicimos hoy?	Se separaron las rutas principales de las rutas de app1.													
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento													
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...												
• Parámetros en la URL: datos que viajan dentro de la dirección, por ejemplo saludar/Lis/16/. La vista los recibe como argumentos: def saludar(request, nombre, edad). • Convertidores: en la ruta se indica el tipo de dato: <str:nombre> para texto y <int:edad> para enteros. Si falta un valor, Django marca error porque la ruta lo exige. • Usar las variables: el valor se coloca en el texto con llaves, como en "mi nombre es {nombre}", y también se puede operar, por ejemplo edad * 2. • urls.py de la aplicación: cada app debe tener su propio urls.py con sus rutas e importar sus vistas con from . import views. Cuidado: el del proyecto se llama igual. • include(): en el urls.py del proyecto, path('', include('app1.urls')) delega las rutas a la app; así el archivo principal no crece aunque haya muchas aplicaciones.	<pre># app1/views.py def saludar(request, nombre, edad): return HttpResponse(f"Hola a todos, mi nombre es {nombre} " f"y tengo {edad} años") # app1/urls.py (archivo nuevo) from django.urls import path from . import views urlpatterns = [path("saludar/<str:nombre>/<int:edad>/", views.saludar), path("despedirse/", views.despedirse),] # webproject/urls.py from django.urls import path, include urlpatterns = [path("admin/", admin.site.urls), path("", include("app1.urls")),]</pre>													
		 Las dudas que me quedaron												
 Pasos que seguí	 Resultados /salida	Mi  en clase												
1. En la vista saludar agregar los parámetros nombre y edad, y usarlos en el texto. 2. Declarar la ruta saludar/<str:nombre>/<int:edad>/ y probar /saludar/Lis/16/. 3. Opcional: operar con la edad (edad * 2) y ver el cambio. 4. Crear app1/urls.py, mover ahí las rutas e importar con from . import views. 5. En webproject/urls.py importar include y agregar path('', include('app1.urls')). 6. Detener y reiniciar el servidor; probar saludar y despedirse.	• /saludar/Lis/16/ muestra «Hola a todos, mi nombre es Lis y tengo 16 años». • Si falta un parámetro o la edad no es número, Django marca error. • Al multiplicar la edad, la página muestra el resultado (16 × 2 = 32). • Después de usar include(), saludar y despedirse funcionan igual que antes.	<table border="1"> <tr> <td rowspan="2">Estado del aprendizaje</td> <td>☐  ¡Todo claro!</td> </tr> <tr> <td>☐  Entendí algunas cosas.</td> </tr> <tr> <td rowspan="2">Contribuí a la clase con:</td> <td>☐  Lluvia de dudas, ayuda.</td> </tr> <tr> <td>☐  Me voló la cabeza.</td> </tr> <tr> <td colspan="2">Mi nivel de energía y atención fue:</td> </tr> <tr> <td colspan="2"> ☐ Alto ☐ Medio ☐ Bajo </td> </tr> </table>			Estado del aprendizaje	☐  ¡Todo claro!	☐  Entendí algunas cosas.	Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.	☐  Me voló la cabeza.	Mi nivel de energía y atención fue:		☐ Alto ☐ Medio ☐ Bajo	
Estado del aprendizaje	☐  ¡Todo claro!													
	☐  Entendí algunas cosas.													
Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.													
	☐  Me voló la cabeza.													
Mi nivel de energía y atención fue:														
☐ Alto ☐ Medio ☐ Bajo														

 Tema de la sesión: Práctica 4. Django include

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 5. Django Templates	 Fecha:											
¿Qué hicimos hoy?	Se trasladó la presentación a archivos HTML independientes.												
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento												
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...											
• HttpResponse vs. render(): HttpResponse sirve para pruebas con texto; lo normal es render(), que envía archivos HTML al navegador. Ya viene importado en views.py. • Carpeta templates: dentro de app1 se crea templates (en plural) con home.html e info.html. Django la encuentra si app1 está en INSTALLED_APPS. • render(request, plantilla, contexto): recibe la solicitud, el nombre del archivo HTML y, si hace falta, un diccionario con datos: render(request, "info.html", {"name": nombre}). • {{ variable }}: en la plantilla marca dónde se coloca un valor que viene de la vista: Creado por: {{ name }}. Se puede enviar cualquier tipo de dato, incluso listas. • Word Wrap: en VS Code, "editor.wordWrap": "on" en settings.json ajusta las líneas largas al ancho de la pantalla.	<pre># app1/views.py from django.shortcuts import render def home(request): return render(request, "home.html") def info(request): nombre = "Angel" return render(request, "info.html", {"name": nombre}) <!-- app1/templates/info.html --> <h2>Creado por: {{ name }}</h2> # app1/urls.py urlpatterns = [path("", views.home), path("info/", views.info),]</pre>												
		 Las dudas que me quedaron											
 Pasos que seguí	 Resultados /salida	Mi  en clase											
1. Verificar el entorno: F1, elegir el intérprete y abrir una terminal nueva. 2. Crear la carpeta app1/templates con home.html (h1 y párrafo) e info.html (h2 Creado por: {{ name }}). 3. En views.py crear home con render(request, "home.html"). 4. Crear info con la variable nombre y enviarla con {"name": nombre}. 5. Agregar las rutas '' e 'info/' en app1/urls.py y 'app1' en INSTALLED_APPS. 6. Ejecutar runserver y revisar la página principal e /info/.	• La página principal muestra «Bienvenidos a mi sitio web» y el párrafo. • /info/ muestra «Creado por:» seguido del nombre enviado desde la vista. • Al cambiar el nombre en views.py, cambia el texto en la página. • En VS Code las líneas largas se ajustan al ancho (Word Wrap).	<table border="1"> <tr> <td rowspan="2">Estado del aprendizaje</td> <td>☐  ¡Todo claro!</td> </tr> <tr> <td>☐  Entendí algunas cosas.</td> </tr> <tr> <td rowspan="2">Contribuí a la clase con:</td> <td>☐  Lluvia de dudas, ayuda.</td> </tr> <tr> <td>☐  Me voló la cabeza.</td> </tr> <tr> <td colspan="2">Mi nivel de energía y atención fue:</td> </tr> <tr> <td colspan="2"> ☐ Alto ☐ Medio ☐ Bajo </td> </tr> </table>		Estado del aprendizaje	☐  ¡Todo claro!	☐  Entendí algunas cosas.	Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.	☐  Me voló la cabeza.	Mi nivel de energía y atención fue:		☐ Alto ☐ Medio ☐ Bajo	
Estado del aprendizaje	☐  ¡Todo claro!												
	☐  Entendí algunas cosas.												
Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.												
	☐  Me voló la cabeza.												
Mi nivel de energía y atención fue:													
☐ Alto ☐ Medio ☐ Bajo													

 Tema de la sesión: Práctica 5. Django Templates

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 6. Navegación, herencia y nombres de URL		 Fecha:	
¿Qué hicimos hoy?	Se construyó un menú reutilizable mediante herencia de plantillas.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Menú de navegación: lista de enlaces (<nav>, ,) para que el usuario cambie de página sin escribir la dirección. Si solo está en home, al ir a Información no hay forma de regresar. • Plantilla base: archivo con lo que se repite en todas las páginas; se guarda en templates/extras/navegacion.html para tenerlo organizado. • {% extends %}: en la primera línea de home.html e info.html, {% extends 'extras/navegacion.html' %} hereda el menú sin volver a escribirlo. • {% block %} / {% endblock %}: marcan dónde empieza y termina el contenido propio de cada página. Sin el bloque, la página hija no muestra su información. • name y {% url %}: a cada ruta se le da un nombre (name='info') y el menú la llama con {% url 'info' %}. Así se puede cambiar la dirección (info/ → information/) sin romper los enlaces.	<pre> <!-- templates/extras/navegacion.html --> <nav> Home Información </nav> {% block pagina %}{% endblock %} <!-- templates/home.html --> {% extends 'extras/navegacion.html' %} {% block pagina %} <h1>Bienvenidos a mi sitio web</h1> {% endblock %} # app/urls.py path("", views.home, name="home"), path("information/", views.info, name="info"), </pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida		Mi  en clase	
1. Crear en home un <nav> con enlaces a Home e Información y probar que no se puede regresar. 2. Crear templates/extras/navegacion.html y mover ahí el menú. 3. Escribir {% extends 'extras/navegacion.html' %} al inicio de home e info. 4. Agregar {% block pagina %}{% endblock %} en la base y encerrar el contenido de cada página. 5. Dar name='home' y name='info' a las rutas y usar {% url %} en el menú. 6. Cambiar info/ por information/ y comprobar que los enlaces siguen funcionando.	• Home e Información muestran el mismo menú y se puede ir y regresar. • Cada página conserva su propio contenido dentro del bloque pagina. • Sin {% block %}, la página hija aparece vacía; con él, se ve completa. • Al cambiar info/ por information/, los enlaces del menú siguen funcionando.		Estado del aprendizaje ☐ ☀️ ¡Todo claro! ☐ ☁️ Entendí algunas cosas. ☐ 🌧️ Lluvia de dudas, ayuda. ☐ 🤖 Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 6. Navegación, herencia y nombres de URL

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 7. Bases de datos, parte 1		 Fecha:	
¿Qué hicimos hoy?	Se creó el modelo Stores y se verificó su tabla en SQLite.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Base de datos: guarda la información de forma permanente; sin ella, lo que se captura se pierde al salir. Es parte del backend. • DB Browser for SQLite: programa para abrir db.sqlite3 y ver tablas, campos y registros. En la sala de cómputo se instala como administrador. • Modelo: clase en models.py que se convierte en tabla, por ejemplo class Stores(models.Model). Cada atributo es un campo; Django agrega el id (llave primaria). • CharField y TextField: CharField guarda texto corto con max_length=100; TextField guarda texto largo, como la descripción. • makemigrations y migrate: el primero crea el archivo migrations/0001_initial.py (no se edita); el segundo lo aplica y crea la tabla app1_stores junto con las tablas que Django trae por defecto.	<pre># app1/models.py from django.db import models class Stores(models.Model): name = models.CharField(max_length=100) description = models.TextField() # Terminal con (entorno) activo python manage.py makemigrations app1 # crea app1/migrations/0001_initial.py python manage.py migrate app1 # crea la tabla app1_stores en db.sqlite3</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida		Mi  en clase	
1. Descargar e instalar DB Browser for SQLite (64 bits, como administrador). 2. En app1/models.py crear Stores con name (CharField) y description (TextField); guardar. 3. Abrir db.sqlite3 en DB Browser y ver que aún no hay tablas. 4. Con el entorno activo ejecutar python manage.py makemigrations y revisar 0001_initial. 5. Ejecutar python manage.py migrate. 6. En DB Browser presionar F5 y revisar app1_stores con Modificar tabla.	• DB Browser for SQLite queda instalado y abre db.sqlite3. • makemigrations crea app1/migrations/0001_initial.py. • migrate responde «Applying app1.0001_initial... OK». • En DB Browser (F5) aparece app1_stores con id, name y description.		Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 7. Bases de datos, parte 1

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 8. Bases de datos, parte 2		 Fecha:	
¿Qué hicimos hoy?	Se creó Products con precio y relación a Stores.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Usuarios como creadores: los datos los capturarán los usuarios mediante formularios y el panel de Django admin; el programador crea la estructura de tablas. • ForeignKey: campo store que relaciona cada producto con una tienda de la tabla Stores. Es una relación uno a muchos: una tienda, muchos productos. • on_delete=CASCADE: si se elimina una tienda, se eliminan también todos sus productos. • FloatField: guarda números con decimales, como el precio. • Migrar solo una app: makemigrations app1 y migrate app1 procesan solo esa aplicación aunque el proyecto tenga varias. Si app1 no está en INSTALLED_APPS, marca error.	<pre># app1/models.py from django.db import models class Stores(models.Model): name = models.CharField(max_length=100) description = models.TextField() class Products(models.Model): title = models.CharField(max_length=100) price = models.FloatField() store = models.ForeignKey(Stores, on_delete=models.CASCADE) # Solo la aplicación app1 python manage.py makemigrations app1 python manage.py migrate app1 # app1_products: id, title, price, store_id</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. En models.py crear la clase Products debajo de Stores, heredando de models.Model. 2. Agregar title (CharField, 100), price (FloatField) y store (ForeignKey a Stores). 3. Indicar on_delete=models.CASCADE y guardar. 4. Ejecutar python manage.py makemigrations app1. 5. Ejecutar python manage.py migrate app1. 6. En DB Browser revisar app1_products y su relación con stores.	• makemigrations app1 crea la migración del modelo Products. • migrate app1 la aplica sin errores. • En DB Browser aparece app1_products con id, title, price y store_id. • store_id está relacionado con el id de app1_stores.	Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 8. Bases de datos, parte 2

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

Tema de la sesión: Práctica 9. Bases de datos, parte 3		Fecha:							
¿Qué hicimos hoy? Se insertaron y consultaron tiendas desde Django Shell.									
Hoy comencé con: ☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento									
Conceptos clave que aprendí • Django Shell: python manage.py shell abre el intérprete de Python con el proyecto cargado. Primero se importan los modelos: from appl.models import Stores, Products. • Crear y guardar: s = Stores(name=..., description=...) crea el objeto y s.save() lo guarda en la tabla. Los nombres de campo deben coincidir exactamente con el modelo. • objects.all(): devuelve todos los registros de la tabla; sirve para ver cuántos hay. • objects.get(id=1): trae un registro específico por su id. • __str__(): método del modelo que regresa un texto significativo (self.name) en lugar de "Stores object (1)". Para verlo hay que abrir una terminal y un Shell nuevos.		Código clave <pre>python manage.py shell >>> from appl.models import Stores, Products >>> s = Stores(name="Foods", description= ... "En esta sección vamos a vender comidas") >>> s.save() >>> s = Stores(name="Drinks", description= ... "En esta sección vamos a vender bebidas") >>> s.save() >>> Stores.objects.all() <QuerySet [<Stores: Stores object (1)>, ...]> >>> Stores.objects.get(id=1) # appl/models.py (dentro de Stores) def __str__(self): return self.name # Abrir otra terminal y otro shell: # >>> Stores.objects.get(id=1) # <Stores: Foods></pre>							
		Hoy aprendí que ...							
		Las dudas que me quedaron							
Pasos que seguí 1. Abrir la terminal con el entorno activo y ejecutar python manage.py shell. 2. Importar: from appl.models import Stores, Products. 3. Crear Foods y Drinks con Stores(name=..., description=...) y .save(). 4. Revisar la tabla appl_stores en DB Browser (F5). 5. Consultar con Stores.objects.all() y get(id=1). 6. Agregar __str__ al modelo, abrir otra terminal y Shell y repetir la consulta.		Resultados /salida • Las tiendas Foods y Drinks quedan guardadas en appl_stores. • Stores.objects.all() regresa un QuerySet con 2 registros. • get(id=1) y get(id=2) regresan cada tienda. • Con __str__ y un Shell nuevo se ve <Stores: Foods> en lugar de «Stores object (1)».							
		Mi día en clase							
		<table border="1"> <tr> <td rowspan="2"> Estado del aprendizaje </td> <td> ☐ ¡Todo claro! </td> </tr> <tr> <td> ☐ Entendí algunas cosas. </td> </tr> <tr> <td rowspan="2"> Contribuí a la clase con: </td> <td> ☐ Lluvia de dudas, ayuda. </td> </tr> <tr> <td> ☐ Me voló la cabeza. </td> </tr> </table>		Estado del aprendizaje	☐ ¡Todo claro!	☐ Entendí algunas cosas.	Contribuí a la clase con:	☐ Lluvia de dudas, ayuda.	☐ Me voló la cabeza.
Estado del aprendizaje	☐ ¡Todo claro!								
	☐ Entendí algunas cosas.								
Contribuí a la clase con:	☐ Lluvia de dudas, ayuda.								
	☐ Me voló la cabeza.								
		<table border="1"> <tr> <td> ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea. </td> </tr> </table>		☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.					
☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.									
		Mi nivel de energía y atención fue:							
		☐ Alto ☐ Medio ☐ Bajo							

 Tema de la sesión: Práctica 9. Bases de datos, parte 3

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

Tema de la sesión: Práctica 10. Bases de datos, parte 4		Fecha:							
¿Qué hicimos hoy? Se insertaron productos y probaron consultas get y filter.									
Hoy comencé con: ☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento									
Conceptos clave que aprendí		Código clave							
• Crear desde la tienda: con una tienda guardada en s, s.products_set.create(title=..., price=...) crea el producto ya relacionado con ella. • products_set: nombre que Django da a la relación inversa: desde una tienda se llega a todos sus productos. • get(): s.products_set.get(id=4) trae un solo producto de esa tienda. • filter(): regresa los productos que cumplen la condición; si ninguno la cumple, regresa una lista vacía <QuerySet []>, por ejemplo al buscar un id de otra tienda. • Verificar en DB Browser: con F5 se actualiza la tabla: Pan y Pasta tienen store_id 1; Agua y Café, store_id 2.		<pre>>>> from app1.models import Stores, Products >>> s = Stores.objects.get(id=1) # Foods >>> s.products_set.create(title="Bread", price=0.5) >>> s.products_set.create(title="Pasta", price=2) >>> s = Stores.objects.get(id=2) # Drinks >>> s.products_set.create(title="Water", price=0.32) >>> s.products_set.create(title="Coffee", price=1) >>> s.products_set.get(id=3) <Products: Water> >>> s.products_set.filter(id=3) <QuerySet [<Products: Water>]> >>> s.products_set.filter(id=1) <QuerySet []> # el 1 es de la tienda Foods # app1/models.py (dentro de Products) def __str__(self): return self.title</pre>							
		Hoy aprendí que ...							
		Las dudas que me quedaron							
Pasos que seguí		Resultados /salida							
1. Obtener la tienda 1: s = Stores.objects.get(id=1). 2. Crear Pan (0.5) y Pasta (2) con s.products_set.create(...). 3. Cambiar a la tienda 2 y crear Agua (0.32) y Café (1). 4. Actualizar app1_products en DB Browser con F5. 5. Probar s.products_set.get(id=...) y filter(id=...). 6. Buscar un id de otra tienda y ver que regresa una lista vacía.		• En app1_products: Bread y Pasta con store_id 1; Water y Coffee con store_id 2. • s.products_set.get(id=3) regresa Water. • filter(id=3) regresa un QuerySet con Water. • filter(id=1) desde Drinks regresa <QuerySet []>.							
		Mi en clase							
		<table border="1"> <tr> <td rowspan="2"> Estado del aprendizaje </td> <td> ☐ ¡Todo claro! </td> </tr> <tr> <td> ☐ Entendí algunas cosas. </td> </tr> <tr> <td rowspan="2"> Contribuí a la clase con: </td> <td> ☐ Lluvia de dudas, ayuda. </td> </tr> <tr> <td> ☐ Me voló la cabeza. </td> </tr> </table>		Estado del aprendizaje	☐ ¡Todo claro!	☐ Entendí algunas cosas.	Contribuí a la clase con:	☐ Lluvia de dudas, ayuda.	☐ Me voló la cabeza.
Estado del aprendizaje	☐ ¡Todo claro!								
	☐ Entendí algunas cosas.								
Contribuí a la clase con:	☐ Lluvia de dudas, ayuda.								
	☐ Me voló la cabeza.								
		<table border="1"> <tr> <td> ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea. </td> </tr> </table>		☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.					
☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.									
		Mi nivel de energía y atención fue:							
		☐ Alto ☐ Medio ☐ Bajo							

 Tema de la sesión: Práctica 10. Bases de datos, parte 4

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 11. Jinja, parte 1		 Fecha:											
¿Qué hicimos hoy?	Se mostró en HTML la lista de tiendas almacenada en SQLite.													
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento													
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...												
• De la base a la página: la vista importa el modelo (from .models import Stores), consulta los registros y los envía a una plantilla para mostrarlos. • stores_view: la función se llama así y no stores para no confundirla con el modelo Stores. • objects.all() y contexto: s = Stores.objects.all() trae todas las tiendas y {"stores": s} las entrega a stores.html. • Ciclo {% for %}: {% for store in stores %} repite el bloque por cada tienda y se cierra con {% endfor %}. La sangría solo ayuda a ver dónde empieza y termina. • Nombres que coinciden: el name de la ruta, el enlace del menú y las variables (store.name, store.description) deben escribirse igual que en urls.py y en el modelo; si no, hay error o la página sale vacía.	<pre># app1/views.py from django.shortcuts import render from .models import Stores def stores_view(request): s = Stores.objects.all() return render(request, "stores.html", { "stores": s }) # app1/urls.py path("stores/", views.stores_view, name="stores"), <!-- templates/stores.html --> {% extends 'extras/navegacion.html' %} {% block pagina %} <h1>Stores</h1> {% for store in stores %} <h3>{{ store.name }}</h3> <p>{{ store.description }}</p> {% endfor %} {% endblock %} <!-- extras/navegacion.html (nuevo enlace) --> Stores</pre>													
		 Las dudas que me quedaron												
 Pasos que seguí	 Resultados /salida	Mi  en clase												
1. En views.py importar Stores y crear stores_view con Stores.objects.all(). 2. Regresar render(request, "stores.html", {"stores": s}). 3. Agregar path("stores/", views.stores_view, name="stores"). 4. Crear templates/stores.html con extends, block, h1 y el ciclo for (h3 nombre, p descripción). 5. Agregar el enlace Stores en navegacion.html con {% url 'stores' %}. 6. Ejecutar runserver, corregir errores de nombres y comparar con DB Browser.	• El menú tiene el enlace Stores. • /stores/ muestra Foods y Drinks, cada una con su descripción. • Lo que aparece coincide con la tabla app1_stores de DB Browser. • Al agregar una tienda en la base, aparece sin editar el HTML.	<table border="1"> <tr> <td rowspan="2">Estado del aprendizaje</td> <td>☐  ¡Todo claro!</td> </tr> <tr> <td>☐  Entendí algunas cosas.</td> </tr> <tr> <td rowspan="2">Contribuí a la clase con:</td> <td>☐  Lluvia de dudas, ayuda.</td> </tr> <tr> <td>☐  Me voló la cabeza.</td> </tr> <tr> <td colspan="2">Mi nivel de energía y atención fue:</td> </tr> <tr> <td colspan="2"> ☐ Alto ☐ Medio ☐ Bajo </td> </tr> </table>			Estado del aprendizaje	☐  ¡Todo claro!	☐  Entendí algunas cosas.	Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.	☐  Me voló la cabeza.	Mi nivel de energía y atención fue:		☐ Alto ☐ Medio ☐ Bajo	
Estado del aprendizaje	☐  ¡Todo claro!													
	☐  Entendí algunas cosas.													
Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.													
	☐  Me voló la cabeza.													
Mi nivel de energía y atención fue:														
☐ Alto ☐ Medio ☐ Bajo														

 Tema de la sesión: Práctica 11. Jinja, parte 1

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 12. Jinja, parte 2		 Fecha:	
¿Qué hicimos hoy?	Se mostró el catálogo de productos con título y precio.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• products_view: igual que stores_view, pero importa Products y envía {"products": p} a products.html. • Reutilizar código: se puede copiar la vista y la plantilla de tiendas y cambiar solo los nombres: store → product, name → title, description → price. • Mostrar campos: {{ product.title }} - {{ product.price }} imprime el nombre y el precio de cada producto; no es necesario mostrar id ni store_id. • Ruta y menú: cada página nueva necesita su path con nombre (name="products") y su enlace en navegacion.html; si falta el enlace, no aparece en el menú. • Lenguaje de plantillas: las etiquetas {% %} y {{ }} combinan HTML con lógica. En el video se le llama Jinja; Django usa su propio lenguaje con sintaxis casi igual.	<pre># app1/views.py from .models import Stores, Products def products_view(request): p = Products.objects.all() return render(request, "products.html", { "products": p }) # app1/urls.py path("products/", views.products_view, name="products"), <!-- templates/products.html --> {% extends 'extras/navegacion.html' %} {% block pagina %} <h1>Products</h1> {% for product in products %} <h3>{{ product.title }} - {{ product.price }}</h3> {% endfor %} {% endblock %} <!-- extras/navegacion.html (nuevo enlace) --> Products</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. Importar Products y crear products_view (copiar stores_view y adaptar). 2. Crear templates/products.html copiando stores.html. 3. Cambiar el ciclo a {% for product in products %} con title y price. 4. Agregar la ruta products/ con name="products". 5. Agregar el enlace Products en navegacion.html. 6. Recargar y comparar la lista con la tabla app1_products.	• El menú tiene el enlace Products. • /products/ muestra cada producto con su precio: Bread - 0.5, Pasta - 2.0, Water - 0.32, Coffee - 1.0. • La lista coincide con app1_products. • No se muestran id ni store_id, solo título y precio.	Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 12. Jinja, parte 2

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 13. Jinja, condicionales		 Fecha:	
¿Qué hicimos hoy?	Se indicó visualmente si un producto está disponible o vendido.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• BooleanField: sold = models.BooleanField(default=False) agrega el campo vendido; False (0) significa disponible y True (1), vendido. • Migrar el campo nuevo: makemigrations app1 registra el cambio ("Add field sold") y migrate crea la columna. • {% if %} / {% else %} / {% endif %}: condición en la plantilla: si el producto está vendido muestra un círculo rojo; si no, uno verde. Todo lo que se abre se cierra. • Escribir bien las etiquetas: {% y %} van juntos, sin espacio entre la llave y el porcentaje; un espacio de más provoca error. • Cambiar datos en DB Browser: se edita el valor (0 o 1), se presiona Aplicar y después Guardar cambios (Write Changes); si no se guarda, la página no cambia.	<pre># app1/models.py (dentro de Products) sold = models.BooleanField(default=False) # False (0) = disponible, True (1) = vendido python manage.py makemigrations app1 # Add field sold to products python manage.py migrate app1 <!-- templates/products.html --> {% for product in products %} <h3> {% if product.sold == True %}&#128308; {% else %}&#128994;{% endif %} {{ product.title }} - {{ product.price }} </h3> {% endfor %} <!-- &#128308; = círculo rojo, &#128994; = verde. También se puede pegar el emoji directamente. --></pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi 🧑 en clase		
1. Agregar sold = models.BooleanField(default=False) a Products. 2. Ejecutar makemigrations app1 y migrate app1; revisar la columna en DB Browser. 3. En products.html agregar {% if %} ... {% else %} ... {% endif %} con círculo rojo y verde. 4. Ejecutar runserver y corregir espacios dentro de las etiquetas si hay error. 5. En DB Browser poner 1 a Pasta, Aplicar y Guardar cambios. 6. Recargar /products/ y verificar los colores.	• La migración informa «Add field sold to products» y la columna aparece en DB Browser. • Al inicio todos los productos tienen sold = 0 y salen con círculo verde. • Tras poner 1 a Pasta, Aplicar y Guardar cambios, Pasta sale con círculo rojo. • Con espacios dentro de {% %} la página marca error; al quitarlos funciona.	Estado del aprendizaje ☐ 🌞 ¡Todo claro! ☐ 🌤️ Entendí algunas cosas. ☐ 🌧️ Lluvia de dudas, ayuda. ☐ 🤖 Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 13. Jinja, condicionales

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 14. Django Admin		 Fecha:	
¿Qué hicimos hoy?	Se administraron tiendas y productos desde /admin/.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Superusuario: cuenta con todos los permisos, creada con <code>python manage.py createsuperuser</code> (usuario, correo y contraseña); con ella se entra a /admin/. • Usuarios, grupos y permisos: desde el panel se agregan usuarios y se definen grupos con lo que pueden y no pueden hacer. • admin.site.register(): en <code>admin.py</code> se importan Stores y Products y se registran para que aparezcan en el panel. • Relación y CASCADE en el panel: la ForeignKey hace que al crear un producto aparezca la lista de tiendas. Al borrar una tienda, Django avisa y borra también sus productos. • Contraseñas encriptadas: la base guarda la contraseña cifrada: el administrador puede cambiarla, pero no verla ni recuperarla.	<pre>python manage.py createsuperuser # Username, Email, Password (si es corta, # confirmar con y) python manage.py runserver # Navegador: http://127.0.0.1:8000/admin/ # app1/admin.py from django.contrib import admin from .models import Stores, Products admin.site.register(Stores) admin.site.register(Products) # app1/models.py (recordatorio) store = models.ForeignKey(Stores, on_delete=models.CASCADE) # lista de tiendas en el panel y borrado # en cascada de sus productos</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. Detener el servidor y ejecutar <code>python manage.py createsuperuser</code>. 2. Ejecutar <code>runserver</code> y entrar a /admin/ con el usuario creado. 3. En <code>app1/admin.py</code> importar Stores y Products y registrarlos. 4. Agregar la tienda Candies y los productos Chocolate y Gomas. 5. Borrar Candies y leer el aviso de productos relacionados. 6. Revisar en DB Browser las tablas y la contraseña cifrada.	• Se entra a /admin/ con el superusuario. • El panel muestra Users, Groups, Stores y Products. • Al crear un producto aparece la lista de tiendas; Chocolate se guarda en Candies. • Al borrar Candies, el panel lista sus productos y los elimina (CASCADE). • En la tabla de usuarios, la contraseña aparece cifrada.	Estado del aprendizaje	☐ ☀️ ¡Todo claro! ☐ ☁️ Entendí algunas cosas. ☐ ☔ Lluvia de dudas, ayuda. ☐ 🤪 Me voló la cabeza.	
		Contribuí a la clase con:	☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 14. Django Admin

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 15. Forms, parte 1		 Fecha:											
¿Qué hicimos hoy?	Se creó un formulario para registrar nuevas tiendas.													
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento													
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...												
• forms.py: archivo nuevo en app1 donde los formularios se escriben en Python (from django import forms) en lugar de hacerlos a mano en HTML. • NewStoreForm: clase que hereda de forms.Form; cada campo lleva label (texto visible) y, si aplica, max_length. • widget Textarea: widget=forms.Textarea convierte el campo descripción en una caja grande de texto. • Mostrarlo en la página: la vista lo envía en el contexto y la plantilla lo dibuja con {{ form.as_p }} dentro de <form>, junto con {% csrf_token %} y un botón Save. • GET y POST: GET pide la página (ver el formulario); POST envía los datos para guardarlos. Mientras no se programe el POST, el botón no guarda nada.	<pre># app1/forms.py (archivo nuevo) from django import forms class NewStoreForm(forms.Form): name = forms.CharField(label="Name", max_length=50) description = forms.CharField(label="Description", widget=forms.Textarea) # app1/views.py from .forms import NewStoreForm def create_store(request): return render(request, "create_store.html", { "form": NewStoreForm() # en el video: "forms" }) # app1/urls.py path("create_store/", views.create_store, name="create_store"), <!-- templates/create_store.html --> {% extends 'extras/navegacion.html' %} {% block pagina %} <h1>Crear una nueva tienda</h1> <form> {% csrf_token %} {{ form.as_p }} <button>Save</button> </form> {% endblock %}</pre>													
		 Las dudas que me quedaron												
 Pasos que seguí	 Resultados /salida	Mi  en clase												
1. En views.py crear create_store que regresa create_store.html. 2. Agregar la ruta create_store/ con name y el enlace en navegacion.html. 3. Crear templates/create_store.html (fuera de extras) con extends, block y h1. 4. Crear app1/forms.py con NewStoreForm (name y description). 5. Importarlo en views.py y enviarlo en el contexto. 6. Agregar <form>, csrf_token, {{ form.as_p }} y botón; probar.	• El menú tiene el enlace Create store. • La página muestra «Crear una nueva tienda» con las cajas Name y Description. • Description es una caja grande (Textarea). • Al presionar Save todavía no se guarda nada: solo funciona GET.	<table border="1"> <tr> <td rowspan="2">Estado del aprendizaje</td> <td>☐  ¡Todo claro!</td> </tr> <tr> <td>☐  Entendí algunas cosas.</td> </tr> <tr> <td rowspan="2">Contribuí a la clase con:</td> <td>☐  Lluvia de dudas, ayuda.</td> </tr> <tr> <td>☐  Me voló la cabeza.</td> </tr> <tr> <td colspan="2">Mi nivel de energía y atención fue:</td> </tr> <tr> <td colspan="2"> ☐ Alto ☐ Medio ☐ Bajo </td> </tr> </table>			Estado del aprendizaje	☐  ¡Todo claro!	☐  Entendí algunas cosas.	Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.	☐  Me voló la cabeza.	Mi nivel de energía y atención fue:		☐ Alto ☐ Medio ☐ Bajo	
Estado del aprendizaje	☐  ¡Todo claro!													
	☐  Entendí algunas cosas.													
Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.													
	☐  Me voló la cabeza.													
Mi nivel de energía y atención fue:														
☐ Alto ☐ Medio ☐ Bajo														

 Tema de la sesión: Práctica 15. Forms, parte 1

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 16. Forms, parte 2		 Fecha:											
¿Qué hicimos hoy?	Se guardaron tiendas recibidas por el método POST.													
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento													
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...												
• method="post": se escribe en la etiqueta <form> para que el botón envíe los datos al servidor. • request.method: la vista revisa si la petición es GET (mostrar el formulario) o POST (guardar). • request.POST: diccionario con lo que escribió el usuario; se lee con el nombre del campo: request.POST['name']. • Guardar y redirigir: Stores.objects.create(...) guarda la tienda y redirect('stores') lleva a la lista, donde aparece la nueva tienda. • Errores frecuentes: un nombre de campo mal escrito provoca error; con DB Browser abierto la base puede bloquearse, así que conviene cerrarlo. Mejora: validar con form.is_valid().	<pre># app1/views.py from django.shortcuts import render, redirect def create_store(request): if request.method == "GET": return render(request, "create_store.html", { "form": NewStoreForm() }) else: Stores.objects.create(name=request.POST["name"], description=request.POST["description"]) return redirect("stores") <!-- templates/create_store.html --> <form method="post"> {% csrf_token %} {{ form.as_p }} <button>Save</button> </form></pre>													
		 Las dudas que me quedaron												
 Pasos que seguí	 Resultados /salida	Mi  en clase												
1. Agregar method="post" a la etiqueta form. 2. En views.py importar redirect. 3. Separar GET (mostrar) y POST (guardar) con request.method. 4. Guardar con Stores.objects.create(name=request.POST['name'], ...). 5. Redirigir a stores. 6. Registrar «Tienda 4» y verla en la lista.	• Al llenar el formulario y presionar Save, se regresa a /stores/. • «Tienda 4 - Esta tienda vende juguetes» aparece en la lista. • El registro nuevo está en app1_stores. • Un nombre de campo mal escrito en request.POST provoca error.	<table border="1"> <tr> <td rowspan="2">Estado del aprendizaje</td> <td>☐  ¡Todo claro!</td> </tr> <tr> <td>☐  Entendí algunas cosas.</td> </tr> <tr> <td rowspan="2">Contribuí a la clase con:</td> <td>☐  Lluvia de dudas, ayuda.</td> </tr> <tr> <td>☐  Me voló la cabeza.</td> </tr> <tr> <td colspan="2">Mi nivel de energía y atención fue:</td> </tr> <tr> <td colspan="2"> ☐ Alto ☐ Medio ☐ Bajo </td> </tr> </table>			Estado del aprendizaje	☐  ¡Todo claro!	☐  Entendí algunas cosas.	Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.	☐  Me voló la cabeza.	Mi nivel de energía y atención fue:		☐ Alto ☐ Medio ☐ Bajo	
Estado del aprendizaje	☐  ¡Todo claro!													
	☐  Entendí algunas cosas.													
Contribuí a la clase con:	☐  Lluvia de dudas, ayuda.													
	☐  Me voló la cabeza.													
Mi nivel de energía y atención fue:														
☐ Alto ☐ Medio ☐ Bajo														

 Tema de la sesión: Práctica 16. Forms, parte 2

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 17. Forms, parte 3		 Fecha:	
¿Qué hicimos hoy?	Se creó el formulario para dar de alta productos.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Reutilizar lo de tiendas: la vista, la ruta, la plantilla y el formulario de productos se copian de los de tiendas y se cambian los nombres (store → product). • NewProductForm: campos title (CharField, 100), price (FloatField) y store; sold no se pide porque tiene valor por defecto. • Llave foránea al guardar: el producto necesita el id de su tienda: a = Stores.objects.get(name=...) la busca y store_id=a.id la relaciona. • Tienda inexistente: si se escribe una tienda que no existe, get() marca error: la relación protege los datos. • Mejora: ModelChoiceField: muestra una lista con las tiendas guardadas, así el usuario no tiene que escribir el nombre.	<pre># app1/forms.py class NewProductForm(forms.Form): title = forms.CharField(label="Title", max_length=100) price = forms.FloatField(label="Price") store = forms.CharField(label="Store", max_length=100) # Mejora: forms.ModelChoiceField(# queryset=Stores.objects.all()) # app1/views.py from .forms import NewStoreForm, NewProductForm def create_product(request): if request.method == "GET": return render(request, "create_product.html", {"form": NewProductForm()}) else: a = Stores.objects.get(name=request.POST["store"]) Products.objects.create(title=request.POST["title"], price=request.POST["price"], store_id=a.id) return redirect("products") # app1/urls.py path("create_product/", views.create_product, name="create_product"),</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida		 Mi en clase	
1. Copiar create_store y adaptarla como create_product (vista, ruta y plantilla). 2. Crear NewProductForm con title, price y store. 3. Agregar el enlace Create product en el menú. 4. Corregir errores: nombre del formulario, importación, paréntesis. 5. Buscar la tienda con get(name=...) y guardar con store_id=a.id. 6. Probar con una tienda inexistente y luego con Foods.	• El menú tiene Create product y la página muestra Title, Price y Store. • Con una tienda inexistente, get() marca error y no se guarda. • Con Foods, el producto (Rice) se guarda con store_id 1. • Al guardar, se regresa a /products/ con el producto nuevo.		Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 17. Forms, parte 3

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 18. Forms, parte 4	 Fecha:	
 ¿Qué hicimos hoy?	Se conectó la creación de productos con su tienda y se añadieron detalles.		
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento		
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...	
• Parámetro en la ruta: path('details/<int:id>', ...) recibe el id de la tienda que se quiere ver. • get() y filter(): Stores.objects.get(id=id) trae la tienda; Products.objects.filter(store_id=id) trae solo sus productos. • Plantilla details.html: muestra el nombre de la tienda y recorre sus productos; reutiliza el if de vendido/disponible de products.html. • Enlace con parámetro: en stores.html, {% url 'details' store.id %} convierte cada tienda en un enlace a su página de detalles, sin agregar otra opción al menú. • Comprobar: en DB Browser se confirma que cada producto tiene el store_id correcto y que sold quedó en 0 por defecto.	<pre># app1/views.py def details(request, id): s = Stores.objects.get(id=id) p = Products.objects.filter(store_id=id) return render(request, "details.html", { "store": s, "products": p }) # app1/urls.py path("details/<int:id>", views.details, name="details"), <!-- templates/details.html --> {% extends 'extras/navegacion.html' %} {% block pagina %} <h1>{{ store.name }}</h1> {% for product in products %} <h3>{% if product.sold == True %}&#128308; {% else %}&#128994;{% endif %} {{ product.title }} - {{ product.price }}</h3> {% endfor %} {% endblock %} <!-- templates/stores.html (dentro del for) --> <article> <h3>{{ store.name }}</h3> </article></pre>		
		 Las dudas que me quedaron	
 Pasos que seguí	 Resultados /salida	 Mi  en clase	
1. Revisar en DB Browser el store_id y el sold del producto nuevo. 2. Crear la ruta details/<int:id> con name='details'. 3. Crear la vista details con get() y filter(store_id=id). 4. Crear details.html con el nombre de la tienda y el ciclo de productos. 5. En stores.html convertir cada tienda en enlace con {% url 'details' store.id %}. 6. Probar Foods y Drinks y comparar con la base.	• En DB Browser, Rice tiene store_id 1 y sold 0. • En /stores/ cada tienda es un enlace. • Foods abre /details/1 con Bread, Pasta y Rice. • Drinks abre /details/2 con Water y Coffee, cada uno con su círculo de estado.	Estado del aprendizaje ☐ ☀️ ¡Todo claro! ☐ ☁️ Entendí algunas cosas. ☐ 🌧️ Lluvia de dudas, ayuda. ☐ 🤖 Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.
		Mi nivel de energía y atención fue:	
		☐ Alto ☐ Medio ☐ Bajo	

 Tema de la sesión: Práctica 18. Forms, parte 4

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 19. Enviar email automáticamente, parte 1		 Fecha:	
¿Qué hicimos hoy?	Se construyó un formulario de contacto para solicitar el catálogo.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Página de contacto: vista contact, plantilla contact.html, ruta con name="contact" y enlace en el menú: el mismo recorrido de las páginas anteriores. • NewContactForm: formulario con un solo campo, el correo del visitante (en el video, CharField con max_length=100; mejora: EmailField, que valida el formato). • GET y POST en la vista: si la petición es GET se muestra el formulario; si no (POST), se arma y envía el correo. • EmailMessage: objeto del módulo email.message que arma el correo: From (remitente), To (destinatario, tomado de request.POST["email"]), Subject y contenido. • smtplib y SMTP: SMTP es el protocolo de envío de correo. Con smtplib.SMTP("smtp.gmail.com", 587), starttls() cifra la conexión, login() inicia sesión y sendmail() envía.	<pre># app1/forms.py class NewContactForm(forms.Form): email = forms.CharField(label="Email", max_length=100) # Mejora: forms.EmailField valida el formato # app1/views.py from email.message import EmailMessage import smtplib def contact(request): if request.method == "GET": return render(request, "contact.html", {"form": NewContactForm()}) else: remitente = "correo@gmail.com" destinatario = request.POST["email"] mensaje = "Hola" email = EmailMessage() email["From"] = remitente email["To"] = destinatario email["Subject"] = "Asunto" email.set_content(mensaje) smtp = smtplib.SMTP("smtp.gmail.com", 587) smtp.starttls() smtp.login(remitente, password) # práctica 20 smtp.sendmail(remitente, destinatario, email.as_string()) smtp.quit()</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. Crear la vista contact, contact.html y la ruta contact con name. 2. En forms.py crear NewContactForm con el campo email. 3. En contact.html: extends, block, h1, form POST, csrf_token, {{ form.as_p }} y botón Send. 4. Agregar Contact al menú y probar la página. 5. Importar EmailMessage y smtplib; separar GET y POST. 6. Armar remitente, destinatario y mensaje, y escribir la conexión SMTP.	• El menú tiene Contact y la página muestra el campo Email y el botón Send. • Con GET se ve el formulario; al enviar se ejecuta la rama POST. • El código de envío queda listo; falta la contraseña de aplicación (práctica 20).	Estado del aprendizaje ☐ ☀️ ¡Todo claro! ☐ ☁️ Entendí algunas cosas. ☐ 🌧️ Lluvia de dudas, ayuda. ☐ 🤖 Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 19. Enviar email automáticamente, parte 1

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 20. Enviar email automáticamente, parte 2		 Fecha:	
¿Qué hicimos hoy?	Se completó el envío del catálogo de productos por Gmail.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• Cuenta de correo de la tienda: se crea un correo propio de la empresa (Gmail, Outlook...) desde el que saldrán los mensajes. • Verificación en dos pasos: en la sección Seguridad de la cuenta debe estar activada; es requisito para obtener la contraseña de aplicación. • Contraseña de aplicación: clave que da Gmail para que un programa use la cuenta; no es la contraseña normal. Se guarda en una variable de entorno, nunca en el código ni en capturas. • Mensaje con el catálogo: un ciclo for sobre <code>Products.objects.all()</code> agrega al texto el nombre y precio de cada producto. • try / except y redirect: el envío va dentro de try; si falla, except muestra el error en consola. Si todo sale bien, <code>redirect("home")</code> regresa al inicio.	<pre># appl/views.py (rama POST de contact) import os remitente = "tienda@gmail.com" destinatario = request.POST["email"] password = os.environ["SMTP_APP_PASSWORD"] mensaje = "Saludos, gracias por contactarnos. " mensaje += "Nuestros productos son los siguientes: " for producto in Products.objects.all(): mensaje += f"{producto.title} - \${producto.price} " email = EmailMessage() email["From"] = remitente email["To"] = destinatario email["Subject"] = "Contacto tienda, nuestros productos" email.set_content(mensaje) try: smtp = smtplib.SMTP("smtp.gmail.com", 587) smtp.starttls() smtp.login(remite, password) smtp.sendmail(remite, destinatario, email.as_string()) smtp.quit() except Exception as error: print("Error al enviar:", error) return redirect("home")</pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida	Mi  en clase		
1. Crear el correo de la tienda y activar la verificación en dos pasos. 2. Generar una contraseña de aplicación (por ejemplo, «Django»). 3. Guardarla como variable de entorno, no en el código. 4. Armar el mensaje con un for de productos y precios. 5. Poner el envío dentro de try/except y redirigir a home. 6. Enviar desde Contact y revisar el correo recibido.	• La cuenta tiene verificación en dos pasos y contraseña de aplicación. • Al enviar el formulario, la página regresa a Home. • Llega un correo «Contacto tienda, nuestros productos» con cada producto y su precio. • Si algo falla, el error aparece en la consola y no detiene la página.	Estado del aprendizaje ☐ ☀️ ¡Todo claro! ☐ ☁️ Entendí algunas cosas. ☐ 🌧️ Lluvia de dudas, ayuda. ☐ 🤖 Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 20. Enviar email automáticamente, parte 2

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 21. Elementos estáticos		 Fecha:								
¿Qué hicimos hoy?	Se añadieron CSS, imágenes y favicon al proyecto.										
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento										
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...									
• Dinámico vs. estático: lo dinámico cambia con la base de datos; lo estático (imágenes, CSS) no necesita procesarse y da el estilo de la página. • Carpeta static y settings.py: los archivos van en app1/static; settings.py ya incluye django.contrib.staticfiles y STATIC_URL = "static/". • {% load static %}: se escribe en cada plantilla que use archivos estáticos; después {% static 'icono.png' %} genera su ruta. • Hoja base.css: en static/css/base.css; se enlaza con <link rel="stylesheet"> en navegacion.html, que es común a todas las páginas. • <title> por página: cada plantilla tiene su título de pestaña; en details se usa {{ store.name }} details para mostrar el nombre de la tienda.	<pre># webproject/settings.py (ya viene así) STATIC_URL = "static/" # + django.contrib.staticfiles /* app1/static/css/base.css */ body { background: black; color: white; } <!-- templates/extras/navegacion.html --> {% load static %} <!DOCTYPE html> <html lang="es"> <head> <meta charset="UTF-8"> <link rel="stylesheet" href="{% static 'css/base.css' %}"> </head> <body> <nav> ... </nav> {% block pagina %}{% endblock %} </body> </html> <!-- templates/home.html --> {% load static %} <title>Home</title> <!-- templates/details.html --> <title>{{ store.name }} details</title></pre>										
		 Las dudas que me quedaron									
 Pasos que seguí	 Resultados /salida	Mi  en clase									
1. Crear app1/static y revisar staticfiles y STATIC_URL en settings.py. 2. Descargar un ícono PNG (64 px), renombrarlo icono.png y pasarlo a static. 3. En contact.html y home.html: {% load static %} e . 4. Crear static/css/base.css con fondo negro y texto blanco. 5. Dar estructura HTML a navegacion.html, cargar static y enlazar base.css. 6. Agregar <title> a cada plantilla (details con el nombre de la tienda).	• El ícono de la tienda aparece en Contact y en Home (con width 150 se ve más grande). • Todas las páginas tienen fondo negro y texto blanco. • Cada pestaña muestra su título (Home, Stores, Products, Contact...). • En details la pestaña muestra «Foods details», «Drinks details», etc.	<table border="1"> <tr> <td rowspan="2">Estado del aprendizaje</td> <td>☐  ¡Todo claro!</td> </tr> <tr> <td>☐  Entendí algunas cosas.</td> </tr> <tr> <td></td> <td>☐  Lluvia de dudas, ayuda.</td> </tr> <tr> <td></td> <td>☐  Me voló la cabeza.</td> </tr> </table>			Estado del aprendizaje	☐  ¡Todo claro!	☐  Entendí algunas cosas.		☐  Lluvia de dudas, ayuda.		☐  Me voló la cabeza.
Estado del aprendizaje	☐  ¡Todo claro!										
	☐  Entendí algunas cosas.										
	☐  Lluvia de dudas, ayuda.										
	☐  Me voló la cabeza.										
		<table border="1"> <tr> <td rowspan="2">Contribuí a la clase con:</td> <td>☐ pregunta, ☐ escuchando,</td> </tr> <tr> <td>☐ ayudando a un compañero,</td> </tr> <tr> <td></td> <td>☐ compartiendo una idea.</td> </tr> </table>			Contribuí a la clase con:	☐ pregunta, ☐ escuchando,	☐ ayudando a un compañero,		☐ compartiendo una idea.		
Contribuí a la clase con:	☐ pregunta, ☐ escuchando,										
	☐ ayudando a un compañero,										
	☐ compartiendo una idea.										
		Mi nivel de energía y atención fue:									
		☐ Alto ☐ Medio ☐ Bajo									

 Tema de la sesión: Práctica 21. Elementos estáticos

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

Tema de la sesión: Práctica 22. Mejora de página, parte 1		Fecha:	
¿Qué hicimos hoy? Se mejoró la interfaz Home y el estilo general del catálogo.			
Hoy comencé con: ☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
Conceptos clave que aprendí		Código clave	
• Favicon: <code><link rel="shortcut icon" href="{% static 'icono.png' %}" type="image/png"></code> en <code>navegacion.html</code> cambia el ícono de la pestaña en todo el sitio. • Otro puerto: si el servidor está lento o no se ven los cambios, se puede usar <code>runserver 3000</code> en lugar del 8000. • Clases CSS: <code>.nombre { }</code> define un estilo y <code>class="nombre"</code> lo aplica; si no se pone el <code>class</code> en el HTML, el estilo no se ve. • Contenedor main: <code><main class="all"></code> envuelve el bloque de cada página para centrar el contenido y darle márgenes (<code>padding</code>). • Flexbox en el menú: <code>display: flex</code> pone la lista en horizontal, <code>justify-content: space-between</code> reparte el espacio y <code>list-style: none</code> quita las viñetas.		<pre> <!-- templates/extras/navegacion.html --> <link rel="shortcut icon" type="image/png" href="{% static 'icono.png' %}"> ... <nav class="navigation"> Home ... </nav> <main class="all"> {% block pagina %}{% endblock %} </main> /* app1/static/css/base.css */ .all { max-width: 1500px; margin: auto; padding: 10px 20px; text-align: center; } .navigation { background: #1e3a8a; } .navigation ul { display: flex; justify-content: space-between; list-style: none; } .navigation ul li a { color: white; text-decoration: none; font-size: x-large; font-style: italic; padding: 0px 20px; } </pre>	
		Hoy aprendí que ...	
		Las dudas que me quedaron	
Pasos que seguí		Resultados /salida	
1. Agregar el favicon en <code>navegacion.html</code> y revisarlo en la pestaña. 2. Corregir el título de <code>details</code> (nombre de variable en la vista). 3. Si no se ven cambios, ejecutar <code>runserver 3000</code>. 4. Envolver el bloque en <code><main class="all"></code> y crear <code>.all</code> en <code>base.css</code>. 5. Crear <code>.navigation</code> (color de fondo) y aplicarla al <code><nav></code>. 6. Menú horizontal con flex, sin viñetas, y estilo de enlaces.		• La pestaña muestra el ícono de la tienda (favicon) en todas las páginas. • Con <code>runserver 3000</code> la página abre en <code>127.0.0.1:3000</code>. • El contenido aparece centrado y el menú tiene fondo azul. • El menú es horizontal, sin viñetas, con letra blanca, grande y cursiva.	
		Mi  en clase	
Estado del aprendizaje		☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	
Contribuí a la clase con:		☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:	
		☐ Alto ☐ Medio ☐ Bajo	

 Tema de la sesión: Práctica 22. Mejora de página, parte 1

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

 Tema de la sesión:	Práctica 23. Mejora de página, parte 2		 Fecha:	
¿Qué hicimos hoy?	Se personalizaron formularios, tarjetas, detalles y campos con íconos.			
Hoy comencé con:	☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
 Conceptos clave que aprendí	 Código clave	 Hoy aprendí que ...		
• max-width y margin: auto: limitan el ancho de un elemento y lo centran en la pantalla. • :hover: estilo que se aplica al pasar el mouse, por ejemplo cambiar el color del enlace a rojo y su tamaño. • Clase select (tarjetas): fondo de color, padding y ancho máximo; se aplica a cada <article> de stores.html y al bloque de details para que parezcan tarjetas. • Estilo en formularios: en forms.py cada campo recibe un widget con attrs={"class": "text-input"} para aplicar la clase CSS a sus cajas. • Widgets según el dato: TextInput para texto corto, Textarea para descripción y NumberInput para el precio, que solo acepta números.	<pre> /* appl/static/css/base.css */ .navigation { padding: 10px; max-width: 1500px; margin: auto; } .navigation ul li a:hover { color: red; font-size: large; } .select { background: orange; padding: 5px 0px; max-width: 500px; margin: auto; } .select a { color: white; } .select a:hover { color: red; font-size: large; } .text-input { background: #fde7ef; margin: 10px; text-align: center; } </pre> <pre> <!-- stores.html y details.html --> <article class="select"> ... </article> # appl/forms.py name = forms.CharField(label="Name", max_length=50, widget=forms.TextInput(attrs={"class": "text-input"})) description = forms.CharField(label="Description", widget=forms.Textarea(attrs={"class": "text-input"})) price = forms.FloatField(label="Price", widget=forms.NumberInput(attrs={"class": "text-input"})) </pre>			
		 Las dudas que me quedaron		
 Pasos que seguí	 Resultados /salida		Mi  en clase	
1. Agregar padding, max-width y margin a .navigation. 2. Crear el :hover de los enlaces del menú (rojo, más chico). 3. Crear .select y aplicarla a los article de stores.html y a details. 4. Agregar color blanco y :hover a los enlaces de .select. 5. Crear .text-input en base.css. 6. En forms.py agregar widgets con attrs a todos los campos y probar.	• Al pasar el mouse sobre el menú, el texto se pone rojo. • En Stores cada tienda se ve como tarjeta de color; su enlace cambia a rojo con el mouse. • Details usa la misma tarjeta. • Las cajas de los formularios tienen fondo rosa suave y texto centrado; Price solo acepta números.		Estado del aprendizaje ☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	Contribuí a la clase con: ☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.
		Mi nivel de energía y atención fue:		
		☐ Alto ☐ Medio ☐ Bajo		

 Tema de la sesión: Práctica 23. Mejora de página, parte 2

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____

Tema de la sesión: Práctica 24. URL pública		Fecha:	
¿Qué hicimos hoy? Se probó el sitio en el puerto 3000 y se reenvió con VS Code.			
Hoy comencé con: ☐ Con curiosidad ☐ Con entusiasmo ☐ Sin ganas ☐ Con aburrimiento			
Conceptos clave que aprendí		Código clave	
Clase .btn: estilo para botones (fondo, padding, tamaño de letra) que se aplica con <code>class="btn"</code> en <code>create_store</code>, <code>create_product</code> y <code>contact</code>. Reenvío de puertos: en VS Code, pestaña Puertos → Reenviar un puerto (3000); VS Code crea un túnel con una dirección web que apunta al servidor local. Cuenta de GitHub: el reenvío pide iniciar sesión con GitHub y autorizar la conexión con VS Code. Visibilidad: Privado solo abre con la cuenta del dueño; Público permite que cualquiera con el enlace vea la página. Al abrirla aparece un aviso de túnel de desarrollador. Túnel no es hosting: la página, los formularios y <code>/admin/</code> funcionan mientras el servidor esté encendido; al apagarlo el enlace deja de responder. Para publicar de forma permanente se usa un hosting.		<pre> /* app1/static/css/base.css */ .btn { background: #2563eb; padding: 10px 15px; font-size: x-large; } <!-- create_store, create_product, contact --> <button class="btn">Save</button> # 1. Servidor encendido python manage.py runserver 3000 # 2. VS Code > Puertos > Reenviar un puerto: 3000 # 3. Iniciar sesión con GitHub y autorizar # 4. Clic derecho > Visibilidad del puerto # Privado: solo con tu cuenta # Público: cualquiera con el enlace # 5. Abrir https://...devtunnels.ms/ # y aceptar el aviso del túnel # 6. Probar /stores/, /create_store/ y /admin/ # Al detener runserver (Ctrl + C) # el enlace deja de funcionar. </pre>	
		Hoy aprendí que ...	
		Las dudas que me quedaron	
Pasos que seguí		Resultados /salida	
- Crear <code>.btn</code> en <code>base.css</code> y aplicarla a los botones de los tres formularios. - Ejecutar <code>python manage.py runserver 3000</code>. - En VS Code: Puertos → Reenviar un puerto → 3000. - Iniciar sesión con GitHub y autorizar. - Cambiar la visibilidad (clic derecho) y abrir la dirección; aceptar el aviso. - Probar crear una tienda y entrar a <code>/admin/</code> desde el enlace.		- Los botones de Create store, Create product y Contact tienen el estilo <code>.btn</code>. - La pestaña Puertos muestra el 3000 con una dirección <code>https://...devtunnels.ms</code>. - Desde esa dirección se abre el sitio, se crea una tienda (Shop 1) y se entra a <code>/admin/</code>. - Al detener runserver, el enlace deja de responder.	
		Mi  en clase	
Estado del aprendizaje		☐  ¡Todo claro! ☐  Entendí algunas cosas. ☐  Lluvia de dudas, ayuda. ☐  Me voló la cabeza.	
Contribuí a la clase con:		☐ pregunta, ☐ escuchando, ☐ ayudando a un compañero, ☐ compartiendo una idea.	
		Mi nivel de energía y atención fue:	
		☐ Alto ☐ Medio ☐ Bajo	

 Tema de la sesión: Práctica 24. URL pública

 Fecha:

Evidencia de la práctica · Capturas de pantalla

Pega en cada recuadro la captura que se indica y escribe en una línea qué demuestra.

Captura 1 · Estructura

Pega aquí tu captura
Carpetas y archivos del proyecto en VS Code

¿Qué demuestra? _____

Captura 2 · Código

Pega aquí tu captura
Archivo que escribiste o modificaste (views.py, urls.py, HTML...)

¿Qué demuestra? _____

Captura 3 · Consola o comandos

Pega aquí tu captura
Terminal con los comandos ejecutados y su respuesta

¿Qué demuestra? _____

Captura 4 · Salida o resultados

Pega aquí tu captura
Página en el navegador, DB Browser o panel admin

¿Qué demuestra? _____