



Servidor Django completo

Débora Rubio

Enlace al repositorio: https://github.com/debsign/UE_MOD6_desarrollo_backend/tree/main/ejercicio_entregable_4

Explicación del código desarrollado

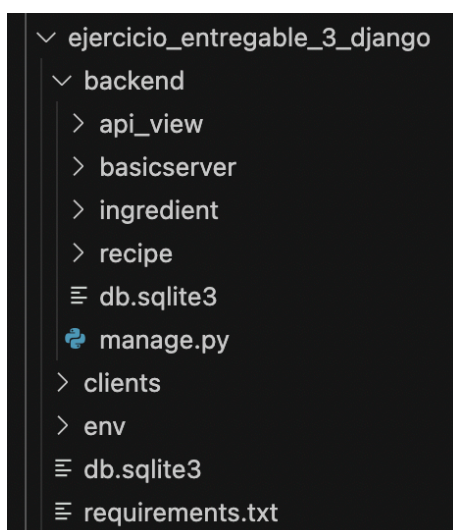
Estos son los pasos que hemos seguido para el desarrollo de un servidor completo con Django que implemente todas las herramientas vistas en la unidad 4 del módulo.

El ejercicio es la continuación del ejercicio ya realizado para la unidad 3, por lo que los modelos serán los mismos (y sabemos que tienen sentido).

Los requisitos son los siguientes:

1. Gestión de usuarios:
 - a. Crear al menos 2 grupos distintos (por ejemplo, administradores y empleados)
 - b. Limitar el uso de los grupos que no sean administradores (Por ejemplo, hacer que los empleados no puedan borrar registros de algún modelo)
2. Implementar los tests unitarios necesarios para comprobar que la API funciona correctamente.
3. Implementar al menos 1 viewset (en caso de estar implementando el ejercicio anterior, se puede cambiar alguna vista a implementada a viewset).
4. Utilizar MySQL como BBDD (importante explicar paso a paso como se ha hecho la migración, en caso de que la BBDD ya estuviese creada en sqlite).
5. Que sea completamente portable (que exista un fichero requirements.txt con todas las dependencias utilizadas).
6. Aplicar DRY
7. Utilizar al menos 4 vistas genéricas distintas para cada modelo
8. Al menos una api_view propia que enlace modelos (que no sean vistas genéricas)

Esta era la estructura final de mi proyecto en la entrega de la unidad anterior:



Por lo que iremos desgranando cada parte, según las hemos necesitado. Lo principal fue crear el entorno, junto con las carpetas **backend** y **clients**. Dentro de backend se encuentra nuestro servidor, la carpeta **api_view** y las carpetas de cada modelo que vaya a crear, todo junto al archivo *manage.py*. En mi caso para los modelos escogí la temática “receta” e “ingrediente”.

Gestión de usuarios

- a. Crear al menos 2 grupos distintos (por ejemplo, administradores y empleados)

Para poder acceder al admin, en nuestro caso, debemos crear un superadmin para tener acceso:

```
(env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend % python manage.py createsuperuser
Username (leave blank to use 'deborarubiosoliva'): debora
Email address:
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend %
```

username: debora
password: 123123

Y ahora ya podemos acceder al admin (<http://127.0.0.1:8000/admin/>) de nuestro proyecto con las credenciales que acabamos de crear:

← → ↺ 🔒 127.0.0.1:8000

Page not found (404)

Request Method: GET
Request URL: http://127.0.0.1:8000/

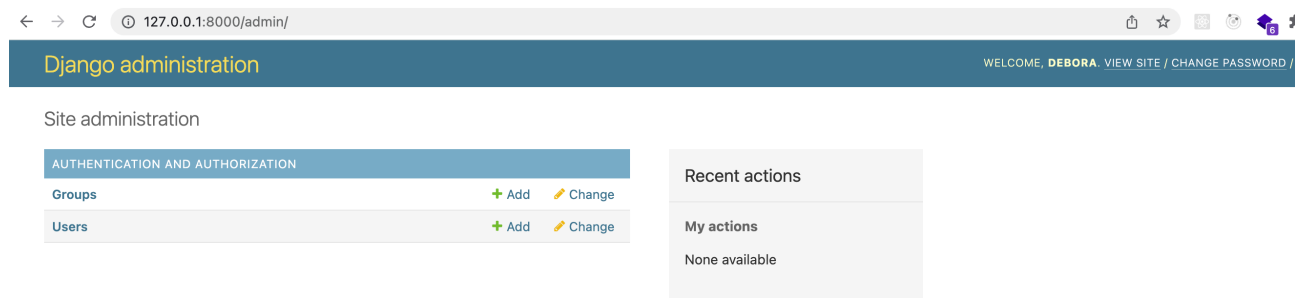
Using the URLconf defined in `basicserver.urls`, Django tried these URL patterns, in this order:

1. `admin/`
2. `api_view/`
3. `api_view/recipe/`
4. `api_view/ingredient/`
5. `api_view/ingredient-recipes/`

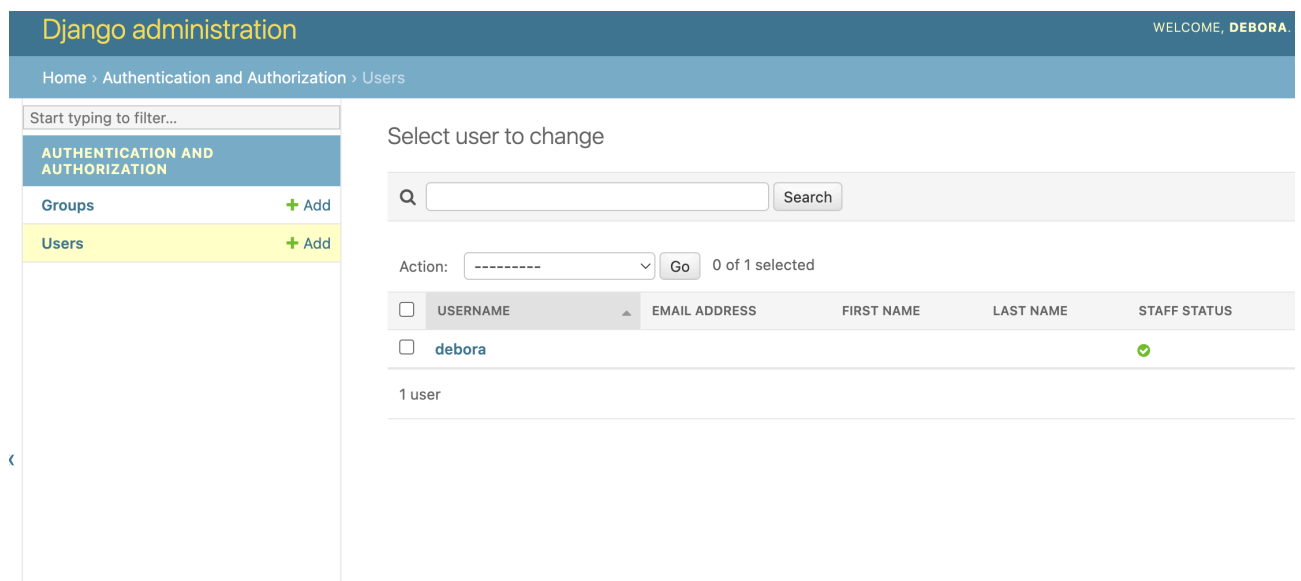
The empty path didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

De momento nos encontramos con lo siguiente:



En Users está el que hemos creado:



Y vamos a crear los dos grupos:

- administrador
- trabajador

b. Limitar el uso de los grupos que no sean administradores (Por ejemplo, hacer que los empleados no puedan borrar registros de algún modelo)

Lo que hacemos es modificar los grupos para que los usuarios que pertenezcan al grupo de administradores puedan realizar cualquier consulta de ambos modelos y que los usuarios que pertenezcan al grupo de trabajadores no puedan borrar entradas de ninguno de los dos modelos:

Vamos a comprobar esto:

Creamos dos usuarios, uno para cada grupo:

username: Maria_jefa

password: ejemplo123123

y le asignamos el grupo “administrador”.

username: Jose_trabajador

password: ejemplo123123

y le asignamos el grupo “trabajador”.

Importante que les marquemos que forman parte del staff para que puedan entrar al admin.

Add group

Name:

Permissions:

Available permissions ?

auth | permission | Can view permission

auth | user | Can add user

auth | user | Can change user

auth | user | Can delete user

auth | user | Can view user

contenttypes | content type | Can add content type

contenttypes | content type | Can change content type

contenttypes | content type | Can delete content type

contenttypes | content type | Can view content type

sessions | session | Can add session

sessions | session | Can change session

sessions | session | Can delete session

sessions | session | Can view session

Choose all ?

Chosen permissions ?

ingredient | ingredient | Can add ingredient

ingredient | ingredient | Can change ingredient

ingredient | ingredient | Can delete ingredient

ingredient | ingredient | Can view ingredient

recipe | recipe | Can add recipe

recipe | recipe | Can change recipe

recipe | recipe | Can delete recipe

recipe | recipe | Can view recipe

Remove all

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

Add group

Name:

Permissions:

Available permissions ?

auth | user | Can change user

auth | user | Can delete user

auth | user | Can view user

contenttypes | content type | Can add content type

contenttypes | content type | Can change content type

contenttypes | content type | Can delete content type

contenttypes | content type | Can view content type

ingredient | ingredient | Can delete ingredient

recipe | recipe | Can delete recipe

sessions | session | Can add session

sessions | session | Can change session

sessions | session | Can delete session

sessions | session | Can view session

Choose all ?

Chosen permissions ?

ingredient | ingredient | Can add ingredient

ingredient | ingredient | Can change ingredient

ingredient | ingredient | Can view ingredient

recipe | recipe | Can add recipe

recipe | recipe | Can change recipe

recipe | recipe | Can view recipe

Remove all

Para poder consultar con mayor facilidad el efecto que las restricciones tienen sobre nuestros modelos, lo que hacemos es importarlos en el archivo admin.py de cada modelo:

```
ejercicio_entregable_3_django > backend > ingredient > admin.py
1  from django.contrib import admin
2
3  # Register your models here.
4  from .models import Ingredient
5
6  admin.site.register(Ingredient)
7
```

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#) [Change](#)

Users [+ Add](#) [Change](#)

INGREDIENT

Ingredients [+ Add](#) [Change](#)

RECIPE

Recipes [+ Add](#) [Change](#)

Y nos logueamos con cada usuario:

En el caso de Maria_jefa, vemos que podemos entrar a cada ingrediente, por ejemplo, y eliminarlo, añadir, etc

Django administration

WELCOME, MARIA_JEFA. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Ingredient > Ingredients > Ingredient object (9)

Start typing to filter...

INGREDIENT

Ingredients [+ Add](#)

RECIPE

Recipes [+ Add](#)

Change ingredient

Ingredient object (9)

Name:

Quantity:

SAVE

Save and add another

Save and continue editing

Delete

Pero en el caso de Jose_trabajador, vemos que podemos entrar a cada ingrediente, por ejemplo, y añadir pero no eliminar:

Django administration

WELCOME, JOSE_TRABAJADOR. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Ingredient > Ingredients > Ingredient object (9)

Start typing to filter...

INGREDIENT

Ingredients + Add

RECIPE

Recipes + Add

Change ingredient

Ingredient object (9) HISTORY

Name: Patata

Quantity: 300

SAVE Save and add another Save and continue editing

Por consola también podemos restringir los permisos, añadiendo las clases de permiso y autenticación en las vistas, pero creo que con esto he completado lo que se requería en este punto.

Implementar los tests unitarios necesarios para comprobar que la API funciona correctamente.

Vamos al archivo backend/ingredient/tests.py y borramos todo lo que hay, porque vamos a usar los tests de rest_framework, no de TestCase:

```
ejercicio_entregable_3_django > backend > ingredient > tests.py > ...
1  from rest_framework import status
2  from rest_framework.test import APITestCase
3
4  # vamos a testear si podemos crear los usuarios
5  class IngredientTestCase(APITestCase):
6      def test_ingredient_creation(self):
7          # le pasamos el dato para crear el ingredient
8          data = {'name':'Bacon', 'quantity':100}
9          # le decimos la url que usamos para crearlo, más el tipo post
10         res = self.client.post('/ingredients/', data)
11         # importante saber que esto no se va a guardar en nuestra base de datos, si no en una ddbb temporal
12         # lo que queremos saber es si el código de respuesta es igual al 201, lo que significará que ha sido creado
13         self.assertEqual(res.status_code, status.HTTP_201_CREATED)
14
```

Y vamos a la terminal y nos aparece que ha pasado el test correctamente:

```
● (env) deborarubiosoliva@MacBook-Air-de-Debora-2 backend % python manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
-----
Ran 1 test in 0.006s

OK
Destroying test database for alias 'default'...
○ (env) deborarubiosoliva@MacBook-Air-de-Debora-2 backend %
```

Realizamos el resto de tests para el resto de peticiones:

```
def setUp(self):
    # creamos un ingrediente para usar en las pruebas
    self.ingredient = Ingredient.objects.create(name='Bacon', quantity=100)
```



```
def test_ingredient_retrieval(self):
    # petición get para obtener el ingrediente creado
    res = self.client.get(f'/ingredients/{self.ingredient.id}/')
    # lo que queremos saber es si el código de respuesta es igual al 200, lo que significará que se ha obtenido correctamente
    self.assertEqual(res.status_code, status.HTTP_200_OK)
    # comprobamos que el ingrediente que obtenemos es el que hemos creado
    self.assertEqual(res.data['name'], 'Bacon')
```

```
def test_ingredient_update(self):
    # petición patch para actualizar el nombre del ingrediente
    res = self.client.patch(f'/ingredients/{self.ingredient.id}/', {'name': 'Panceta'})
    # lo que queremos saber es si el código de respuesta es igual al 200, lo que significará que se ha modificado correctamente
    self.assertEqual(res.status_code, status.HTTP_200_OK)
    # actualizamos el objeto de la ddbb
    self.ingredient.refresh_from_db()
    # comprobamos que el ingrediente que obtenemos se llama con el nuevo nombre
    self.assertEqual(self.ingredient.name, 'Panceta')
```

```
def test_ingredient_deletion(self):
    # petición delete para borrar el ingrediente
    res = self.client.delete(f'/ingredients/{self.ingredient.id}/')
    # lo que queremos saber es si el código de respuesta es igual al 204, lo que significará que no hay contenido en la respuesta
    self.assertEqual(res.status_code, status.HTTP_204_NO_CONTENT)
    # usamos assertRaises para verificar que se genera una excepción Ingredient.DoesNotExist cuando se intenta obtener un ingrediente que ha sido eliminado
    # assertRaises(exception, callable, *args, **kwargs)
    self.assertRaises(Ingredient.DoesNotExist, Ingredient.objects.get, id=self.ingredient.id)
```

Y los pasa correctamente:

```
(env) deborarubiosoliva@MacBook-Air-de-Debora-2 backend % python manage.py test
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
-----
Ran 4 tests in 0.011s

OK
Destroying test database for alias 'default'...
(env) deborarubiosoliva@MacBook-Air-de-Debora-2 backend %
```

Implementar al menos 1 viewset (en caso de estar implementando el ejercicio anterior, se puede cambiar alguna vista a implementada a viewset).

Creamos el archivo backend/ingredient/viewsets.py (se puede hacer en el archivo views.py directamente pero como los videos de ejemplo, lo hacemos en un archivo separado)

```
ejercicio_entregable_3_django > backend > ingredient > viewsets.py > ...
1  from rest_framework import viewsets
2
3  from .models import Ingredient
4  from .serializer import IngredientSerializer
5
6  # bastante similar a lo que teníamos en el genérico
7  class IngredientViewSet(viewsets.ModelViewSet):
8      queryset = Ingredient.objects.all()
9      serializer_class = IngredientSerializer
10     lookup_field = 'pk'
11
```

Creamos el archivo backend/basicserver/routers.py (se puede hacer en el archivo urls.py directamente, pero mismo caso que con el viewset)

```
ejercicio_entregable_3_django > backend > basicserver > routers.py > ...
1  from rest_framework.routers import DefaultRouter
2  from ingredient.viewsets import IngredientViewSet
3
4  router = DefaultRouter()
5  router.register('ingredients', IngredientViewSet, basename='ingredient')
6
7  urlpatterns = router.urls
8
```

Ahora vamos a backend/basicserver/urls.py y añadimos a las urlpatterns:

```
20
21  urlpatterns = [
22      path('admin/', admin.site.urls),
23      # se incluyen las urls del archivo creado en api_view
24      path('api_view/', include('api_view.urls')),
25      path('api_view/recipe/', include('recipe.urls')),
26      path('api_view/ingredient/', include('ingredient.urls')),
27      path('api_view/ingredient-recipes/', include('api_view.urls')),
28      path('viewset/', include('basicserver.routers'))
29  ]
30
```

Y ahora nos falta que, ya que queremos que nos busque por el lookupfield, tenemos que añadirselo en el serializer del User, backend/users/serializer.py:

```
class IngredientSerializer(serializers.ModelSerializer):  
    class Meta:  
        model = Ingredient  
        fields = [  
            'pk'  
            , 'name'  
            , 'quantity'  
        ]
```

Ahora visitamos <http://127.0.0.1:8000/viewset/>

127.0.0.1:8000/viewset/

Django REST framework

Api Root

Api Root

The default basic root view for DefaultRouter

GET /viewset/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  
  "ingredients": "http://127.0.0.1:8000/viewset/ingredients/"  
}
```

Y si vamos a la url que nos pone <http://127.0.0.1:8000/viewset/ingredients/>

Vamos al viewset de nuestro user donde podemos realizar todas las peticiones:

Django REST framework debora

Ingredient List

OPTIONS GET +

GET /viewset/ingredients/

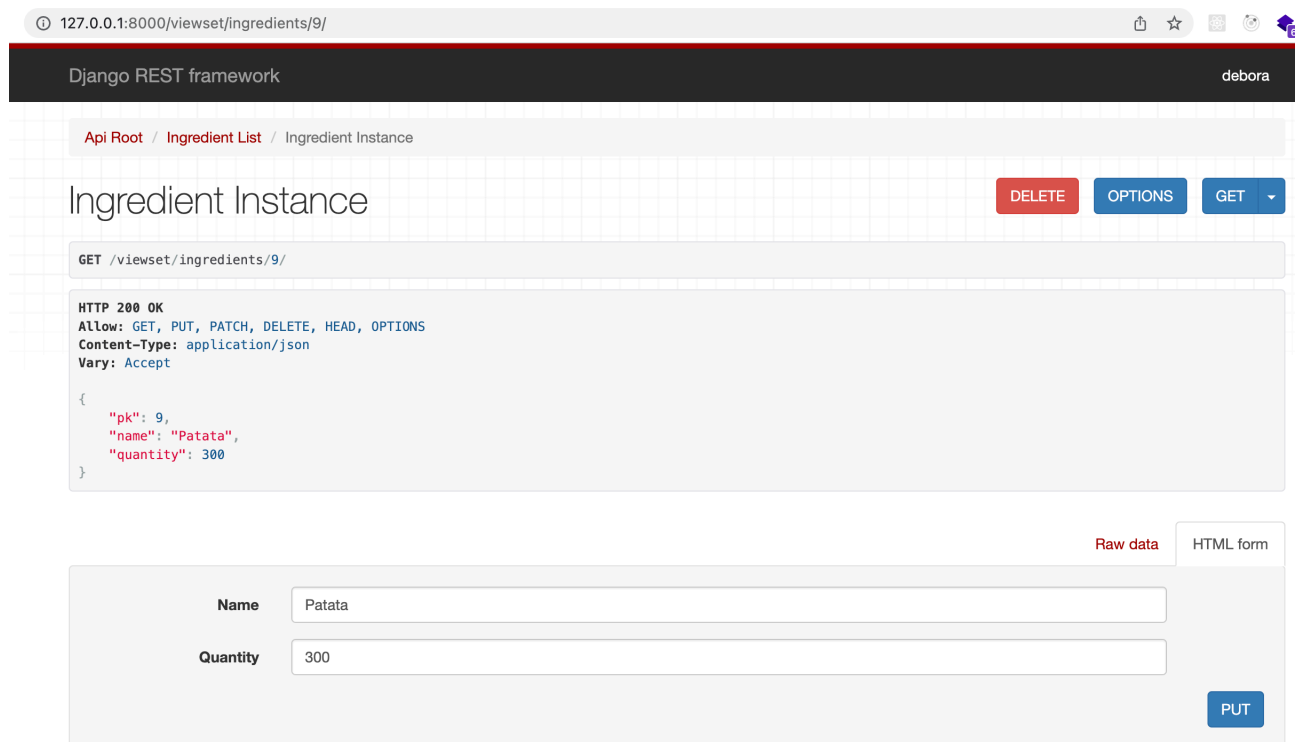
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
[  
  {  
    "pk": 7,  
    "name": "Tomate",  
    "quantity": 100  
  },  
  {  
    "pk": 8,  
    "name": "Queso",  
    "quantity": 50  
  },  
  {  
    "pk": 9,  
    "name": "Patata",  
    "quantity": 300  
  }  
]
```

Raw data HTML form

Name

Y si además vamos a <http://127.0.0.1:8000/viewset/ingredients/9/> nos devuelve el ingrediente con el id = 9, y podemos editarlo, borrarlo... todo



127.0.0.1:8000/viewset/ingredients/9/

Django REST framework debora

Api Root / Ingredient List / Ingredient Instance

Ingredient Instance

DELETE OPTIONS GET

GET /viewset/ingredients/9/

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "pk": 9,
  "name": "Patata",
  "quantity": 300
}
```

Raw data HTML form

Name Patata

Quantity 300

PUT

El resumen es que únicamente con este viewset estamos haciendo todo lo que hacemos en views, simplificando el código al máximo.

A veces queremos un viewset que no tenga para editar, por ejemplo, y ahí es donde entra el concepto de mixins, pero en este ejercicio no vamos a implementarlo.

Pero realmente lo que pide el ejercicio es que, al tener las vistas ya implementadas de la entrega anterior, se pide sustituir esas 4 vistas creadas por un viewset, por lo que haremos las siguientes modificaciones:

- En el modelo ingredients, que es el que vamos a modificar, vamos al archivo urls.py y ponemos el mismo contenido que habíamos puesto en routers.py, donde añadimos el viewset creado (comentamos las urls de cada view creada anteriormente):

```
ejercicio_entregable_3_django > backend > ingredient > urls.py > ...
1  # from django.urls import path
2  # from . import views
3
4  #views
5  # urlpatterns = [
6  #     path('<int:pk>', views.IngredientRetrieveAPIView.as_view()),
7  #     path('', views.IngredientListCreateAPIView.as_view()),
8  #     path('<int:pk>/update/', views.IngredientUpdateAPIView.as_view()),
9  #     path('<int:pk>/destroy/', views.IngredientDestroyAPIView.as_view()),
10 # ]
11
12 # viewset
13 from rest_framework.routers import DefaultRouter
14 from ingredient.viewsets import IngredientViewSet
15
16 router = DefaultRouter()
17 router.register('', IngredientViewSet, basename='ingredient')
18
19 urlpatterns = router.urls
20
```

- En el archivo basicserver/urls.py importamos el router:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    # se incluyen las urls del archivo creado en api_view
    path('api_view/', include('api_view.urls')),
    path('api_view/recipe/', include('recipe.urls')),
    # path('api_view/ingredient/', include('ingredient.urls')),
    path('api_view/ingredient-recipes/', include('api_view.urls')),
    # path('viewset/', include('basicserver.routers')),
    path('ingredients/', include('ingredient.urls'))
]
```

Es cierto que este contenido podemos directamente importarlo desde routers.py que ya lo tenemos, pero si lo importamos, la url que nos pinta del viewset sería <http://127.0.0.1:8000/ingredients/ingredients> o [viewset/ingredients](http://127.0.0.1:8000/viewset/ingredients), entonces bueno, lo dejamos separado (todo esto lo dejo comentado porque es un ejercicio de práctica, en el caso de usarlo como proyecto para mi portfolio o incluso en producción lo limpiaría).

El caso es que si entramos en <http://127.0.0.1:8000/ingredients/> comprobaremos que tenemos el listado de todos los ingredientes, al cual le podemos añadir nuevos ingredientes:

Django REST framework

Ingredient List

OPTIONS GET

GET /ingredients/

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

Cargar página de nuevo

```
[
  {
    "pk": 7,
    "name": "Tomate",
    "quantity": 100
  },
  {
    "pk": 8,
    "name": "Queso",
    "quantity": 50
  },
  {
    "pk": 9,
    "name": "Patata",
    "quantity": 300
  }
]
```

Raw data HTML form

Name

Y si entramos a <http://127.0.0.1:8000/ingredients/9/> podremos eliminar, editar... el ingrediente en cuestión:

Django REST framework

Ingredient List / Ingredient Instance

DELETE OPTIONS GET

GET /ingredients/9/

HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "pk": 9,
  "name": "Patata",
  "quantity": 300
}
```

Raw data HTML form

Name

Quantity

PUT

En definitiva, todo lo que podíamos hacer en las 4 views creadas.

Utilizar MySQL como BBDD (importante explicar paso a paso como se ha hecho la migración, en caso de que la BBDD ya estuviese creada en sqlite).

Vamos a migrar nuestra base de datos sqlite de Django a MySQL, y para ello hay que crear la base de datos. Abrimos terminal y escribimos:

```
> mysql -u root -p
```

Ponemos la contraseña de nuestro ordenador y entra a la consola de mysql:

creamos la tabla

```
mysql> CREATE DATABASE cookbook CHARACTER SET UTF8;
```

le damos privilegios a todas las tablas de la tabla al usuario root

```
mysql> GRANT ALL PRIVILEGES ON cookbook.* TO root@localhost;
```

hacemos un flush

```
mysql> FLUSH PRIVILEGES;
```

Salimos de la consola mysql:

```
mysql> QUIT
```

Toda la información que tenemos en la ddbb de django no queremos perderla, por lo que ponemos esto en la terminal, en backend:

```
(env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend % python manage.py dumpdata --natural-foreign --natural-primary -e contenttypes -e auth.Permission --indent 4 > datadump.json
(env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend %
```

Y esto lo que hace es crearnos un datadump.json con toda la información (usuarios, objetos, permisos...)

```
ejercicio_entregable_3_django > backend > {} datadump.json > ...
163 {
164   "model": "recipe.recipe",
165   "pk": 5,
166   "fields": {
167     "title": "Ensalada de tomate y queso",
168     "steps": "Paso 1: Cortar el tomate y el queso, Paso 2: Poner en un
169     "ingredients": [
170       7,
171       8
172     ]
173   }
174 },
175 {
176   "model": "recipe.recipe",
177   "pk": 6,
178   "fields": {
179     "title": "Pizza de tomate y queso",
180     "steps": "Paso 1: Comprar una masa de pizza congelada, Paso 2: Pone
181     "ingredients": [
182       7,
183       8
184     ]
185   }
186 },
187 {
188   "model": "recipe.recipe",
189   "pk": 7,
```

Hay que asegurarse de que tenemos en el requirements.txt el "mysqlclient", si no lo tenemos lo añadimos y volvemos a instalarlo. Hacemos un pip list y vemos que nos aparece instalado, por lo que no tenemos que añadirlo de nuevo.

Ahora vamos al archivo backend/basicserver/settings.py y buscamos DATABASES y cambiamos la base de datos por defecto por la que vamos a usar ahora en mysql:

```
# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    # 'default': {
    #     'ENGINE': 'django.db.backends.sqlite3',
    #     'NAME': BASE_DIR / 'db.sqlite3',
    # }
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'cookbook',
        'USER': 'root',
        'PASSWORD': 'deb361990',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Y guardamos. Para que esto se haga efectivo debemos poner (dentro de backend):

```
● (env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, ingredient, recipe, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying ingredient.0001_initial... OK
  Applying recipe.0001_initial... OK
  Applying sessions.0001_initial... OK
○ (env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend %
```

Si volvemos a la terminal:

```
mysql> mysql -u root -p
mysql> use cookbook;
mysql> show tables;
```



```
[mysql> show tables;
+-----+
| Tables_in_cookbook |
+-----+
| auth_group          |
| auth_group_permissions |
| auth_permission     |
| auth_user           |
| auth_user_groups    |
| auth_user_user_permissions |
| django_admin_log    |
| django_content_type |
| django_migrations   |
| django_session      |
| ingredient_ingredient |
| recipe_recipe       |
| recipe_recipe_ingredients |
+-----+
13 rows in set (0,00 sec)
```

Están todas las tablas del proyecto, todos los schemas, pero si hacemos:

```
mysql> select * from ingredient_ingredient;
```

Aparece un empty set, esto significa que los datos no se han importado aun. Para ello volvemos a la terminal de VSCode:

```
Applying recipe.0001_initial... OK
Applying sessions.0001_initial... OK
● (env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend % python manage.py loaddata datadump.json
Installed 21 object(s) from 1 fixture(s)
○ (env) deborarubiosoliva@MacBook-Air-de-Debra-2 backend %
```

Entonces si ahora hacemos el mismo select de antes:

```
[mysql> select * from ingredient_ingredient;
Empty set (0,00 sec)

[mysql> select * from ingredient_ingredient;
+----+-----+-----+
| id | name  | quantity |
+----+-----+-----+
| 7  | Tomate | 100      |
| 8  | Queso  | 50       |
| 9  | Patata | 300      |
+----+-----+-----+
3 rows in set (0,01 sec)

mysql>
```

Y ya tenemos nuestra base de datos totalmente funcional igual que la teníamos antes.

Conclusiones y observaciones

El ejercicio ha sido muy completo, ha sido vital volver a ver los videos para comprender bien los procesos, pero me ha parecido más sencillo de entender e interiorizar.

Como mejoras propongo implementar mixins en los viewsets, poner restricciones a dichos viewsets y realizar más tests unitarios al modelo “recetas”.

Quizá este es el ejercicio que necesitábamos para terminar de comprender bien las conexiones con el backend, en concreto con la migración y uso de una base de datos mysql y no la que Django ofrece por defecto.

Seguiré desarrollando este proyecto como me recomendaste para mi portfolio personal.