Tutorial: Hola Mundo con Django + MySQL para ÁgoraUN

1. Lenguaje y Framework Seleccionado

En este tutorial construiremos una aplicación "Hola Mundo" usando Django como framework web y MySQL como sistema de gestión de bases de datos. La idea es mostrar paso a paso desde la infraestructura (Docker) hasta la entidad, la vista y la plantilla que muestran la información.

Lenguaje y versiones recomendadas

• Lenguaje: Python 3.9 (compatible con Django 4.2).
• Framework: Django 4.2 (LTS en la serie 4.x).
• Base de datos: MySQL 8.0.
• Containerización: Docker + Docker Compose.

Nota: puedes ajustar las versiones según tus necesidades; aquí usamos estas versiones por estabilidad y compatibilidad con paquetes comunes.

2. Librerías de ORM Utilizadas

En este proyecto se usan:
• Django ORM: el ORM integrado en Django que facilita la definición de modelos y consultas.
• PyMySQL: conector para que Django se comunique con MySQL (alternativa a mysqlclient cuando no deseas compilar dependencias nativas).
• cryptography: cuando tu configuración de MySQL requiere cifrado o autenticación más segura (opcional según despliegue).

Archivo requirements.txt sugerido (líneas):

requirements.txt

```
Django>=4.2,<5
pymysql>=1.0
cryptography>=3.0
```

3. Configuración y Levantamiento de la Base de Datos

A continuación se muestra una configuración mínima de docker-compose para levantar MySQL y el servicio web que ejecuta Django. Explicaciones inline ayudan a entender cada bloque.

3.1. Docker Compose (docker-compose.yml)

Archivo docker-compose.yml: copia y pega en la raíz del proyecto. Este compose crea un servicio 'db' con MySQL y un servicio 'web' que construye el backend. Se expone el puerto 8000 para la aplicación Django y 3306 para MySQL (útil para pruebas locales).

docker-compose.yml (contenido)

None

```
services:
  db:
    image: mysql:8.0
    container_name: hm_mysql
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: holamundo_db
      MYSQL_USER: hmuser
      MYSQL_PASSWORD: hmpass
    ports:
      - "3306:3306"
    volumes:
      - db_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  web:
    build: ./backend
    container_name: hm_web
    command: bash -c "python manage.py migrate && python manage.py
runserver 0.0.0.0:8000"
    volumes:
      - ./backend:/code
    ports:
      - "8000:8000"
    environment:
      - DB_HOST=db
      - DB_NAME=holamundo_db
      - DB_USER=hmuser
      - DB_PASSWORD=hmpass
    depends_on:
```

```
        db:
          condition: service_healthy

  volumes:
    db_data:
```

Explicación breve:
• `image: mysql:8.0` usa la imagen oficial de MySQL 8.
• Variables de entorno configuran la base de datos inicial.
• `healthcheck` permite que el contenedor 'web' espere hasta que la base de datos esté lista.
• `command` en web aplica migraciones y levanta el servidor durante el build/run.

3.2. Configuración Django (settings.py)

Configura la sección DATABASES en settings.py para usar MySQL. Usamos variables de entorno porque así es más sencillo cambiar credenciales en distintos entornos sin editar código fuente.

DATABASES (fragmento)

Python

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': os.environ.get('DB_NAME', 'holamundo_db'),
        'USER': os.environ.get('DB_USER', 'hmuser'),
        'PASSWORD': os.environ.get('DB_PASSWORD', 'hmpass'),
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET
sql_mode='STRICT_TRANS_TABLES'"
        }
    }
}
```

- Consejo: revisa que la librería PyMySQL esté instalada y que en `backend/project/__init__.py` se haya agregado `pymysql.install_as_MySQLdb()` para compatibilidad cuando no uses mysqlclient.

4. Ejecución del Hola Mundo Completo

En esta sección veremos la conexión, el modelo (entidad) y la parte visual (vista + template). Mantuvimos la estructura original pero ampliamos explicaciones.

4.1. Conexión a la Base de Datos

Archivo de inicialización (backend/project/__init__.py):

```python
import pymysql
pymysql.install_as_MySQLdb()
```

Esto indica a Django que use PyMySQL como reemplazo de MySQLdb, evitando la necesidad de compilar extensiones nativas.

4.2. Instanciación de Entidades - Modelo Grupo

Se define el modelo `Grupo` manteniendo los nombres de columnas que podrían corresponder a una base de datos preexistente. Explicamos cada campo para mayor claridad.

Modelo Django (backend/grupos/models.py)

```python
from django.db import models

class Grupo(models.Model):
    id_grupo = models.IntegerField(primary_key=True, db_column='ID_GRUPO')
    nombre_grupo = models.CharField(max_length=60, db_column='NOMBRE_GRUPO')
    area_interes = models.CharField(max_length=40, db_column='AREA_INTERES', default='Cultura')
    fecha_creacion = models.DateField(db_column='FECHA_CREACION', auto_now_add=True)
    tipo_grupo = models.CharField(max_length=40, db_column='TIPO_GRUPO', default='Club')
    logo = models.BinaryField(db_column='LOGO', null=True, blank=True)
```

```
    correo_grupo = models.CharField(max_length=128,
db_column='CORREO_GRUPO')
    descripcion = models.TextField(db_column='DESCRIPCION')
    link_whatsapp = models.CharField(max_length=128,
db_column='LINK_WHATSAPP', blank=True, null=True)

    class Meta:
        db_table = 'GRUPO'
        verbose_name = 'Grupo'
        verbose_name_plural = 'Grupos'

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

Explicación de campos:
• `id_grupo`: clave primaria que usa un nombre de columna específico para compatibilidad con esquemas existentes.
• `nombre_grupo`: nombre representativo del grupo.
• `area_interes`: área o temática del grupo (p. ej. Tecnología, Cultura).
• `fecha_creacion`: fecha automática de creación (`auto_now_add=True`).
• `logo`: campo binario — útil si se desea guardar imágenes directamente en la base de datos (no recomendado para producción; mejor usar almacenamiento de objetos o filesystem).
• `link_whatsapp`: opcional, permite mantener un enlace de contacto.

4.3. Componente Visual - Vista y Template

La vista obtiene una instancia del modelo y la envía al template que renderiza la información. Mostramos el código y explicamos la ruta y template.

Vista Django (backend/grupos/views.py)

Python

```
from django.shortcuts import render, get_object_or_404
from .models import Grupo

def grupo_detail(request, pk=1):
    grupo = get_object_or_404(Grupo, pk=pk)
    return render(request, "grupos/grupo_detail.html", {"grupo":
grupo})
```

Explicación: `get_object_or_404` devuelve la instancia o lanza 404 si no existe. La vista pasa el contexto `{'grupo': grupo}` al template.

Template HTML (backend/grupos/templates/grupos/grupo_detail.html)

HTML

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Hola Mundo - Grupo</title>
</head>
<body>
  <h1>Hola Mundo — Grupo</h1>
  <h2>{{ grupo.nombre_grupo }}</h2>
  <p><strong>Descripción:</strong> {{ grupo.descripcion|linebreaks }}</p>
  <p><strong>Correo:</strong> {{ grupo.correo_grupo }}</p>

  <hr>
  <p>Para ver otro grupo: <a href="{% url 'grupo-detail' pk=2 %}">Grupo 2 (ejemplo)</a></p>
</body>
</html>
```

Consejos: valida que la ruta `grupo-detail` esté definida y que en `INSTALLED_APPS` tengas agregada la app `grupos`.

Configuración de URLs (backend/project/urls.py)

Python

```
from django.contrib import admin
from django.urls import path
from grupos.views import grupo_detail

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', grupo_detail, name='home'),
    path('grupo/<int:pk>/', grupo_detail, name='grupo-detail'),
```

```
]
```

Nota: la ruta raíz muestra el detalle del grupo por defecto (pk=1). Puedes crear vistas adicionales según necesites.

5. Pasos de Ejecución Completos

Resumen paso a paso para dejar la aplicación funcionando en tu máquina (o entorno de pruebas). Mantuvimos las mismas subsecciones que en el documento original.

5.1. Levantar la Infraestructura

Con Docker y docker-compose configurados, ejecuta:

Shell

```
docker-compose up --build -d
```

Esto construirá la imagen del backend (según Dockerfile en `backend`) y levantará los contenedores en segundo plano. Revisa logs con `docker-compose logs -f web` para ver el estado.

5.2. Configurar la Base de Datos

Aplica migraciones para crear las tablas necesarias en MySQL:

Shell

```
# Aplicar migraciones de Django
docker-compose exec web python manage.py makemigrations
docker-compose exec web python manage.py migrate
```

Si ya tienes una base de datos con esquema preexistente, podrías usar `inspectdb` o ajustar los modelos para que concuerden con las tablas existentes.

5.3. Crear Datos de Prueba

Para poblar datos de ejemplo, accede al shell de Django y crea una instancia del modelo:

Shell

```
# Acceder al shell de Django
docker-compose exec web python manage.py shell
```

Dentro del shell:

Python

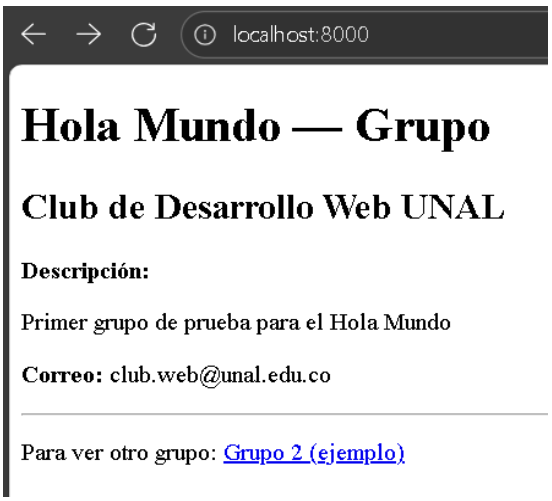
```
from grupos.models import Grupo
from datetime import date

# Crear instancia de la entidad Grupo
grupo = Grupo(
    id_grupo=1,
    nombre_grupo="Club de Desarrollo Web UNAL",
    area_interes="Tecnología",
    fecha_creacion=date.today(),
    tipo_grupo="Semillero",
    correo_grupo="club.web@unal.edu.co",
    descripcion="Primer grupo de prueba para el Hola Mundo"
)

# Guardar en la base de datos
grupo.save()
print("¡Grupo creado exitosamente!")
exit()
```

Consejo: si `fecha_creacion` tiene `auto_now_add=True` puedes omitir pasar `fecha_creacion` al crear la instancia.

5.4. Verificar el Funcionamiento

Abre en el navegador: http://localhost:8000 y deberías ver:

• Título "Hola Mundo — Grupo"
• Nombre del grupo: "Club de Desarrollo Web UNAL"
• Descripción del grupo
• Correo electrónico del grupo
• Enlace para navegar a otros grupos



- Si algo no aparece, revisa logs de `web` y la conexión a la base de datos (credenciales, host, puertos).