

#

DESARROLLO WEB FULLSTACK

con Python y JavaScript

polotic
misiones



Relaciones entre Modelos



Relaciones entre Modelos



- ❖ Este es un buen comienzo, pero pensando en lo anterior, no queremos tener que almacenar el nombre de la ciudad como origen y destino para cada vuelo, por lo que probablemente queramos otro modelo para un aeropuerto que de alguna manera esté relacionado con el modelo de vuelo. :

```
class Aeropuerto(models.Model):
    codigo = models.CharField(max_length=3)
    ciudad = models.CharField(max_length=64)

    def __str__(self):
        return f"{self.ciudad} ({self.codigo})"
```

```
class Vuelo(models.Model):
    origen = models.ForeignKey(Aeropuerto, on_delete=models.CASCADE,
                              related_name="salidas")
    destino = models.ForeignKey(Aeropuerto, on_delete=models.CASCADE,
                               related_name="arribos")
    duracion = models.IntegerField()

    def __str__(self):
        return f"{self.id}: {self.origen } a {self.destino}"
```



Relaciones entre Modelos



Ya hemos visto todo en nuestra nueva clase Aeropuerto antes, pero los cambios en los campos de origen y destino dentro de la clase de Vuelo son nuevos para nosotros:

- ❖ Especificamos que los campos de origen y destino son claves foráneas, lo que significa que se refieren a otro objeto.
- ❖ Al ingresar Aeropuerto como nuestro primer argumento, estamos especificando el tipo de objeto al que se refiere este campo.
- ❖ El siguiente argumento, `on_delete = models.CASCADE` da instrucciones sobre lo que debería suceder si se elimina un aeropuerto. En este caso, especificamos que cuando se elimina un aeropuerto, también se deben eliminar todos los vuelos asociados a él. Hay varias otras opciones además de `CASCADE`.
- ❖ Proporcionamos un nombre relacionado, que nos brinda una manera de buscar todos los vuelos con un aeropuerto determinado como origen o destino.
- ❖ Cada vez que hacemos cambios en `models.py`, tenemos que realizar migraciones y luego migrar. Ten en cuenta que es posible que debas eliminar tu vuelo existente de Nueva York a Londres, ya que no encaja con la nueva estructura de la base de datos



Relaciones entre Modelos



TIP:

Si hay problemas al modificar tablas que ya existen y Django no compila la migracion entonces debe comentarse el Modelo y ejecutar:

```
python manage.py makemigration  
python manage.py migrate --fake
```

Luego descomentamos el Modelo y ejecutamos nuevamente:

```
python manage.py makemigration  
python manage.py migrate
```

Mostrar los vuelos en la App



urls.py

```
urlpatterns = [
    path('', views.index, name="index"),
]
```

views.py

```
from django.shortcuts import render
from .models import Vuelo, Aeropuerto

# Create your views here.
def index(request):
    return render(request, "vuelos/index.html", {
        "vuelos": Vuelo.objects.all()
    })
```

layout.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Vuelos</title>
  </head>
  <body>
    {% block body %}
    {% endblock %}
  </body>
</html>
```

index.html

```
{% extends "vuelos/layout.html" %}

{% block body %}
  <h1>Vuelos:</h1>
  <ul>
    {% for unVuelo in vuelos %}
      <li>Vuelo {{unVuelo.id }}:
        {{unVuelo.origen }} a {{unVuelo.destino }}</li>
    {% endfor %}
  </ul>
{% endblock %}
```

Mostrar los vuelos en el Shell

```
# Usando el comando de filtro para encontrar todos los aeropuertos con base en Nueva York:
```

```
Aeropuerto.objects.filter(ciudad="Buenos Aires")
```

```
Out: <QuerySet [<Aeropuerto: Buenos Aires (EZE)>]>
```

```
# Usar el comando get para obtener solo un aeropuerto en Nueva York:
```

```
Aeropuerto.objects.get(ciudad="Buenos Aires")
```

```
Out: <Aeropuerto: Buenos Aires (EZE)>
```

```
# Asignar algunos aeropuertos a nombres de variables:
```

```
eze = Aeropuerto.objects.get(ciudad="Buenos Aires")
```

```
In: cdg = Aeropuerto.objects.get(ciudad="Paris")
```

```
# Crear y guardar un nuevo vuelo:
```

```
un_vuelo = Vuelo(origen=eze, destino=cdg, duracion=15)
```

```
In: un_vuelo.save()
```

Relaciones de muchos a muchos



Ahora, trabajemos en la integración de pasajeros en nuestros modelos. Crearemos un modelo de pasajero para comenzar:

```
class Pasajero(models.Model):
    nombre = models.CharField(max_length=64)
    apellido = models.CharField(max_length=64)
    vuelos = models.ManyToManyField(Vuelo, blank=True, related_name="pasajeros")

    def __str__(self):
        return f"{self.nombre} {self.apellido}"
```

- Como comentamos, los pasajeros tienen una relación de Muchos a Muchos (Many To Many) con los vuelos, que describimos en Django usando `ManyToManyField`.
- El primer argumento en este campo es la clase de objetos con los que está relacionado.
- Hemos proporcionado el argumento en `blank=True`, lo que significa que un pasajero puede no tener vuelos.
- Hemos agregado un `related_name` que tiene el mismo propósito que antes: nos permitirá encontrar a todos los pasajeros de un vuelo determinado.

Para realizar estos cambios, debemos realizar migraciones y migrar.

Luego, podemos registrar el modelo de Pasajero en `admin.py` y visitar la página de administración para crear algunos pasajeros.

Relaciones de muchos a muchos



Ahora que agregamos algunos pasajeros, actualizamos nuestra página de vuelos para que muestre a todos los pasajeros de un vuelo. Primero visitaremos `views.py` y actualizaremos nuestra vista de vuelo para proporcionar una lista de pasajeros como contexto. Accedemos a la lista usando el nombre relacionado que definimos anteriormente

```
def vuelo(request, vuelo_id):
    vuelo = Vuelo.objects.get(id=vuelo_id)
    pasajeros = vuelo.pasajeros.all()
    return render(request, "vuelos/vuelo.html", {
        "vuelo": vuelo,
        "pasajeros": pasajeros
    })
```

Ahora, agregamos una lista de pasajeros a `vuelo.html`:

```
<h2>Pasajeros:</h2>
<ul>
    {% for unPasajero in pasajeros %}
        <li>{{ pasajeros }}</li>
    {% empty %}
        <li>No hay pasajeros.</li>
    {% endfor %}
</ul>
```

Relaciones de muchos a muchos



Ahora, trabajemos para brindarles a los visitantes de nuestro sitio la posibilidad de reservar un vuelo. Haremos esto agregando una ruta de reserva en `urls.py`:

```
path("<int:vuelo_id>/reserva", views.reserva, name="reserva")
```

Ahora, agregaremos una función de libro a `views.py` que agrega un pasajero a un vuelo:

```
def reserva(request, vuelo_id):  
    # si el request es un POST agrego un nuevo vuelo    if request.method == "POST":  
  
        # Accesar al vuelo                                vuelo = Vuelo.objects.get(pk=vuelo_id)  
  
        # Encontrar el id del pasajero desde los datos del form    pasajero_id = int(request.POST["pasajero"])  
  
        # Encontrar el pasajero basado en el id                pasajero = Pasajero.objects.get(pk= pasajero_id)  
  
        # Agregar pasajero al vuelo                            pasajero.vuelos.add(vuelo)  
  
        # Redireccionar usuario a la pagina de vuelo            return HttpResponseRedirect(reverse("vuelo", args=(vuelo.id,)))
```

Relaciones de muchos a muchos



A continuación, agregaremos algo de contexto a nuestra plantilla de vuelo para que la página tenga acceso a todos los que no son actualmente un pasajero en el vuelo usando la capacidad de Django para excluir ciertos objetos de una consulta:

```
def vuelo(request, vuelo_id):
    vuelo = Flight.objects.get(id=vuelo_id)
    pasajeros = vuelo.pasajeros.all()
    no_pasajeros = Pasajero.objects.exclude(vuelos=vuelo).all()
    return render(request, "vuelos/vuelo.html", {
        "vuelo": vuelo,
        "pasajeros": pasajeros,
        "no_pasajeros": no_pasajeros
    })
```

Ahora, agregaremos un formulario al HTML de nuestra página de vuelo mediante un campo de entrada de selección:

```
<form action="{% url 'reserva' vuelo.id %}" method="post">
    {% csrf_token %}
    <select name="pasajero" id="">
        {% for unPasajero in no_pasajeros %}
            <option value="{{ pasajero.id }}">{{ pasajero }}</option>
        {% endfor %}
    </select>
    <input type="submit">
</form>
```

Relaciones de muchos a muchos



Podemos usar la aplicación de administración de Django para personalizar lo que mostramos en el admin. Por ejemplo, si deseamos ver todos los aspectos de un vuelo en la interfaz de administración, podemos crear una nueva clase dentro de `admin.py` y agregarla como argumento al registrar el modelo de vuelo:

```
class VueloAdmin(admin.ModelAdmin):  
    list_display = ("id", "origen", "destino", "duracion")  
  
# Register your models here. admin.site.register(Vuelo, VueloAdmin)
```

Consulta la [documentación de administración de Django](#) para encontrar más formas de personalizar la aplicación de administración.

Persistencia





Persistencia

- ❖ Todos los datos de nuestros sistemas necesitan ser guardados en una estructura donde puedan ser recuperados de manera simple, relativamente rápida, y que además sea confiable.
- ❖ Estas estructuras de datos donde se persiste la información se llaman **sistemas de gestión de bases de datos**



SQL – Structured Query Language

- ❖ Los sistemas de gestión de bases de datos SQL son los mas populares hoy en día (y es el que trataremos aquí).
- ❖ Sin embargo Django permite guardar la información en otros sistemas como los denominados NoSQL (MongoDB, Firebase, etc.)
- ❖ Si bien Django **genera automáticamente** las tablas y relaciones sobre la base de datos, necesitamos saber como es ésta estructura y como recuperar la información directamente.

SQL – Structured Query Language

- ❖ El sistema de gestión de bases de datos basado en SQL permite guardar la información de nuestros modelos en tablas y luego relaciona esas tablas entre sí. Por ejemplo:

| origen | destino | duracion |
|--------------|--------------|----------|
| Buenos Aires | Londres | 14 |
| Iguazú | Madrid | 13 |
| Madrid | Tokyo | 13 |
| Buenos Aires | Berlin | 16 |
| Moscú | París | 4 |
| Lima | Buenos Aires | 5 |

SQL – Structured Query Language

- ❖ Existen muchos sistemas de gestión de bases de datos SQL:



SQL – Structured Query Language

- ❖ Django, por defecto, nos permite persistir la información en una base de datos SQLite



- ❖ Que es una base de datos contenida en un solo archivo. Es decir podemos tener varias tablas en un solo archivo de bases de datos.

DESCARGAS Y GUIAS

- [INSTALAR SQLITE](#)
- GUI [DB BROWSER](#)
- GUI [SQLite Administrator](#)
- TEORÍA SOBRE [BASES DE DATOS](#)

TIPOS DE DATOS

- TEXT
- NUMERIC
- INTEGER
- REAL
- BLOB

MySQL

- ❖ Otro de los sistemas gestión de bases de datos SQL más populares de la web es MySQL, por su versatilidad, facilidad de instalación, disponibilidad, y poca utilización de recursos



- ❖ Es propiedad de Oracle y la puedes [descargar aquí](#).

TIPOS DE DATOS

- CHAR(Tamaño)
- VARCHAR(Tamaño)
- SMALLINT
- INT
- BIGINT
- FLOAT
- DOUBLE
- ...

CREAR UNA TABLA

```
CREATE TABLE vuelos (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    origen TEXT NOT NULL,  
    destino TEXT NOT NULL,  
    duracion INTEGER NOT NULL  
);
```

CONSTRAINTS / RESTRICCIONES

- **CHECK:** se utiliza para limitar el rango de valores que se puede colocar en una columna.
- **DEFAULT:** se utiliza para establecer un valor predeterminado para una columna.
- **NOT NULL:** obliga a una columna a NO aceptar valores NULL.
- **PRIMARY KEY:** identifica de forma única cada registro en una tabla.
- **UNIQUE:** asegura que todos los valores de una columna sean diferentes.
- **FOREIGN KEY:** se utiliza para evitar acciones que destruyan los enlaces entre tablas.

INSERTAR DATOS EN UNA TABLA

```
INSERT INTO vuelos  
  (origen, destino, duracion)  
VALUES ("Buenos Aires", "Londres", 14);
```

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos;
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT origen, destino FROM vuelos;
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE id = 3;
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE origen = "Iguazú";
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE origen = "Buenos Aires";
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE duracion > 5;
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE duracion > 5  
AND destino = "Madrid";
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE duracion > 5  
OR destino = "Buenos Aires";
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE  
origen IN ("Buenos Aires", "Lima")
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

OBTENER DATOS DE UNA TABLA

```
SELECT * FROM vuelos WHERE  
origen LIKE "%o%"
```

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |



MySQL

FUNCIONES

- AVERAGE: Promedio
- COUNT: Contar
- MAX: Máximo
- MIN: Mínimo
- SUM: Suma
- ...

ACTUALIZAR DATOS DE UNA TABLA

```
UPDATE vuelos SET duracion = 13  
                WHERE origen = "Buenos Aires"  
                AND destino = "Londres";
```



MySQL

polotic
misiones

BORRAR DATOS DE UNA TABLA

```
DELETE FROM vuelos  
WHERE destino = "Tokyo";
```

OTRAS CLAUSULAS

- **LIMIT**: Limitar a cierta cantidad de filas
- **ORDER BY**: Ordenar por atributo
- **GROUP BY**: Agrupar por atributo
- **HAVING**: Debe devolver filas donde los valores agregados cumplan las condiciones especificadas
- ...



MySQL



CLAVES FORÁNEAS / FOREIGN KEYS

CLAVES FORÁNEAS / FOREIGN KEYS

vuelos

| id | origen | destino | duracion |
|----|--------------|--------------|----------|
| 1 | Buenos Aires | Londres | 14 |
| 2 | Iguazú | Madrid | 13 |
| 3 | Madrid | Tokyo | 13 |
| 4 | Buenos Aires | Berlin | 16 |
| 5 | Moscú | París | 4 |
| 6 | Lima | Buenos Aires | 5 |

CLAVES FORÁNEAS / FOREIGN KEYS

vuelos

| id | origen | codigo_origen | destino | codigo_destino | duracion |
|----|--------------|---------------|--------------|----------------|----------|
| 1 | Buenos Aires | EZE | Londres | LCY | 14 |
| 2 | Iguazú | IGR | Madrid | MAD | 13 |
| 3 | Madrid | MAD | Tokyo | HND | 13 |
| 4 | Buenos Aires | EZE | Berlin | TXL | 16 |
| 5 | Moscú | SVO | París | CDG | 4 |
| 6 | Lima | LIM | Buenos Aires | EZE | 5 |

CLAVES FORÁNEAS / FOREIGN KEYS

aeropuertos

| id | codigo | lugar |
|----|--------|--------------|
| 1 | EZE | Buenos Aires |
| 2 | IGR | Iguazú |
| 3 | MAD | Madrid |
| 4 | LCY | Londres |
| 5 | SVO | Moscú |
| 6 | LIM | Lima |
| 7 | HND | Tokyo |
| 8 | TXL | Berlin |
| 9 | CDG | París |

CLAVES FORÁNEAS / FOREIGN KEYS

aeropuertos

| id | origen_id | destino_id | duracion |
|----|-----------|------------|----------|
| 1 | 1 | 4 | 14 |
| 2 | 2 | 3 | 13 |
| 3 | 3 | 7 | 13 |
| 4 | 1 | 8 | 16 |
| 5 | 5 | 9 | 4 |
| 6 | 6 | 1 | 5 |

CLAVES FORÁNEAS / FOREIGN KEYS

pasajeros

| id | nombre | apellido | vuelos_id |
|----|----------|----------|-----------|
| 1 | Harry | Potter | 1 |
| 2 | Ron | Weasley | 1 |
| 3 | Hermione | Granger | 2 |
| 4 | Draco | Malfoy | 4 |
| 5 | Luna | Lovegood | 6 |
| 6 | Ginny | Weasley | 6 |

CLAVES FORÁNEAS / FOREIGN KEYS

personas

| id | nombre | apellido |
|----|----------|----------|
| 1 | Harry | Potter |
| 2 | Ron | Weasley |
| 3 | Hermione | Granger |
| 4 | Draco | Malfoy |
| 5 | Luna | Lovegood |
| 6 | Ginny | Weasley |

CLAVES FORÁNEAS / FOREIGN KEYS

pasajeros

| persona_id | vuelos_id |
|------------|-----------|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 4 |
| 5 | 6 |
| 6 | 6 |

JUNTAS / JOIN

```
SELECT nombre, origen, destino  
FROM vuelos JOIN pasajeros  
ON pasajeros.vuelos_id = vuelos.id;
```

| nombre | origen | destino |
|----------|--------------|--------------|
| Harry | Buenos Aires | Londres |
| Ron | Buenos Aires | Londres |
| Hermione | Iguazú | Madrid |
| Draco | Buenos Aires | Berlin |
| Luna | Lima | Buenos Aires |
| Ginny | Lima | Buenos Aires |

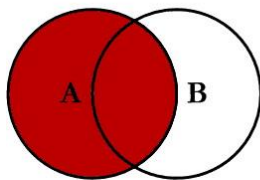
JUNTAS / JOIN

OTRAS CLAUSULAS

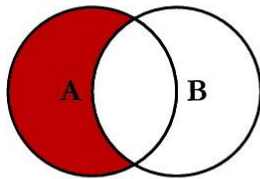
- **JOIN / INNER JOIN**
- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

MySQL

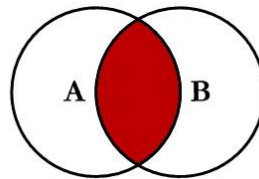
SQL JOINS



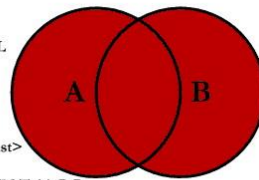
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



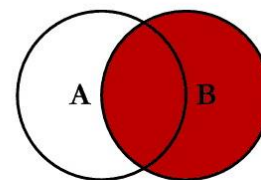
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



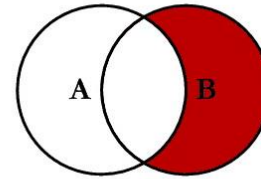
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



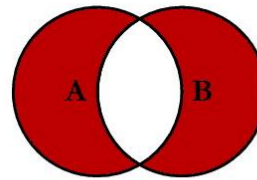
```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```



MySQL

polotic
misiones

VINCULAR MYSQL CON DJANGO

Ejecutar en una terminal el siguiente comando:

```
pip install pymysql
```

Luego agregamos el cliente MySQL a todo nuestro proyecto modificando el archivo `__init__.py` que se encuentra dentro del directorio del proyecto Django escribiendo lo siguiente:

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



VINCULAR MYSQL CON DJANGO

Luego debemos modificar las configuraciones de nuestro proyecto en el archivo settings.py:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```

Lo cambiamos por:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'nombreDB',  
        'USER': 'nombreusuario',  
        'PASSWORD': 'pass',  
        'HOST': 'localhost',  
        'PORT': '3306',  
    }  
}
```



MySQL

polotic
misiones

VINCULAR MYSQL CON DJANGO

Ahora debemos realizar las migraciones de los modelos para poder transportarlos a nuestra nueva BD:

```
python manage.py makemigrations
```

Y también ejecutamos

```
python manage.py migrate
```

Luego podemos controlar que todos los datos se hayan migrado simplemente ejecutando consultas SELECT sobre la base de datos con el cliente de base de datos que mas nos guste. Ya sea el Workbench de MySQL, el HeidiSQL, Valentina Studio, etc.



SEGURIDAD



MANTENER LA PRIVACIDAD CON PYTHON DECOUPLE

Es probable que querramos compartir nuestro código con un equipo de trabajo, pero siempre manteniendo ocultos los datos sensibles de nuestra base de datos. Esto se puede hacer con Django mediante Python Decouple.

Para ello ejecutamos en el terminal:

```
pip install python-decouple
```

Y luego creamos un archivo denominado `.env` en la raíz de nuestro proyecto. También podemos usar una extensión `.ini`. (Recuerda siempre mantenerlo fuera del git agregándolo en `.gitignore`)



SEGURIDAD



MANTENER LA PRIVACIDAD CON PYTHON DECOUPLE

Es probable que querramos compartir nuestro código con un equipo de trabajo, pero siempre manteniendo ocultos los datos sensibles de nuestra base de datos. Esto se puede hacer con Django mediante Python Decouple.

Para ello ejecutamos en el terminal:

```
pip install python-decouple
```

Y luego creamos un archivo denominado `.env` en la raíz de nuestro proyecto. También podemos usar una extensión `.ini`. (Recuerda siempre mantenerlo fuera del git agregándolo en `.gitignore`)

Puedes ver mas detalles de porque se usa env y algunas cosas mas [aquí](#).

SEGURIDAD



MANTENER LA PRIVACIDAD CON PYTHON DECOUPLE

Dentro del archivo .env colocamos los datos sensibles de nuestra BD MySQL (por ejemplo):

```
NAME=nombreDB
USER=nombreusuario
PASSWORD=pass
HOST=localhost
PORT=3306
```

Y dentro de settings.py importamos la librería:

```
from decouple import config
```

Y luego obtenemos los parámetros para cada dato:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': config('NAME'),
        'USER': config('USER'),
        'PASSWORD': config('PASSWORD'),
        'HOST': config('HOST'),
        'PORT': config('PORT', cast=int),
    }
}
```


Autenticación





Autenticación

- ❖ Para que nuestro sistema sea seguro necesitamos que hayan permisos de acceso.
- ❖ La ventaja de usar el framework Django es que ya trae la implementación de autenticación y también disponemos de un sistema de gestión de usuarios y permisos.



Autenticación



- ❖ La autenticación en Django define credenciales de usuario, y las acciones que cada usuario puede realizar.
- ❖ El framework incluye modelos integrados para Users y Groups (una forma genérica de aplicar permisos a más de un usuario a la vez), permisos/flags que designan si un usuario puede realizar una tarea, formularios y vistas para iniciar sesión en usuarios, y herramientas de views para restringir contenido.



Autenticación (Nota)

- ❖ Según Django, el sistema de autenticación pretende ser muy genérico y, por lo tanto, no proporciona algunas características que se proporcionan en otros sistemas de autenticación web. Las soluciones para algunos problemas comunes están disponibles como paquetes de terceros. Por ejemplo, la limitación de los intentos de inicio de sesión y la autenticación mediante terceros (por ejemplo, OAuth).

Autenticación: ¿Cómo lo implementamos?



- ❖ La configuración necesaria se hace automáticamente cuando creamos la aplicación usando el comando
`django-admin startproject.`
- ❖ Las tablas de la base de datos para usuarios y permisos para los modelos se crearon cuando llamamos por primera vez a
`python manage.py migrate.`



Autenticación: ¿Cómo lo implementamos?



- ❖ Ya creaste tu primer usuario cuando mirábamos el sitio de administración de Django clases anteriores (este era un superusuario, creado con el comando `python manage.py createsuperuser`).
- ❖ Nuestro superusuario ya está autenticado y tiene todos los permisos, por lo que necesitaremos crear un usuario de prueba para representar a un usuario normal del sitio. Usaremos **el sitio de administración** para crear nuestros grupos y credenciales, ya que es una de las formas más rápidas de hacerlo.

Autenticación: ¿Cómo lo implementamos?

- ❖ También podemos hacerlo programáticamente, en el caso de que estés armando un panel de control personalizado.

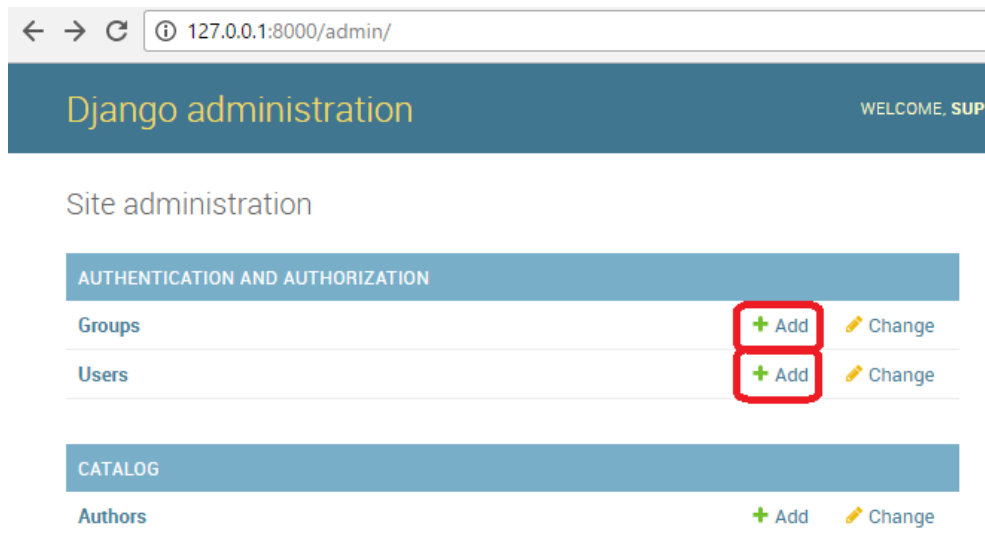
```
from django.contrib.auth.models import User

# Crear usuario y guardarlo en la base de datos
user = User.objects.create_user('harryelgroso', 'harryelgroso@unmail.com', 'laclavedeharry')

# Actualizar los campos del usuario y guardarlo nuevamente
user.first_name = 'Harry'
user.last_name = 'Potter'
user.save()
```

Autenticación: Usuarios y Grupos

- ❖ Inicializamos el servidor local y accedemos a la administración
(<http://127.0.0.1:8000/admin/>)



Autenticación: Grupos

- ❖ Cuando creamos grupos podemos definir los permisos por defecto que Django nos permite configurar:

Home · Authentication and Authorization · Groups · Add group

Add group

Name:

Permissions:

Available permissions ⓘ

Q Filter

- admin | log entry | Can add log entry
- admin | log entry | Can change log entry
- admin | log entry | Can delete log entry
- auth | group | Can add group
- auth | group | Can change group
- auth | group | Can delete group
- auth | permission | Can add permission
- auth | permission | Can change permission
- auth | permission | Can delete permission
- auth | user | Can add user
- auth | user | Can change user
- auth | user | Can delete user

Choose all ⓘ

Hold down "Control", or "Command" on a Mac, to select more than one.

Chosen permissions ⓘ

Remove all ⓘ



Autenticación: Vistas



- ❖ Django proporciona casi todo lo que necesitas para crear páginas de autenticación para manejar el inicio de sesión, el cierre de sesión y la administración de contraseñas "desde el primer momento".
- ❖ Esto incluye un mapeador de URL, vistas y formularios, pero no incluye las plantillas, ¡tenemos que crear las nuestras propias!



Autenticación: Vistas



- ❖ Django proporciona casi todo lo que necesitas para crear páginas de autenticación para manejar el inicio de sesión, el cierre de sesión y la administración de contraseñas "desde el primer momento".
- ❖ Esto incluye un mapeador de URL, vistas y formularios, pero no incluye las plantillas, ¡tenemos que crear las nuestras propias!

Autenticación: Vistas



❖ En nuestro `urls.py` del proyecto agregamos lo siguiente:

#Agregar urls para la autenticación en Django (login, logout, gestión de claves)

```
urlpatterns += [  
    path('accounts/', include('django.contrib.auth.urls')),  
]
```

Navega a la URL `http://127.0.0.1:8000/accounts/` (¡ten en cuenta la barra final!) y Django mostrará un error de que no pudo encontrar esta URL y enumerará todas las URL que intentó.

Autenticación: Vistas

Tendremos que crear templates para cada una de las vistas (vistas automáticas, la genera Django):

- `accounts/ login/ [name='login']`
- `accounts/ logout/ [name='logout']`
- `accounts/ password_change/ [name='password_change']`
- `accounts/ password_change/done/ [name='password_change_done']`
- `accounts/ password_reset/ [name='password_reset']`
- `accounts/ password_reset/done/ [name='password_reset_done']`
- `accounts/ reset/<uidb64>/<token>/ [name='password_reset_confirm']`
- `accounts/ reset/done/ [name='password_reset_complete']`



Autenticación: Directorio de Templates



Las URL (e implícitamente, las vistas) que acabamos de agregar esperan encontrar sus templates asociadas en un directorio `/registration/` en algún lugar de la ruta de búsqueda de templates.

Para este sitio, colocaremos nuestras páginas HTML en el directorio `templates/registration/`.

(Este directorio debe estar en el directorio raíz de tu proyecto, y registrado en los `settings.py`).

Crea estas carpetas ahora.

Autenticación: Directorio de Templates



Para que el directorio de templates sea visible para el cargador de templates, debemos agregarlo en la ruta de búsqueda de templates.

Para ello abres la configuración del proyecto (/tuproyecto/tuproyecto/settings.py).

Luego importas el módulo del sistema operativo os:

```
import os # necesario para el codigo siguiente
```

Autenticación: Directorio de Templates



Actualice la línea 'DIRS' de la sección TEMPLATES como se muestra:

...

```
TEMPLATES = [
```

```
{
```

```
...
```

```
'DIRS': [os.path.join(os.path.dirname(os.path.abspath(__file__)), 'templates')],
```

```
'APP_DIRS': True,
```

```
...
```


Autenticación: Template para el Login

Crea un nuevo archivo HTML llamado:

/tuproyecto/templates/registration/login.html

y proporciona el siguiente contenido:

```
{% extends "registration/layout.html" %}

{% block content %}

{% if form.errors %}
<p>El usuario y la clave no coinciden, por favor intente nuevamente.</p>
{% endif %}

{% if next %}
  {% if user.is_authenticated %}
    <p>Tu cuenta no tiene acceso a esta página. Para proceder,
    por favor inicie sesión con una cuenta que tenga acceso.</p>
  {% else %}
    <p>Inicie sesión para ver esta página.</p>
  {% endif %}
{% endif %}

<form method="post" action="{% url 'login' %}">
{% csrf_token %}

<div>
  <td>Usuario</td>
  <td>{{ form.username }}</td>
</div>
<div>
  <td>{{ form.password.label_tag }}</td>
  <td>{{ form.password }}</td>
</div>

<div>
  <input type="submit" value="ACCEDER" />
  <input type="hidden" name="next" value="{{ next }}" />
</div>
</form>

{# Asume que configuras la vista password_reset en tu URLconf #}

<p><a href="{% url 'password_reset' %}">¿Olvidó su contraseña?</a></p>

<p><a href="{% url 'USUARIOS:registrarse' %}">Registrarse</a></p>

{% endblock %}
```

Autenticación: Template para el Login



Si inicias sesión con credenciales válidas, serás redirigido a otra página (de forma predeterminada, será `http://127.0.0.1:8000/accounts/profile/`).

El problema es que, de forma predeterminada, Django espera que, al iniciar sesión, vayas al perfil, que a veces no es lo que quieras hacer.

Como aún no has definido esta página, obtendrás otro error.

Abre la configuración del proyecto (`/tuproyecto/tuproyecto/settings.py`) y agrega el texto a continuación en la parte inferior.

Ahora, cuando inicie sesión, debería ser redirigido a la página de inicio del sitio de forma predeterminada.

```
# Redirigir a la URL de home después de iniciar sesión (la
redirección predeterminada va a /accounts/profile/)
LOGIN_REDIRECT_URL = '/'
```

Autenticación: Template para el Logout



Si bien un Usuario puede acceder a la URL `http://127.0.0.1:8000/accounts/logout/` para intentar desconectarse, lo que hará en realidad es ser llevado a la página de Administración. Ésta área es solo permitida para los que son administradores (`is_staff`).

Abrimos el template que encontramos en `/templates/registration/logged_out.html` y editamos el código:

```
{% extends "registration/layout.html" %}

{% block content %}

<p>Has sido desconectado.</p>

<a href="{% url 'login'%}">Click aqui para volver a acceder.</a>

{% endblock %}
```



Autenticación: Template para Reinicio de Clave



Si un usuario se olvida de su clave, es bueno proveerle de un método para reiniciarla mediante el correo electrónico. Necesitamos crear varias plantillas para cada situación:

1. Formulario de reinicio de contraseña
2. Reinicio de contraseña exitoso
3. Email de reinicio de contraseña
4. Confirmación de reinicio de contraseña
5. Reinicio de contraseña completado

Autenticación: Formulario de reinicio de contraseña



Crea la plantilla: /templates/registration/password_reset_form.html

```
{% extends "registration/layout.html" %}
{% block content %}
<h1>Formulario para cambiar su clave</h1>
<p>Debe ingresar el correo electrónico que usó al registrarse:</p>
<form action="" method="post">{% csrf_token %}
    {% if form.email.errors %} {{ form.email.errors }} {% endif %}
    <p>{{ form.email }}</p>
    <input type="submit" class="btn btn-default btn-lg" value="Reset password" />
</form>
<p><a href="{% url 'login' %}">Volver al Login</a></p>
{% endblock %}
```

Autenticación: Reinicio de contraseña exitoso



Crea la plantilla: /templates/registration/password_reset_done.html

```
{% extends "registration/layout.html" %}
{% block content %}
<p> Le enviamos por correo electrónico instrucciones para configurar su contraseña.
    Si no han llegado en unos minutos, revise su carpeta de correo no deseado. </p>
{% endblock %}
```

Autenticación: Email de reinicio de contraseña



Crea la plantilla: /templates/registration/password_reset_email.html

Esta plantilla contendrá código HTML con el enlace de reset que será enviado por correo electrónico al email registrado.

```
Alguien solicitó el restablecimiento de la contraseña del correo electrónico {{email}}.  
Siga el enlace a continuación:  
{{ protocol }}://{{ domain }}{% url 'password_reset_confirm' uidb64=uid token=token %}  
Si no fuiste tú, solo ignora este correo.
```

Autenticación: Confirmación de reinicio de contraseña



Crea la plantilla: /templates/registration/password_reset_confirm.html que contendrá el formulario para completar con la nueva clave:

```
{% extends "registration/layout.html" %}

{% block content %}

{% if validlink %}
    <p>Ingresa (y confirme) su nuevo password.</p>
    <form action="" method="post">
        <div style="display:none">
            <input type="hidden" value="{{ csrf_token }}" name="csrfmiddlewaretoken">
        </div>
        <table>
            <tr>
                <td>{{ form.new_password1.errors }}
                <label for="id_new_password1">Nuevo password:</label></td>
                <td>{{ form.new_password1 }}</td>
            </tr>
            <tr>
                <td>{{ form.new_password2.errors }}
                <label for="id_new_password2">Confirmar password:</label></td>
                <td>{{ form.new_password2 }}</td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="Cambiar mi password" /></td>
            </tr>
        </table>
    </form>

{% else %}
    <h1> Error al restablecer el password </h1>
    <p> El enlace para restablecer la contraseña no es válido, posiblemente porque ya se ha utilizado. Solicite un nuevo restablecimiento de contraseña. </p>
{% endif %}
<p><a href="{% url 'login' %}">Volver</a></p>
{% endblock %}
```


Autenticación: Reinicio de contraseña completado



Llegamos al ultimo paso. Con ésta plantilla notificarás que todo ha sido exitoso:

Crea la plantilla: /templates/registration/password_reset_complete.html

```
{% extends "registration/layout.html" %}

{% block content %}

<h1>Ha cambiado su contraseña</h1>
<p><a href="{% url 'login' %}">¿Acceder nuevamente?</a></p>

{% endblock %}
```



Autenticación: Probar las Páginas



Ahora que hemos terminado de configurar todas las paginas de login puedes probarlas accediendo a las URL:

- `http://127.0.0.1:8000/accounts/login/`
- `http://127.0.0.1:8000/accounts/logout/`

¡OJO!

Los correos al enviar el enlace de contraseña no funcionarán porque no hemos configurado el sistema de envíos de correo en Django. Para realizar las pruebas debes setear lo siguiente en tu archivo settings.py (escribirlo al final)

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Autenticación: Probando en Plantillas



Ahora puede que necesitemos obtener información sobre los usuarios que estén conectados al Sistema. Para obtener estos datos accedemos a la variable `{{ user }}`

La variable `{{ user.is_authenticated }}` es la que nos permite saber si esta logueado el usuario para poder mostrar la web.

Podemos por ejemplo cambiar el comportamiento del layout de la aplicación de Aerolínea de nuestra plantilla generica para que muestre el nombre de usuario y el boton **Desconectarse** (si esta logueado) o **Acceder** si no lo está:

```
{% if user.is_authenticated %}
    <p>Usuario: {{ user.get_username }}</p>
    <p><a href="{% url 'logout' %}?next={{ request.path }}">Desconectarse</a></p>
    {% block body %}
    {% endblock %}
{% else %}
    <p>No podra visualizar los datos de los vuelos sin estar conectado.</p>
    <p><a href="{% url 'login' %}?next={{ request.path }}">Acceder</a></p>
{% endif %}
```

Este parámetro nos permite recordar la URL donde se encuentra el usuario para poder redirigir a la misma luego de la acción de click Desconectarse



Autenticación: Restringiendo en Vistas



También puede ser que necesitemos restringir el acceso a ciertas funciones en las vistas si el usuario no está registrado. Para ello importamos el modulo `login_required` y luego agregamos el decorator sobre la función que querramos restringir:

```
from django.contrib.auth.decorators import login_required

@login_required
def vista_restringida(request):
    ...
```

También se puede usar (dentro del código de una función) `request.user.is_authenticated` para bloquear ciertas partes de la ejecución.

Autenticación: Restringiendo en Vistas

Si tienes vistas basadas en clases lo que puedes hacer es heredar de LoginRequiredMixin por ejemplo:

```
from django.contrib.auth.mixins import LoginRequiredMixin  
  
class MyView(LoginRequiredMixin, View):  
    login_url = '/login/'  
    redirect_field_name = 'redirect_to'
```

Redireccion a una URL si el usuario no esta autenticado

Un nombre de parámetro URL en lugar del next para redireccionar

Autenticación: Permisos sobre vistas



También puedes restringir el acceso a vistas según permisos, por ejemplo al alta de un vuelo de nuestra aplicacion Aerolinea:

```
from django.contrib.auth.decorators import permission_required

@permission_required('AEROLINEA.add_vuelo')
def vuelo_alta(request):
    if request.method == "POST":
        form = FormVueloCustom(request.POST)
        if form.is_valid():
            form.save()
    else:
        form = FormVueloCustom()
        return render(request, "vuelos/vuelo_alta.html", {
            "formset": form
        })
```



Autenticación: Permisos sobre templates



Si queremos, ademas podemos restringir porciones de codigo HTML dentro de los Templates:

```
{% if perms.AEROLINEA.add_vuelo %}  
    <a href="{% url 'AEROLINEA:vuelo_alta'%}">Alta de Vuelo</a>  
{% endif %}
```



Registro de un nuevo usuario (Sign Up)



No hemos visto aún como dar el alta de un usuario. Es decir permitir que cualquiera pueda registrarse en nuestro sitio evitando tener que cargarlos individualmente de manera manual por el admin.

Para hacerlo tenemos que crear primeramente una aplicación que gestione el registro de Usuario y tener así esta sección independiente. Lo hacemos creando una app USUARIOS y dentro de urls.py ponemos:

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

app_name = "USUARIOS"
urlpatterns = [
    path('', views.registrarse, name="registrarse"),
]
```


Registro de un nuevo usuario (Sign Up)



Dentro de la app USUARIOS creamos un archivo forms.py con los campos de registro del Usuario:

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class RegistroForm(UserCreationForm):
    username = forms.CharField(
        label='Usuario', widget=forms.TextInput(attrs={'class': 'usuario'}), max_length=150, required=True, help_text='Requerido. 150 caracteres o menos. Letras, dígitos y @ / . / + / - / _ solamente.')
    password1 = forms.CharField(
        label='Password', widget=forms.PasswordInput(), max_length=30, required=True, help_text='Requerido. Al menos 8 caracteres y no pueden ser todos numeros.')
    password2 = forms.CharField(
        label='Repetir Password', widget=forms.PasswordInput(), max_length=30, required=True, help_text='Requerido. Ingrese la misma contraseña que antes, para verificación .')
    first_name = forms.CharField(
        label='Nombre', widget=forms.TextInput(attrs={'class': 'nombre'}), max_length=30, required=False, help_text='Opcional.')
    last_name = forms.CharField(
        label='Apellido', widget=forms.TextInput(attrs={'class': 'apellido'}), max_length=30, required=False, help_text='Opcional.')
    email = forms.EmailField(
        label='Email', widget=forms.TextInput(attrs={'class': 'email'}), max_length=254, required=True, help_text='Se requiere una direccion de email valida.')

    class Meta:
        model = User
        fields = ('username', 'first_name', 'last_name', 'email', 'password1', 'password2', )
```

Registro de un nuevo usuario (Sign Up)



Desglosamos lo que estamos haciendo:

```
username = forms.CharField(  
    label='Usuario',  
    widget=forms.TextInput(attrs={'class': 'usuario'}),  
    max_length=150,  
    required=True,  
    help_text='Requerido. 150 caracteres o menos. Letras, dígitos y @ / . / + / - / _ solamente.'  
)
```

El campo que estoy formateando

El label que aparecerá en el campo del formulario

Puedo colocar atributos que posteriormente usaré en el CSS

La longitud máxima de caracteres del input

Si se debe validar como campo requerido o no

Que mensaje de ayuda mostraré al usuario

Registro de un nuevo usuario (Sign Up)



Tenemos que construir el view dentro de views.py que procese el registro:

```
from django.http.response import HttpResponseRedirect
from django.shortcuts import render
from django.urls import reverse
from .forms import *

def registrarse(request):
    if request.method == 'POST':
        form = RegistroForm(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect(reverse('login'))
    else:
        form = RegistroForm()
    return render(request, 'registration/registro.html', {
        'form': form
    })
```

Registro de un nuevo usuario (Sign Up)



Construimos el formulario en HTML. En este caso lo pondré en el mismo directorio de registration donde se encuentra el login, logout, etc. Lo llamaré registro.html

```
{% extends "registration/layout.html" %}

{% block content %}

<h2>Formulario de Registro</h2>
<form method="post">
  {% csrf_token %}
  {% for campo in form %}
    <p>
      {{ campo.label_tag }}<br>
      {{ campo }}
      {% if campo.help_text %}
        <small style="color: grey">{{ campo.help_text }}</small>
      {% endif %}
      {% for error in campo.errors %}
        <p style="color: red">{{ error }}</p>
      {% endfor %}
    </p>
  {% endfor %}
  <button type="submit">Confirmar Registro</button>
</form>

{% endblock %}
```

Registro de un nuevo usuario (Sign Up)

Por último, en el urls.py de mi Proyecto agrego una URL mas:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("AEROLINEA.urls")),
    path('AEROLINEA/', include("AEROLINEA.urls")),
    path('accounts/', include('django.contrib.auth.urls')),
    path('registrarse/', include('USUARIOS.urls')),
]
```



Registro de un nuevo usuario (Sign Up)



Desde el formulario de login, simplemente se hace una llamada a la pantalla de registro con un hipervinculo:

```
<p><a href="{% url 'USUARIOS:registrarse' %}">Registrarse</a></p>
```



Trabajemos Juntos



Ya cuentan con todas las herramientas para construir un sitio web desde cero con Python, gracias a Django.

Pueden incluso armar toda su estructura sobre SQLite y luego migrar al sistema de gestión de BD que quieran. ¡Incluso NoSQL!

También pueden personalizar cada página para que la pueda ver uno u otro usuario según los permisos. ¡Incluso darle el estilo que quieran a cada pantalla!

¡Muchos éxitos!



#

¡HASTA LA
próxima!

www.polotic.misiones.gob.ar

[f](#) [@](#) [v](#) [d](#) /poloticmisiones

polotic
misiones



Gobierno
de Misiones

