acunab: como achico las letras de vscode?

GitHub Copilot: Para achicar el tamaño de las letras en Visual Studio Code, sigue estos pasos:

1. Abre la configuración de VS Code presionando `Cmd + ,` en tu teclado.

2. En la barra de búsqueda de configuración, escribe "Font Size".

3. Ajusta el valor de "Editor: Font Size" a un número menor. Por ejemplo, si está en 14, puedes cambiarlo a 12.

Alternativamente, puedes usar atajos de teclado:

- Para disminuir el tamaño de la fuente: `Cmd + -`

- Para aumentar el tamaño de la fuente: `Cmd + =`

Estos atajos te permitirán ajustar rápidamente el tamaño de la fuente en el editor.

acunab: luego de crear la aplicación debo crear el entorno virtual?

VS Code: Sí, después de crear la aplicación, es una buena práctica crear un entorno virtual para gestionar las dependencias de tu proyecto de manera aislada. Aquí tienes los pasos para crear un entorno virtual en Visual Studio Code:

1. Abre la carpeta de tu proyecto en Visual Studio Code.

2. Abre la Paleta de Comandos y ejecuta el comando \*\*Python: Create Environment\*\*.

3. Selecciona el tipo de entorno que deseas crear, ya sea \*\*Venv\*\* o \*\*Conda\*\*.

4. Selecciona el intérprete de Python que deseas usar como base para el nuevo entorno virtual.

5. Una vez creado el entorno, asegúrate de seleccionarlo usando el comando \*\*Python: Select Interpreter\*\*.

Esto te permitirá trabajar en un entorno aislado, evitando conflictos con otras versiones de paquetes que puedas tener instaladas globalmente.

acunab: centrate en la carpeta certificacionTD, te haré preguntas solo de ese proyecto

Workspace: Aún se está generando el índice del área de trabajo; es posible que la respuesta sea menos precisa.

Entendido, me centraré en la carpeta

certificacionTD

. ¿En qué puedo ayudarte específicamente con este proyecto?

acunab: creame un entorno virtual

VS Code: Para crear un entorno virtual en Visual Studio Code, puedes seguir estos pasos:

1. Abre la carpeta de tu proyecto en Visual Studio Code.

2. Ejecuta el comando \*\*Python: Create Environment\*\*.

Mostrar en la paleta de comandos

3. Selecciona el tipo de entorno que deseas crear, ya sea \*\*Venv\*\* o \*\*Conda\*\*.

4. Selecciona el intérprete de Python que deseas usar como base para el nuevo entorno virtual.

5. Una vez creado el entorno, asegúrate de seleccionarlo usando el comando \*\*Python: Select Interpreter\*\*.

Mostrar en la paleta de comandos

Esto te permitirá trabajar en un entorno aislado, evitando conflictos con otras versiones de paquetes que puedas tener instaladas globalmente.

acunab: cual es el código para creae un entorno virtual?

GitHub Copilot: Para crear un entorno virtual en tu proyecto, puedes usar el siguiente comando en la terminal:

1. Abre la terminal en Visual Studio Code (`Ctrl + ` o desde el menú `Ver > Terminal`).

2. Navega a la carpeta de tu proyecto

certificacionTD

:

```sh

cd path/to/certificacionTD

```

3. Ejecuta el siguiente comando para crear un entorno virtual usando `venv`:

```sh

python3 -m venv venv

```

Esto creará un entorno virtual llamado `venv` en la carpeta de tu proyecto.

4. Activa el entorno virtual:

```sh

source venv/bin/activate

```

Una vez activado, verás el nombre del entorno virtual en tu terminal, indicando que estás trabajando dentro de él.

acunab: Un sistema solicitado por el dueño de una lavandería para gestionar todas las operaciones del negocio,

abarcando desde la recepción de prendas hasta el cobro final. Este sistema debe permitir:

• Registro de clientes y empleados.

• Gestión de prendas y órdenes de servicio.

• Seguimiento de las órdenes desde su creación hasta la finalización.

• Gestión de pagos mediante distintos métodos de pago.

• Generación de reportes de ventas y cobros.

• Mantenimiento de un historial de clientes y sus órdenes previas.

Funcionalidades para el usuario (Empleados):

1. Generación de órdenes de servicio: Permitir asociar prendas, cliente y empleado a la orden.

2. Seguimiento de órdenes: Visualizar el estado de las órdenes de servicio y actualizar su estatus.

3. Cobro de órdenes de servicio: Cambiar el estado a "pagada" y registrar la transacción en ventas.

4. Reportes de ventas: Mostrar el total de ingresos en un periodo determinado, agrupando por

empleado o tipo de prenda.

Funcionalidades para Administradores:

1. Registro de nuevos clientes.

2. Alta de empleados.

3. Gestión de prendas: Control de tipo y costo de limpieza.

4. Definición de métodos de pago.

5. Control y actualización del estatus de las órdenes de servicio.

Requerimientos Funcionales:

1. Manejo de usuarios con roles diferenciados: administrador y usuario(empleado).

2. Validación de formularios en FrontEnd y BackEnd.

3. Generación de reportes para ventas..

4. Menú de navegación para acceso organizado a funcionalidades.

Requerimientos No Funcionales:

1. Framework backend: Python/Django.

2. Base de datos: PostgreSQL.

3. ORM de Django para la capa de datos.

4. Frontend: diseño responsivo utilizando Bootstrap.

5. Autenticación de usuarios mediante Django-Auth o un sistema personalizado.

6. Archivo README.txt con instrucciones para ejecutar el proyecto y credenciales de acceso.

Diagrama Entidad-Relación (ERD):

+------------------+ +-------------------+ +-------------

--------+

| Cliente | 1..\* | OrdenDeServicio | \*..1 |

Empleado |

+------------------+ +-------------------+ +-------------

--------+

| id | | id | | id

|

| nombre | | cliente\_id (FK) | | nombre

|

| telefono | | empleado\_id (FK) | | rol

(admin,empleado)|

| correo | | fecha\_creacion | +-------------

--------+

| direccion | | estado |

+------------------+ | total |

| pago\_realizado |

M +-------------------+

\* 1

M |

+-------------------+

| Prenda | |

+-------------------+ |

| id | |

| tipo\_prenda | |

| precio | |

| descripcion | |

+-------------------+ 1

+-------------------+

| Venta |

+-------------------+

| id |

| orden\_servicio\_id |

| monto\_total |

| fecha\_venta |

| metodo\_pago |

+-------------------+

Este diagrama representa las relaciones clave entre las entidades del sistema. Muestra cómo interactúan las

entidades OrdenDeServicio, Venta, MétodoDePago e Historial con las entidades Cliente, Empleado

y Prenda.

1. Cliente: Cada cliente puede tener múltiples órdenes de servicio. Los clientes se identifican por su

nombre, teléfono, correo y dirección.

2. Empleado: Los empleados registran las órdenes de servicio y pueden tener diferentes roles

(administrador o empleado), lo cual define los permisos para realizar diferentes tareas.

3. OrdenDeServicio: La orden de servicio tiene una relación directa con un cliente y un empleado.

Está asociada con múltiples prendas, y su total depende de las prendas agregadas. Tiene un estado de

avance (pendiente, en proceso, completada) y puede vincularse a una venta al ser pagada.

4. Prenda: Cada prenda tiene un tipo (camisa, pantalón, vestido, etc.), un precio y una descripción del

tratamiento que se aplicará (lavado, planchado, etc.). Las prendas son agregadas a una orden de

servicio, y su precio se suma para calcular el total de la orden.

5. Venta: Cuando se cobra una orden de servicio, se genera una venta que almacena el monto total y la

fecha. Cada venta está vinculada a un método de pago (efectivo, tarjeta, etc.) y se asocia directamente

con la orden de servicio completada.

GitHub Copilot: Para comenzar con este proyecto, primero crearemos el entorno virtual y luego configuraremos el proyecto Django. Aquí están los pasos detallados:

1. \*\*Crear el entorno virtual\*\*:

```sh

cd path/to/certificacionTD

python3 -m venv venv

source venv/bin/activate

```

2. \*\*Instalar Django y otras dependencias\*\*:

```sh

pip install django psycopg2-binary

```

3. \*\*Crear el proyecto Django\*\*:

```sh

django-admin startproject lavanderia

cd lavanderia

```

4. \*\*Crear la aplicación principal\*\*:

```sh

python manage.py startapp gestion

```

5. \*\*Configurar la base de datos PostgreSQL\*\* en `lavanderia/settings.py`:

```python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'nombre\_de\_tu\_base\_de\_datos',

'USER': 'tu\_usuario',

'PASSWORD': 'tu\_contraseña',

'HOST': 'localhost',

'PORT': '5432',

}

}

```

6. \*\*Agregar la aplicación `gestion` al archivo `settings.py`\*\*:

```python

INSTALLED\_APPS = [

...

'gestion',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

```

7. \*\*Definir los modelos en `gestion/models.py`\*\*:

```python

from django.db import models

from django.contrib.auth.models import User

class Cliente(models.Model):

nombre = models.CharField(max\_length=100)

telefono = models.CharField(max\_length=15)

correo = models.EmailField()

direccion = models.TextField()

def \_\_str\_\_(self):

return self.nombre

class Empleado(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

nombre = models.CharField(max\_length=100)

rol = models.CharField(max\_length=20, choices=[('admin', 'Administrador'), ('empleado', 'Empleado')])

def \_\_str\_\_(self):

return self.nombre

class Prenda(models.Model):

tipo\_prenda = models.CharField(max\_length=50)

precio = models.DecimalField(max\_digits=10, decimal\_places=2)

descripcion = models.TextField()

def \_\_str\_\_(self):

return self.tipo\_prenda

class OrdenDeServicio(models.Model):

cliente = models.ForeignKey(Cliente, on\_delete=models.CASCADE)

empleado = models.ForeignKey(Empleado, on\_delete=models.CASCADE)

fecha\_creacion = models.DateTimeField(auto\_now\_add=True)

estado = models.CharField(max\_length=20, choices=[('pendiente', 'Pendiente'), ('en\_proceso', 'En Proceso'), ('completada', 'Completada')])

total = models.DecimalField(max\_digits=10, decimal\_places=2)

pago\_realizado = models.BooleanField(default=False)

def \_\_str\_\_(self):

return f"Orden {self.id} - {self.cliente.nombre}"

class Venta(models.Model):

orden\_servicio = models.ForeignKey(OrdenDeServicio, on\_delete=models.CASCADE)

monto\_total = models.DecimalField(max\_digits=10, decimal\_places=2)

fecha\_venta = models.DateTimeField(auto\_now\_add=True)

metodo\_pago = models.CharField(max\_length=50)

def \_\_str\_\_(self):

return f"Venta {self.id} - {self.orden\_servicio.cliente.nombre}"

```

8. \*\*Realizar las migraciones\*\*:

```sh

python manage.py makemigrations

python manage.py migrate

```

9. \*\*Crear un superusuario para acceder al panel de administración\*\*:

```sh

python manage.py createsuperuser

```

10. \*\*Ejecutar el servidor de desarrollo\*\*:

```sh

python manage.py runserver

```

Con estos pasos, tendrás la estructura básica del proyecto configurada. Puedes continuar desarrollando las vistas, formularios y plantillas para completar las funcionalidades requeridas.

acunab: de aquí en adelante, entregame todo en python3

GitHub Copilot: Entendido, aquí tienes el código en Python 3 para completar los modelos en

models.py

:

```python

from django.db import models

from django.contrib.auth.models import User

class Cliente(models.Model):

nombre = models.CharField(max\_length=100)

telefono = models.CharField(max\_length=15)

correo = models.EmailField()

direccion = models.TextField()

def \_\_str\_\_(self):

return self.nombre

class Empleado(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

nombre = models.CharField(max\_length=100)

rol = models.CharField(max\_length=20, choices=[('admin', 'Administrador'), ('empleado', 'Empleado')])

def \_\_str\_\_(self):

return self.nombre

class Prenda(models.Model):

tipo\_prenda = models.CharField(max\_length=50)

precio = models.DecimalField(max\_digits=10, decimal\_places=2)

descripcion = models.TextField()

def \_\_str\_\_(self):

return self.tipo\_prenda

class OrdenDeServicio(models.Model):

cliente = models.ForeignKey(Cliente, on\_delete=models.CASCADE)

empleado = models.ForeignKey(Empleado, on\_delete=models.CASCADE)

fecha\_creacion = models.DateTimeField(auto\_now\_add=True)

estado = models.CharField(max\_length=20, choices=[('pendiente', 'Pendiente'), ('en\_proceso', 'En Proceso'), ('completada', 'Completada')])

total = models.DecimalField(max\_digits=10, decimal\_places=2)

pago\_realizado = models.BooleanField(default=False)

def \_\_str\_\_(self):

return f"Orden {self.id} - {self.cliente.nombre}"

class Venta(models.Model):

orden\_servicio = models.ForeignKey(OrdenDeServicio, on\_delete=models.CASCADE)

monto\_total = models.DecimalField(max\_digits=10, decimal\_places=2)

fecha\_venta = models.DateTimeField(auto\_now\_add=True)

metodo\_pago = models.CharField(max\_length=50)

def \_\_str\_\_(self):

return f"Venta {self.id} - {self.orden\_servicio.cliente.nombre}"

```

Este código completa los modelos necesarios para tu sistema de gestión de lavandería. Ahora puedes proceder a realizar las migraciones y continuar con el desarrollo de las vistas, formularios y plantillas.

acunab: el punto 8 al 10 entregamelos con python3

GitHub Copilot: Claro, aquí tienes los pasos 8 al 10 con los comandos en Python 3:

8. \*\*Realizar las migraciones\*\*:

```sh

python3 manage.py makemigrations

python3 manage.py migrate

```

9. \*\*Crear un superusuario para acceder al panel de administración\*\*:

```sh

python3 manage.py createsuperuser

```

10. \*\*Ejecutar el servidor de desarrollo\*\*:

```sh

python3 manage.py runserver

```

Estos comandos te permitirán configurar la base de datos, crear un superusuario y ejecutar el servidor de desarrollo para tu proyecto Django.

acunab: @terminal /explain #terminalLastCommand

Terminal: Parece que el comando que ejecutaste tiene un error tipográfico. Debería haber un espacio entre `makemigrations` y `python3 manage.py migrate`. Además, asegúrate de que el archivo `manage.py` exista en el directorio especificado.

Prueba ejecutar los comandos por separado:

```bash

cd /Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/LaundryERPLAAB

python3 manage.py makemigrations

python3 manage.py migrate

```

Si el archivo `manage.py` no existe en ese directorio, verifica que estés en el directorio correcto o que el archivo no haya sido movido o eliminado.

acunab: averigua los errores que tengo

GitHub Copilot: Para averiguar los errores que tienes, primero asegúrate de que estás en el directorio correcto y que el archivo `manage.py` existe. Luego, ejecuta los comandos de migración por separado para ver los mensajes de error específicos.

1. \*\*Verifica el directorio\*\*:

```sh

cd /Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/LaundryERPLAAB

ls

```

Asegúrate de que el archivo `manage.py` esté presente en este directorio.

2. \*\*Ejecuta las migraciones por separado\*\*:

```sh

python3 manage.py makemigrations

python3 manage.py migrate

```

Si el archivo `manage.py` no está en el directorio, verifica que estés en el directorio correcto o que el archivo no haya sido movido o eliminado.

Si los comandos anteriores no muestran errores específicos, revisa los siguientes puntos en tu configuración:

- \*\*Base de datos\*\*: Asegúrate de que la configuración de la base de datos en

settings.py

sea correcta y que la base de datos esté en funcionamiento.

- \*\*Aplicaciones instaladas\*\*: Verifica que todas las aplicaciones necesarias estén listadas en

INSTALLED\_APPS

en

settings.py

.

Si aún tienes problemas, por favor proporciona el mensaje de error específico que recibes al ejecutar los comandos. Esto ayudará a identificar el problema exacto.

acunab: manage está fuera del directorio

GitHub Copilot: Para solucionar el problema, asegúrate de que el archivo

manage.py

esté en el directorio raíz de tu proyecto Django. El archivo

manage.py

debe estar en el mismo nivel que el directorio que contiene `settings.py`.

Aquí tienes los pasos para mover

manage.py

al directorio correcto:

1. \*\*Mover

manage.py

al directorio raíz del proyecto\*\*:

```sh

mv /Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/LaundryERPLAAB/manage.py /Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/

```

2. \*\*Navegar al directorio raíz del proyecto\*\*:

```sh

cd /Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD

```

3. \*\*Ejecutar las migraciones\*\*:

```sh

python3 manage.py makemigrations

python3 manage.py migrate

```

Si

manage.py

ya está en el directorio correcto, asegúrate de que el archivo `settings.py` esté configurado correctamente y que todas las dependencias estén instaladas en tu entorno virtual.

acunab: revisame los errores de certificacionTD

GitHub Copilot: Para revisar los errores en tu proyecto

certificacionTD

, sigue estos pasos:

1. \*\*Verifica la estructura del proyecto\*\*:

Asegúrate de que la estructura de tu proyecto sea correcta. Debería verse algo así:

```

certificacionTD/

├── manage.py

├── lavanderia/

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── gestion/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── models.py

│ ├── tests.py

│ └── views.py

├── db.sqlite3

└── venv/

```

2. \*\*Activa el entorno virtual\*\*:

Asegúrate de que tu entorno virtual esté activado:

```sh

source venv/bin/activate

```

3. \*\*Instala las dependencias\*\*:

Asegúrate de que todas las dependencias estén instaladas:

```sh

pip install -r requirements.txt

```

4. \*\*Ejecuta las migraciones\*\*:

Navega al directorio raíz del proyecto y ejecuta las migraciones:

```sh

cd /Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD

python3 manage.py makemigrations

python3 manage.py migrate

```

5. \*\*Verifica la configuración de `settings.py`\*\*:

Asegúrate de que `settings.py` esté configurado correctamente, especialmente la configuración de la base de datos y las aplicaciones instaladas.

6. \*\*Ejecuta el servidor de desarrollo\*\*:

Inicia el servidor de desarrollo para ver si hay errores adicionales:

```sh

python3 manage.py runserver

```

Si encuentras algún error específico durante estos pasos, por favor proporciona el mensaje de error exacto para que pueda ayudarte a solucionarlo.

acunab: Migrations for 'gestion':

gestion/migrations/0002\_remove\_ordendeservicio\_cliente\_remove\_empleado\_user\_and\_more.py

- Remove field cliente from ordendeservicio

- Remove field user from empleado

- Remove field empleado from ordendeservicio

- Remove field orden\_servicio from venta

- Delete model Prenda

- Delete model Cliente

- Delete model Empleado

- Delete model OrdenDeServicio

- Delete model Venta

Operations to perform:

Apply all migrations: admin, auth, contenttypes, gestion, sessions

Running migrations:

Traceback (most recent call last):

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 103, in \_execute

return self.cursor.execute(sql)

~~~~~~~~~~~~~~~~~~~^^^^^

psycopg2.errors.InsufficientPrivilege: permission denied for schema public

LINE 1: CREATE TABLE "django\_migrations" ("id" bigint NOT NULL PRIMA...

^

The above exception was the direct cause of the following exception:

Traceback (most recent call last):

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/migrations/recorder.py", line 78, in ensure\_schema

editor.create\_model(self.Migration)

~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/base/schema.py", line 505, in create\_model

self.execute(sql, params or None)

~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/postgresql/schema.py", line 45, in execute

return super().execute(sql, params)

~~~~~~~~~~~~~~~^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/base/schema.py", line 202, in execute

cursor.execute(sql, params)

~~~~~~~~~~~~~~^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 122, in execute

return super().execute(sql, params)

~~~~~~~~~~~~~~~^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 79, in execute

return self.\_execute\_with\_wrappers(

~~~~~~~~~~~~~~~~~~~~~~~~~~~^

sql, params, many=False, executor=self.\_execute

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

)

^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 92, in \_execute\_with\_wrappers

return executor(sql, params, many, context)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 100, in \_execute

with self.db.wrap\_database\_errors:

^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/utils.py", line 91, in \_\_exit\_\_

raise dj\_exc\_value.with\_traceback(traceback) from exc\_value

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 103, in \_execute

return self.cursor.execute(sql)

~~~~~~~~~~~~~~~~~~~^^^^^

django.db.utils.ProgrammingError: permission denied for schema public

LINE 1: CREATE TABLE "django\_migrations" ("id" bigint NOT NULL PRIMA...

^

During handling of the above exception, another exception occurred:

Traceback (most recent call last):

File "/Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/manage.py", line 22, in <module>

main()

~~~~^^

File "/Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/manage.py", line 18, in main

execute\_from\_command\_line(sys.argv)

~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/\_\_init\_\_.py", line 442, in execute\_from\_command\_line

utility.execute()

~~~~~~~~~~~~~~~^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/\_\_init\_\_.py", line 436, in execute

self.fetch\_command(subcommand).run\_from\_argv(self.argv)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/base.py", line 413, in run\_from\_argv

self.execute(\*args, \*\*cmd\_options)

~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/base.py", line 459, in execute

output = self.handle(\*args, \*\*options)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/base.py", line 107, in wrapper

res = handle\_func(\*args, \*\*kwargs)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/commands/migrate.py", line 357, in handle

post\_migrate\_state = executor.migrate(

targets,

...<3 lines>...

fake\_initial=fake\_initial,

)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/migrations/executor.py", line 107, in migrate

self.recorder.ensure\_schema()

~~~~~~~~~~~~~~~~~~~~~~~~~~~^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/migrations/recorder.py", line 80, in ensure\_schema

raise MigrationSchemaMissing(

"Unable to create the django\_migrations table (%s)" % exc

)

django.db.migrations.exceptions.MigrationSchemaMissing: Unable to create the django\_migrations table (permission denied for schema public

LINE 1: CREATE TABLE "django\_migrations" ("id" bigint NOT NULL PRIMA...

GitHub Copilot: El error que estás viendo indica que el usuario de la base de datos no tiene los permisos necesarios para crear tablas en el esquema `public` de PostgreSQL. Para solucionar esto, debes asegurarte de que el usuario de la base de datos tenga los permisos adecuados.

Aquí tienes los pasos para otorgar los permisos necesarios:

1. \*\*Accede a PostgreSQL\*\*:

Abre una terminal y accede a PostgreSQL con un usuario que tenga permisos de superusuario (por ejemplo, `postgres`):

```sh

sudo -u postgres psql

```

2. \*\*Otorga permisos al usuario de la base de datos\*\*:

Una vez dentro de la consola de PostgreSQL, ejecuta los siguientes comandos para otorgar los permisos necesarios al usuario de la base de datos. Reemplaza `tu\_usuario` con el nombre de usuario de tu base de datos y `nombre\_de\_tu\_base\_de\_datos` con el nombre de tu base de datos:

```sql

GRANT ALL PRIVILEGES ON DATABASE nombre\_de\_tu\_base\_de\_datos TO tu\_usuario;

\c nombre\_de\_tu\_base\_de\_datos

GRANT ALL PRIVILEGES ON SCHEMA public TO tu\_usuario;

```

3. \*\*Sal de la consola de PostgreSQL\*\*:

```sql

\q

```

4. \*\*Vuelve a ejecutar las migraciones\*\*:

Ahora que el usuario de la base de datos tiene los permisos necesarios, vuelve a ejecutar las migraciones:

```sh

python3 manage.py makemigrations

python3 manage.py migrate

```

Estos pasos deberían resolver el problema de permisos y permitir que Django cree las tablas necesarias en la base de datos.

acunab: No changes detected

Operations to perform:

Apply all migrations: admin, auth, contenttypes, gestion, sessions

Running migrations:

Traceback (most recent call last):

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 103, in \_execute

return self.cursor.execute(sql)

~~~~~~~~~~~~~~~~~~~^^^^^

psycopg2.errors.InsufficientPrivilege: permission denied for schema public

LINE 1: CREATE TABLE "django\_migrations" ("id" bigint NOT NULL PRIMA...

^

The above exception was the direct cause of the following exception:

Traceback (most recent call last):

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/migrations/recorder.py", line 78, in ensure\_schema

editor.create\_model(self.Migration)

~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/base/schema.py", line 505, in create\_model

self.execute(sql, params or None)

~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/postgresql/schema.py", line 45, in execute

return super().execute(sql, params)

~~~~~~~~~~~~~~~^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/base/schema.py", line 202, in execute

cursor.execute(sql, params)

~~~~~~~~~~~~~~^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 122, in execute

return super().execute(sql, params)

~~~~~~~~~~~~~~~^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 79, in execute

return self.\_execute\_with\_wrappers(

~~~~~~~~~~~~~~~~~~~~~~~~~~~^

sql, params, many=False, executor=self.\_execute

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

)

^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 92, in \_execute\_with\_wrappers

return executor(sql, params, many, context)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 100, in \_execute

with self.db.wrap\_database\_errors:

^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/utils.py", line 91, in \_\_exit\_\_

raise dj\_exc\_value.with\_traceback(traceback) from exc\_value

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/backends/utils.py", line 103, in \_execute

return self.cursor.execute(sql)

~~~~~~~~~~~~~~~~~~~^^^^^

django.db.utils.ProgrammingError: permission denied for schema public

LINE 1: CREATE TABLE "django\_migrations" ("id" bigint NOT NULL PRIMA...

^

During handling of the above exception, another exception occurred:

Traceback (most recent call last):

File "/Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/manage.py", line 22, in <module>

main()

~~~~^^

File "/Users/luisacuna/Library/CloudStorage/OneDrive-Personal/BOOTCAMP/html/Bootcamp\_Python/certificacionTD/manage.py", line 18, in main

execute\_from\_command\_line(sys.argv)

~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/\_\_init\_\_.py", line 442, in execute\_from\_command\_line

utility.execute()

~~~~~~~~~~~~~~~^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/\_\_init\_\_.py", line 436, in execute

self.fetch\_command(subcommand).run\_from\_argv(self.argv)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/base.py", line 413, in run\_from\_argv

self.execute(\*args, \*\*cmd\_options)

~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/base.py", line 459, in execute

output = self.handle(\*args, \*\*options)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/base.py", line 107, in wrapper

res = handle\_func(\*args, \*\*kwargs)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/core/management/commands/migrate.py", line 357, in handle

post\_migrate\_state = executor.migrate(

targets,

...<3 lines>...

fake\_initial=fake\_initial,

)

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/migrations/executor.py", line 107, in migrate

self.recorder.ensure\_schema()

~~~~~~~~~~~~~~~~~~~~~~~~~~~^^

File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/django/db/migrations/recorder.py", line 80, in ensure\_schema

raise MigrationSchemaMissing(

"Unable to create the django\_migrations table (%s)" % exc

)

django.db.migrations.exceptions.MigrationSchemaMissing: Unable to create the django\_migrations table (permission denied for schema public

LINE 1: CREATE TABLE "django\_migrations" ("id" bigint NOT NULL PRIMA...

GitHub Copilot: El error indica que el usuario de la base de datos no tiene los permisos necesarios para crear tablas en el esquema `public` de PostgreSQL. Para solucionar esto, debes asegurarte de que el usuario de la base de datos tenga los permisos adecuados.

Aquí tienes los pasos para otorgar los permisos necesarios:

1. \*\*Accede a PostgreSQL\*\*:

Abre una terminal y accede a PostgreSQL con un usuario que tenga permisos de superusuario (por ejemplo, `postgres`):

```sh

sudo -u postgres psql

```

2. \*\*Otorga permisos al usuario de la base de datos\*\*:

Una vez dentro de la consola de PostgreSQL, ejecuta los siguientes comandos para otorgar los permisos necesarios al usuario de la base de datos. Reemplaza `tu\_usuario` con el nombre de usuario de tu base de datos y `nombre\_de\_tu\_base\_de\_datos` con el nombre de tu base de datos:

```sql

GRANT ALL PRIVILEGES ON DATABASE nombre\_de\_tu\_base\_de\_datos TO tu\_usuario;

\c nombre\_de\_tu\_base\_de\_datos

GRANT ALL PRIVILEGES ON SCHEMA public TO tu\_usuario;

```

3. \*\*Sal de la consola de PostgreSQL\*\*:

```sql

\q

```

4. \*\*Vuelve a ejecutar las migraciones\*\*:

Ahora que el usuario de la base de datos tiene los permisos necesarios, vuelve a ejecutar las migraciones:

```sh

python3 manage.py makemigrations

python3 manage.py migrate

```

Estos pasos deberían resolver el problema de permisos y permitir que Django cree las tablas necesarias en la base de datos.

acunab: Cree un proyecto en Django con el nombre de practica\_orm, usando un ambiente virtual de

nombre practica\_orm\_django.

4. Conecte la base de datos al proyecto de Django.

5. Verifique que funcionen las migraciones iniciales, y ejecute el servidor Web de Django

GitHub Copilot: Para crear un proyecto Django llamado `practica\_orm`, configurar un entorno virtual, conectar la base de datos y verificar las migraciones iniciales, sigue estos pasos:

1. \*\*Crear el entorno virtual\*\*:

```sh

python3 -m venv practica\_orm\_django

source practica\_orm\_django/bin/activate

```

2. \*\*Instalar Django\*\*:

```sh

pip install django psycopg2-binary

```

3. \*\*Crear el proyecto Django\*\*:

```sh

django-admin startproject practica\_orm

cd practica\_orm

```

4. \*\*Configurar la base de datos en `practica\_orm/settings.py`\*\*:

Abre el archivo `practica\_orm/settings.py` y configura la base de datos PostgreSQL:

```python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'nombre\_de\_tu\_base\_de\_datos',

'USER': 'tu\_usuario',

'PASSWORD': 'tu\_contraseña',

'HOST': 'localhost',

'PORT': '5432',

}

}

```

5. \*\*Verificar y ejecutar las migraciones iniciales\*\*:

```sh

python3 manage.py makemigrations

python3 manage.py migrate

```

6. \*\*Ejecutar el servidor web de Django\*\*:

```sh

python3 manage.py runserver

```

Estos pasos te permitirán crear un proyecto Django, configurar la base de datos, verificar las migraciones iniciales y ejecutar el servidor web. Asegúrate de que la base de datos PostgreSQL esté en funcionamiento y que el usuario tenga los permisos necesarios para crear y modificar tablas.

acunab: @terminal /explain #terminalSelection

Terminal: El error que estás viendo indica que el usuario de la base de datos `user\_db` no tiene los permisos necesarios para crear tablas en el esquema `public`. Para solucionar esto, debes otorgar los permisos adecuados al usuario en PostgreSQL.

Sigue estos pasos:

1. Abre una terminal y accede a PostgreSQL como un usuario con privilegios suficientes (por ejemplo, el usuario `postgres`):

```bash

psql -U postgres

```

2. Otorga los permisos necesarios al usuario `user\_db`:

```sql

GRANT ALL PRIVILEGES ON DATABASE db\_practica\_orm TO user\_db;

GRANT ALL PRIVILEGES ON SCHEMA public TO user\_db;

```

3. Sal de PostgreSQL:

```sql

\q

```

Después de otorgar los permisos, intenta ejecutar nuevamente las migraciones:

```bash

python3 manage.py migrate

```

acunab: @terminal /explain #terminalSelection

Terminal: El error indica que el usuario `user\_db` no tiene permisos suficientes para crear tablas en el esquema `public` de la base de datos `db\_practica\_orm`. Para solucionar esto, debes otorgar los permisos necesarios al usuario en PostgreSQL.

Sigue estos pasos:

1. Abre una terminal y accede a PostgreSQL como un usuario con privilegios suficientes (por ejemplo, el usuario `postgres`):

```bash

psql -U postgres

```

2. Otorga los permisos necesarios al usuario `user\_db`:

```sql

GRANT ALL PRIVILEGES ON DATABASE db\_practica\_orm TO user\_db;

GRANT ALL PRIVILEGES ON SCHEMA public TO user\_db;

```

3. Sal de PostgreSQL:

```sql

\q

```

Después de otorgar los permisos, intenta ejecutar nuevamente las migraciones:

```bash

python3 manage.py migrate

```

acunab: ya lo hice y no funciona

GitHub Copilot: Si ya otorgaste los permisos necesarios y aún no funciona, asegúrate de que los cambios se hayan aplicado correctamente y que el usuario `user\_db` tenga los permisos adecuados. Aquí hay algunos pasos adicionales que puedes seguir para solucionar el problema:

1. \*\*Verifica los permisos del usuario\*\*:

Asegúrate de que el usuario `user\_db` tenga los permisos necesarios en la base de datos y el esquema `public`.

Abre una terminal y accede a PostgreSQL como un usuario con privilegios suficientes (por ejemplo, el usuario `postgres`):

```bash

psql -U postgres

```

Luego, verifica los permisos del usuario `user\_db`:

```sql

\c db\_practica\_orm

\dn+

```

Esto debería mostrar los esquemas y sus propietarios. Asegúrate de que el usuario `user\_db` tenga permisos en el esquema `public`.

2. \*\*Otorga permisos explícitos\*\*:

Si los permisos no se han aplicado correctamente, otórgalos explícitamente:

```sql

GRANT ALL PRIVILEGES ON DATABASE db\_practica\_orm TO user\_db;

GRANT ALL PRIVILEGES ON SCHEMA public TO user\_db;

ALTER USER user\_db WITH SUPERUSER;

```

3. \*\*Reinicia el servidor PostgreSQL\*\*:

A veces, los cambios en los permisos pueden requerir un reinicio del servidor PostgreSQL para que surtan efecto. Reinicia el servidor PostgreSQL:

```bash

sudo service postgresql restart

```

4. \*\*Verifica la conexión desde Django\*\*:

Asegúrate de que Django pueda conectarse a la base de datos. Puedes hacer esto ejecutando un simple script de prueba en Django:

```python

python3 manage.py dbshell

```

Si puedes acceder a la base de datos desde el shell de Django, significa que la conexión está funcionando correctamente.

5. \*\*Ejecuta las migraciones nuevamente\*\*:

Después de verificar los permisos y la conexión, intenta ejecutar las migraciones nuevamente:

```bash

python3 manage.py makemigrations

python3 manage.py migrate

```

Si después de seguir estos pasos aún tienes problemas, por favor proporciona cualquier mensaje de error adicional que recibas para que pueda ayudarte a identificar el problema exacto.

acunab: ahora tengo que levantar el servidor

GitHub Copilot: Para levantar el servidor de desarrollo de Django, sigue estos pasos:

1. \*\*Asegúrate de que el entorno virtual esté activado\*\*:

```sh

source practica\_orm\_django/bin/activate

```

2. \*\*Ejecuta el servidor de desarrollo\*\*:

Navega al directorio raíz de tu proyecto y ejecuta el siguiente comando:

```sh

python3 manage.py runserver

```

Esto iniciará el servidor de desarrollo de Django y deberías ver una salida similar a esta:

```sh

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

October 10, 2023 - 10:00:00

Django version 3.2, using settings 'practica\_orm.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

```

Abre tu navegador web y navega a `http://127.0.0.1:8000/` para ver tu aplicación Django en funcionamiento.

acunab: podrías documentar todo lo que hice en "practica\_orm\_django?

GitHub Copilot: Claro, aquí tienes una documentación detallada de los pasos que seguiste para configurar el proyecto

practica\_orm\_django

:

# Documentación del Proyecto

practica\_orm\_django

## 1. Crear el entorno virtual

Primero, creamos un entorno virtual para aislar las dependencias del proyecto.

```sh

python3 -m venv practica\_orm\_django

source practica\_orm\_django/bin/activate

```

## 2. Instalar Django y otras dependencias

Instalamos Django y el conector de PostgreSQL.

```sh

pip install django psycopg2-binary

```

## 3. Crear el proyecto Django

Creamos un nuevo proyecto Django llamado `practica\_orm`.

```sh

django-admin startproject practica\_orm

cd practica\_orm

```

## 4. Configurar la base de datos

Editamos el archivo

settings.py

para configurar la base de datos PostgreSQL.

```python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'db\_practica\_orm',

'

USER

': 'user\_db',

'PASSWORD': 'user\_db',

'HOST': 'localhost',

'PORT': '5432',

}

}

```

## 5. Otorgar permisos en PostgreSQL

Accedemos a PostgreSQL y otorgamos los permisos necesarios al usuario `user\_db`.

```sh

psql -U postgres

```

```sql

GRANT ALL PRIVILEGES ON DATABASE db\_practica\_orm TO user\_db;

GRANT ALL PRIVILEGES ON SCHEMA public TO user\_db;

ALTER USER user\_db WITH SUPERUSER;

\q

```

## 6. Realizar las migraciones iniciales

Ejecutamos las migraciones iniciales para crear las tablas necesarias en la base de datos.

```sh

python3 manage.py makemigrations

python3 manage.py migrate

```

## 7. Crear un superusuario

Creamos un superusuario para acceder al panel de administración de Django.

```sh

python3 manage.py createsuperuser

```

## 8. Ejecutar el servidor de desarrollo

Iniciamos el servidor de desarrollo de Django.

```sh

python3 manage.py runserver

```

## 9. Acceder a la aplicación

Abrimos un navegador web y navegamos a `http://127.0.0.1:8000/` para ver la aplicación en funcionamiento.

## 10. Estructura del proyecto

La estructura del proyecto debería verse algo así:

```

practica\_orm\_django/

├── practica\_orm/

│ ├── \_\_init\_\_.py

│ ├── asgi.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── manage.py

├── db.sqlite3

└── practica\_orm\_django/

├── \_\_init\_\_.py

├── admin.py

├── apps.py

├── models.py

├── tests.py

└── views.py

```

Con estos pasos, has configurado exitosamente un proyecto Django con PostgreSQL como base de datos y has verificado que las migraciones iniciales funcionan correctamente.

Código similar encontrado con 1 tipo de licencia

acunab: Diseñar una aplicación que se va a llamar “productos del proyecto”.

2. Los campos de los productos son:

• nombre: nombre del producto.

• precio: precio del producto.

• descripcion: descripción del producto.

3. Generar las migraciones respectivas.

4. Usar el comando en Django que muestra el SQL que se ejecuta al realizar las migraciones y

para crear el modelo.

5. Consultar la base de datos y el modelo creado en la shell de postgresql

GitHub Copilot: Para diseñar una aplicación llamada "productos" en tu proyecto Django, sigue estos pasos:

## 1. Crear la aplicación "productos"

Primero, crea la aplicación dentro de tu proyecto Django.

```sh

python3 manage.py startapp productos

```

## 2. Definir el modelo de Producto

Edita el archivo `productos/models.py` para definir el modelo `Producto` con los campos especificados.

```python

from django.db import models

class Producto(models.Model):

nombre = models.CharField(max\_length=255)

precio = models.DecimalField(max\_digits=10, decimal\_places=2)

descripcion = models.TextField()

def \_\_str\_\_(self):

return self.nombre

```

## 3. Registrar la aplicación en

settings.py

Agrega la aplicación `productos` a la lista de aplicaciones instaladas en

settings.py

.

```python

INSTALLED\_APPS = [

...

'productos',

]

```

## 4. Generar las migraciones

Crea las migraciones para el modelo `Producto`.

```sh

python3 manage.py makemigrations productos

```

## 5. Mostrar el SQL que se ejecuta al realizar las migraciones

Usa el comando `sqlmigrate` para mostrar el SQL que se ejecutará al aplicar las migraciones.

```sh

python3 manage.py sqlmigrate productos 0001

```

## 6. Aplicar las migraciones

Aplica las migraciones para crear la tabla `Producto` en la base de datos.

```sh

python3 manage.py migrate

```

## 7. Consultar la base de datos y el modelo creado en la shell de PostgreSQL

Accede a la shell de PostgreSQL para consultar la base de datos y verificar que la tabla `productos\_producto` se haya creado correctamente.

```sh

psql -U user\_db -d db\_practica\_orm

```

Dentro de la shell de PostgreSQL, puedes usar el siguiente comando para listar las tablas y verificar que la tabla `productos\_producto` exista:

```sql

\dt

```

Para consultar los datos de la tabla `productos\_producto`, usa:

```sql

SELECT \* FROM productos\_producto;

```

Sal de la shell de PostgreSQL:

```sql

\q

```

## 8. Consultar el modelo en la shell de Django

Accede a la shell de Django para interactuar con el modelo `Producto`.

```sh

python3 manage.py shell

```

Dentro de la shell de Django, puedes crear y consultar instancias del modelo `Producto`:

```python

from productos.models import Producto

# Crear un nuevo producto

producto = Producto(nombre="Producto 1", precio=100.00, descripcion="Descripción del Producto 1")

producto.save()

# Consultar todos los productos

productos = Producto.objects.all()

for p in productos:

print(p.nombre, p.precio, p.descripcion)

```

Sal de la shell de Django:

```python

exit()

```

Con estos pasos, has creado una aplicación llamada "productos", definido el modelo `Producto`, generado y aplicado las migraciones, mostrado el SQL correspondiente, y consultado la base de datos y el modelo creado.

acunab: Vamos a gestionar datos en la base por medio de un CRUD básico dentro del framework Django.

Tomando como base el desarrollo del Drilling, procederemos a desarrollar los siguientes puntos:

1. Agregar campos desde la consola o shell de django al modelo de Producto.

2. Listar los productos desde la consola de psql.

3. Consultar todos los productos desde la consola de Python.

GitHub Copilot: Para gestionar datos en la base de datos mediante un CRUD básico en Django, sigue estos pasos:

## 1. Agregar campos desde la consola o shell de Django al modelo de Producto

Primero, accede a la shell de Django para interactuar con el modelo `Producto`.

```sh

python3 manage.py shell

```

Dentro de la shell de Django, puedes agregar nuevos productos al modelo `Producto`:

```python

from productos.models import Producto

# Crear un nuevo producto

producto1 = Producto(nombre="Producto 1", precio=100.00, descripcion="Descripción del Producto 1")

producto1.save()

producto2 = Producto(nombre="Producto 2", precio=150.00, descripcion="Descripción del Producto 2")

producto2.save()

producto3 = Producto(nombre="Producto 3", precio=200.00, descripcion="Descripción del Producto 3")

producto3.save()

```

Sal de la shell de Django:

```python

exit()

```

## 2. Listar los productos desde la consola de psql

Accede a la shell de PostgreSQL para consultar los datos de la tabla `productos\_producto`.

```sh

psql -U user\_db -d db\_practica\_orm

```

Dentro de la shell de PostgreSQL, usa el siguiente comando para listar todos los productos:

```sql

SELECT \* FROM productos\_producto;

```

Sal de la shell de PostgreSQL:

```sql

\q

```

## 3. Consultar todos los productos desde la consola de Python

Accede nuevamente a la shell de Django para consultar todos los productos:

```sh

python3 manage.py shell

```

Dentro de la shell de Django, puedes consultar todos los productos:

```python

from productos.models import Producto

# Consultar todos los productos

productos = Producto.objects.all()

for p in productos:

print(f"Nombre: {p.nombre}, Precio: {p.precio}, Descripción: {p.descripcion}")

```

Sal de la shell de Django:

```python

exit()

```

Con estos pasos, has agregado campos al modelo `Producto` desde la shell de Django, listado los productos desde la consola de PostgreSQL y consultado todos los productos desde la consola de Python. Esto te permite gestionar datos en la base de datos mediante un CRUD básico en Django.

acunab: desarrolla lo del pdf, es sobre este mismo desarrollo

GitHub Copilot: Para desarrollar un CRUD básico en Django para gestionar los datos del modelo `Producto`, sigue estos pasos:

## 1. Crear la aplicación "productos"

Primero, crea la aplicación dentro de tu proyecto Django.

```sh

python3 manage.py startapp productos

```

## 2. Definir el modelo de Producto

Edita el archivo `productos/models.py` para definir el modelo `Producto` con los campos especificados.

```python

from django.db import models

class Producto(models.Model):

nombre = models.CharField(max\_length=255)

precio = models.DecimalField(max\_digits=10, decimal\_places=2)

descripcion = models.TextField()

def \_\_str\_\_(self):

return self.nombre

```

## 3. Registrar la aplicación en

settings.py

Agrega la aplicación `productos` a la lista de aplicaciones instaladas en

settings.py

.

```python

INSTALLED\_APPS = [

...

'productos',

]

```

## 4. Generar las migraciones

Crea las migraciones para el modelo `Producto`.

```sh

python3 manage.py makemigrations productos

```

## 5. Mostrar el SQL que se ejecuta al realizar las migraciones

Usa el comando `sqlmigrate` para mostrar el SQL que se ejecutará al aplicar las migraciones.

```sh

python3 manage.py sqlmigrate productos 0001

```

## 6. Aplicar las migraciones

Aplica las migraciones para crear la tabla `Producto` en la base de datos.

```sh

python3 manage.py migrate

```

## 7. Crear vistas para el CRUD

Edita el archivo `productos/views.py` para crear las vistas necesarias para el CRUD.

```python

from django.shortcuts import render, get\_object\_or\_404, redirect

from .models import Producto

from .forms import ProductoForm

def producto\_list(request):

productos = Producto.objects.all()

return render(request, 'productos/producto\_list.html', {'productos': productos})

def producto\_detail(request, pk):

producto = get\_object\_or\_404(Producto, pk=pk)

return render(request, 'productos/producto\_detail.html', {'producto': producto})

def producto\_new(request):

if request.method == "POST":

form = ProductoForm(request.POST)

if form.is\_valid():

producto = form.save(commit=False)

producto.save()

return redirect('producto\_detail', pk=producto.pk)

else:

form = ProductoForm()

return render(request, 'productos/producto\_edit.html', {'form': form})

def producto\_edit(request, pk):

producto = get\_object\_or\_404(Producto, pk=pk)

if request.method == "POST":

form = ProductoForm(request.POST, instance=producto)

if form.is\_valid():

producto = form.save(commit=False)

producto.save()

return redirect('producto\_detail', pk=producto.pk)

else:

form = ProductoForm(instance=producto)

return render(request, 'productos/producto\_edit.html', {'form': form})

def producto\_delete(request, pk):

producto = get\_object\_or\_404(Producto, pk=pk)

producto.delete()

return redirect('producto\_list')

```

## 8. Crear formularios para el modelo

Crea un archivo `productos/forms.py` y define el formulario para el modelo `Producto`.

```python

from django import forms

from .models import Producto

class ProductoForm(forms.ModelForm):

class Meta:

model = Producto

fields = ['nombre', 'precio', 'descripcion']

```

## 9. Configurar las URLs

Edita el archivo `productos/urls.py` para definir las rutas para las vistas del CRUD.

```python

from django.urls import path

from . import views

urlpatterns = [

path('', views.producto\_list, name='producto\_list'),

path('producto/<int:pk>/', views.producto\_detail, name='producto\_detail'),

path('producto/new/', views.producto\_new, name='producto\_new'),

path('producto/<int:pk>/edit/', views.producto\_edit, name='producto\_edit'),

path('producto/<int:pk>/delete/', views.producto\_delete, name='producto\_delete'),

]

```

Edita el archivo `practica\_orm/urls.py` para incluir las URLs de la aplicación `productos`.

```python

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('productos/', include('productos.urls')),

]

```

## 10. Crear plantillas para las vistas

Crea un directorio `templates/productos` y dentro de él, crea las plantillas `producto\_list.html`, `producto\_detail.html`, y `producto\_edit.html`.

### `producto\_list.html`

```html

<!DOCTYPE html>

<html>

<head>

<title>Lista de Productos</title>

</head>

<body>

<h1>Lista de Productos</h1>

<a href="{% url 'producto\_new' %}">Nuevo Producto</a>

<ul>

{% for producto in productos %}

<li>

<a href="{% url 'producto\_detail' producto.pk %}">{{ producto.nombre }}</a>

<a href="{% url 'producto\_edit' producto.pk %}">Editar</a>

<a href="{% url 'producto\_delete' producto.pk %}">Eliminar</a>

</li>

{% endfor %}

</ul>

</body>

</html>

```

### `producto\_detail.html`

```html

<!DOCTYPE html>

<html>

<head>

<title>Detalle del Producto</title>

</head>

<body>

<h1>{{ producto.nombre }}</h1>

<p>Precio: {{ producto.precio }}</p>

<p>Descripción: {{ producto.descripcion }}</p>

<a href="{% url 'producto\_edit' producto.pk %}">Editar</a>

<a href="{% url 'producto\_list' %}">Volver a la lista</a>

</body>

</html>

```

### `producto\_edit.html`

```html

<!DOCTYPE html>

<html>

<head>

<title>Editar Producto</title>

</head>

<body>

<h1>Editar Producto</h1>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Guardar</button>

</form>

<a href="{% url 'producto\_list' %}">Cancelar</a>

</body>

</html>

```

## 11. Ejecutar el servidor de desarrollo

Inicia el servidor de desarrollo de Django.

```sh

python3 manage.py runserver

```

## 12. Acceder a la aplicación

Abre un navegador web y navega a `http://127.0.0.1:8000/productos/` para ver la lista de productos y gestionar los datos mediante el CRUD básico.

Con estos pasos, has desarrollado un CRUD básico en Django para gestionar los datos del modelo `Producto`.

Código similar encontrado con 1 tipo de licencia

acunab: los registros deben estar como tabla

GitHub Copilot: Para mostrar los registros en una tabla en la plantilla `producto\_list.html`, puedes modificar el código HTML para usar una tabla en lugar de una lista. Aquí tienes el código actualizado:

### `producto\_list.html`

```html

<!DOCTYPE html>

<html>

<head>

<title>Lista de Productos</title>

</head>

<body>

<h1>Lista de Productos</h1>

<a href="{% url 'producto\_new' %}">Nuevo Producto</a>

<table border="1">

<thead>

<tr>

<th>Nombre</th>

<th>Precio</th>

<th>Descripción</th>

<th>Acciones</th>

</tr>

</thead>

<tbody>

{% for producto in productos %}

<tr>

<td><a href="{% url 'producto\_detail' producto.pk %}">{{ producto.nombre }}</a></td>

<td>{{ producto.precio }}</td>

<td>{{ producto.descripcion }}</td>

<td>

<a href="{% url 'producto\_edit' producto.pk %}">Editar</a>

<a href="{% url 'producto\_delete' producto.pk %}">Eliminar</a>

</td>

</tr>

{% endfor %}

</tbody>

</table>

</body>

</html>

```

Este código crea una tabla HTML con encabezados para los campos `Nombre`, `Precio`, `Descripción` y `Acciones`. Cada producto se muestra en una fila de la tabla, y las acciones de `Editar` y `Eliminar` están disponibles en la última columna.

Con esta modificación, los registros de productos se mostrarán en una tabla en la página de lista de productos.

