Introducción

Este proyecto de cocina desarrollado con Django incluye diversas funcionalidades que permiten gestionar recetas, comentarios, listas de compras y desafíos culinarios. La estructura del proyecto se divide en varias aplicaciones: cocina y desafios, cada una con modelos, vistas y rutas propias. A través de una API REST, los usuarios pueden interactuar con los diferentes recursos disponibles, desde la creación de recetas hasta la participación en desafíos culinarios.

Estructura del Proyecto y código

- **Aplicación cocina:**

  Modelos: Receta, Comentario, ListaDeCompras

```python
class Receta(models.Model):
    nombre = models.CharField(max_length=255)
    ingredientes = models.TextField()
    instrucciones = models.TextField()
    tiempo_preparacion = models.IntegerField(help_text="Tiempo en minutos")
    categoria = models.CharField(max_length=100)
    imagen = models.ImageField(upload_to='imagenes_recetas/', null=True, blank=True)

    def __str__(self):
        return self.nombre

You, 17 hours ago | 1 author (You)
class Comentario(models.Model):      You, 17 hours ago • Primer paso práctica django hecha
    autor = models.CharField(max_length=100)
    contenido = models.TextField()
    fecha_publicacion = models.DateTimeField(auto_now_add=True)
    receta = models.ForeignKey(Receta, related_name='comentarios', on_delete=models.CASCADE)

    def __str__(self):
        return f'Comentario de {self.autor} en {self.receta.nombre}'

You, 17 hours ago | 1 author (You)
class ListaDeCompras(models.Model):
    nombre = models.CharField(max_length=255)
    recetas = models.ManyToManyField(Receta)
    fecha_creacion = models.DateTimeField(auto_now_add=True)

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

  Serializadores: Serializadores para cada modelo.

```python
You, 17 hours ago | 1 author (You)
class RecetaSerializer(serializers.ModelSerializer):
    You, 17 hours ago | 1 author (You)
    class Meta:
        model = Receta
        fields = '__all__'

You, 17 hours ago | 1 author (You)
class ComentarioSerializer(serializers.ModelSerializer):
    You, 17 hours ago | 1 author (You)
    class Meta:
        model = Comentario
        fields = '__all__'

You, 17 hours ago | 1 author (You)
class ListaDeComprasSerializer(serializers.ModelSerializer):
    You, 17 hours ago | 1 author (You)
    class Meta:
        model = ListaDeCompras
        fields = '__all__'
```

  Vistas Genéricas: Vistas para listar, crear, actualizar y eliminar recetas, comentarios y listas de compras.

```python
# Vistas Genéricas para Receta
class RecetaList(generics.ListCreateAPIView):
    queryset = Receta.objects.all()
    serializer_class = RecetaSerializer

class RecetaDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Receta.objects.all()
    serializer_class = RecetaSerializer

# Vistas Genéricas para Comentario
class ComentarioList(generics.ListCreateAPIView):
    queryset = Comentario.objects.all()
    serializer_class = ComentarioSerializer
        You, 17 hours ago • Primer paso práctica django hecha
class ComentarioDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Comentario.objects.all()
    serializer_class = ComentarioSerializer

# Vistas Genéricas para ListaDeCompras
class ListaDeComprasList(generics.ListCreateAPIView):
    queryset = ListaDeCompras.objects.all()
    serializer_class = ListaDeComprasSerializer

class ListaDeComprasDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = ListaDeCompras.objects.all()
    serializer_class = ListaDeComprasSerializer
```

Vistas Personalizadas: Vista para listar comentarios de una receta específica.

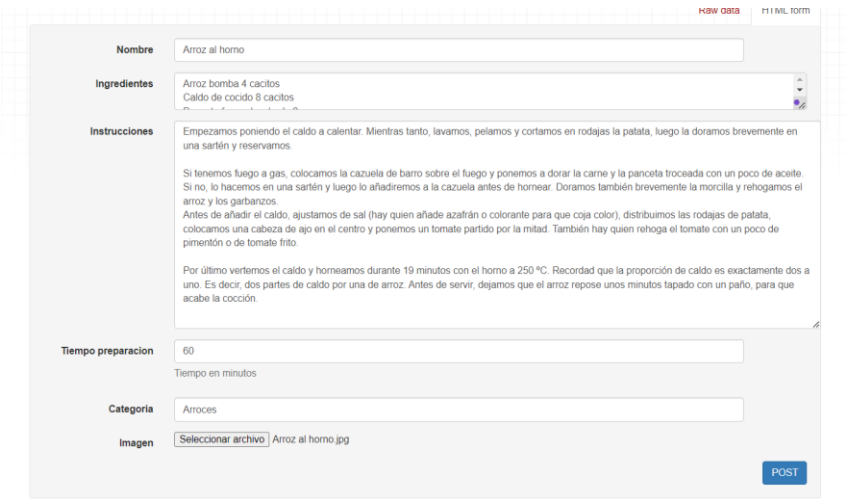
```python
# api_view Propia para listar comentarios de una receta específica
@api_view(['GET'])
def comentarios_de_receta(request, receta_id):
    try:
        receta = Receta.objects.get(id=receta_id)
    except Receta.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)

    comentarios = Comentario.objects.filter(receta=receta)
    serializer = ComentarioSerializer(comentarios, many=True)
    return Response(serializer.data)
```

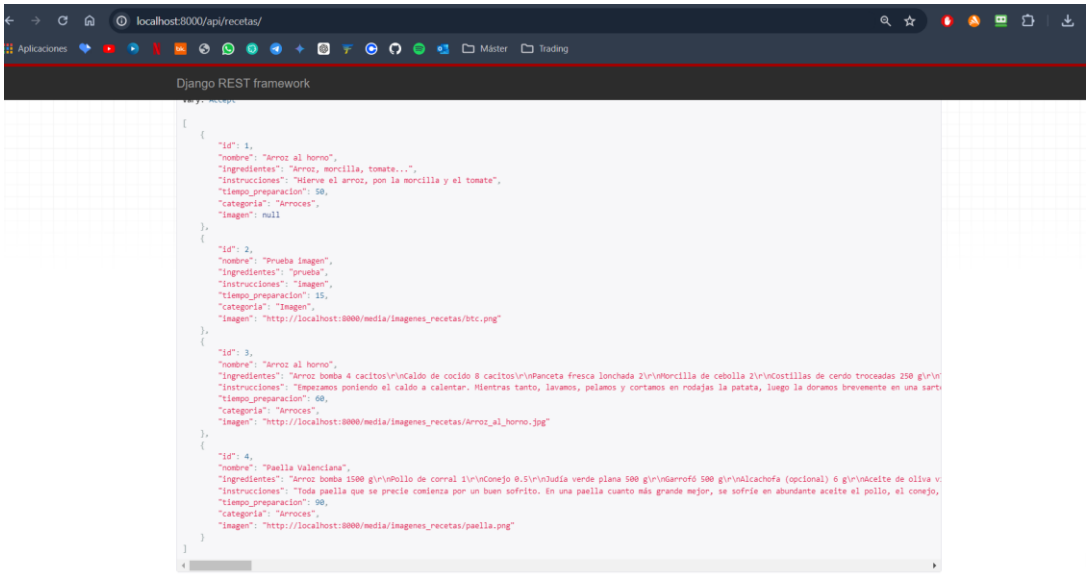
URLs:

Listar y crear recetas: http://localhost:8000/api/recetas/

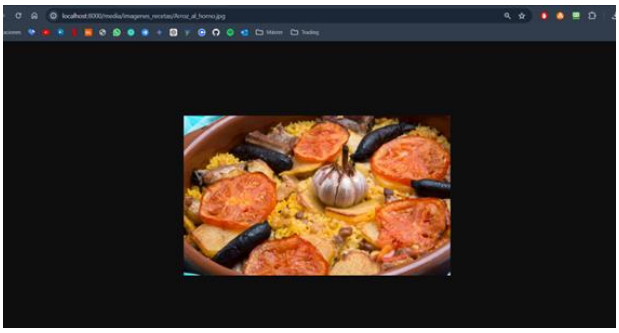
Crear con el método POST: (2 ejemplos)

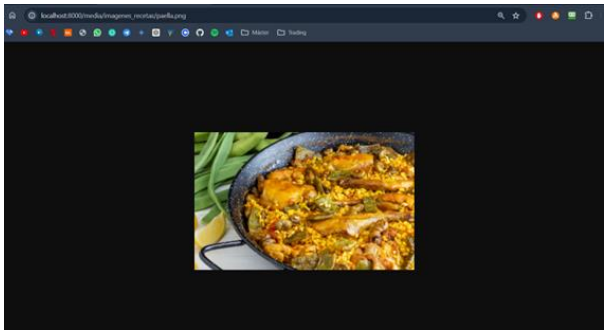


Listar las recetas con el método GET:



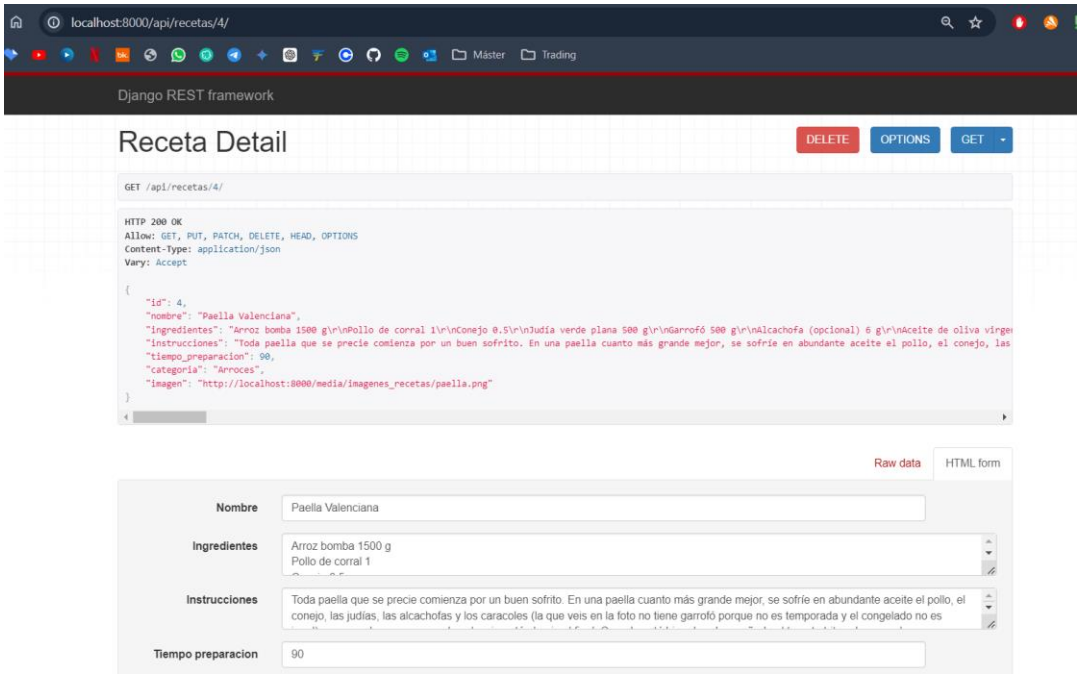
Así como mostrar las imágenes adjuntas:



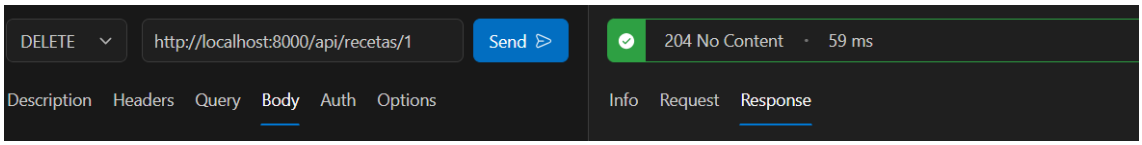


Detallar, actualizar y eliminar una receta: http://localhost:8000/api/recetas/<id>/

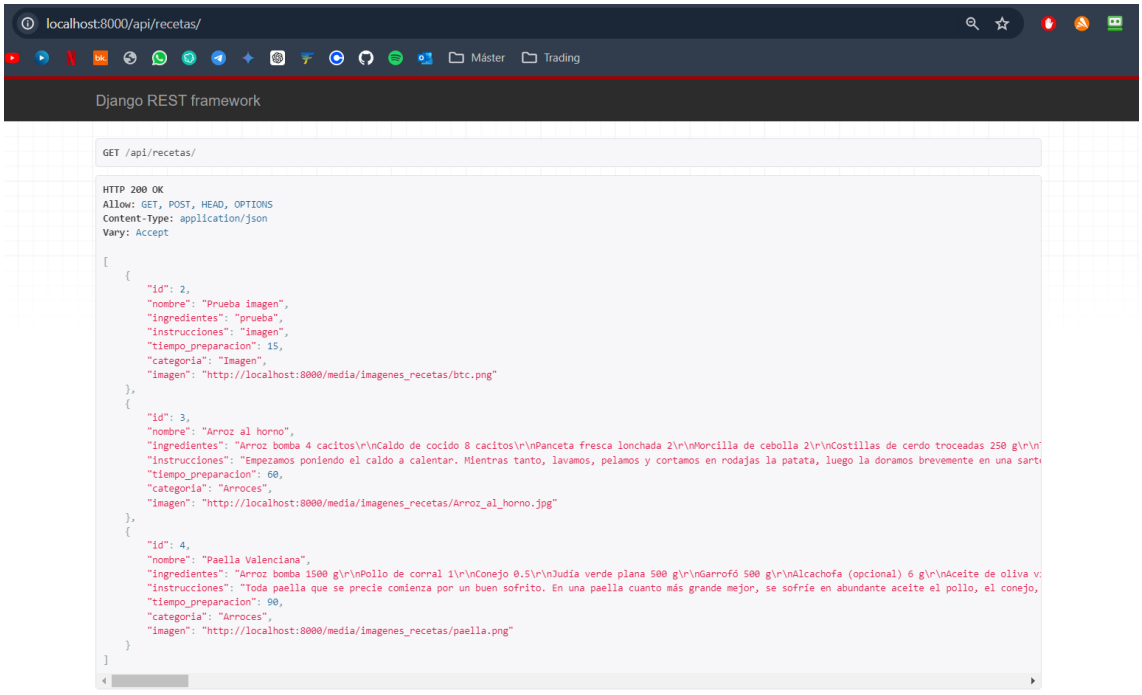
Detalle de una única receta:



Como había 2 recetas de arroz al horno, hemos borrado la primera que era de prueba

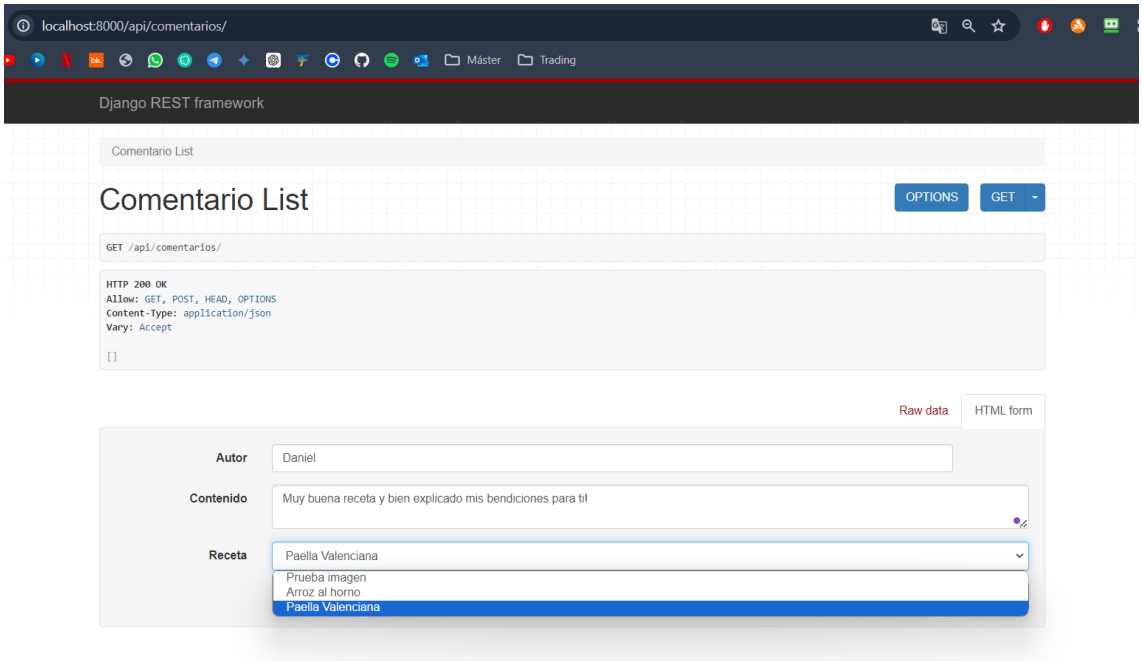


Hecho el delete, podemos observar que ya no sale cuando hacemos el método GET:

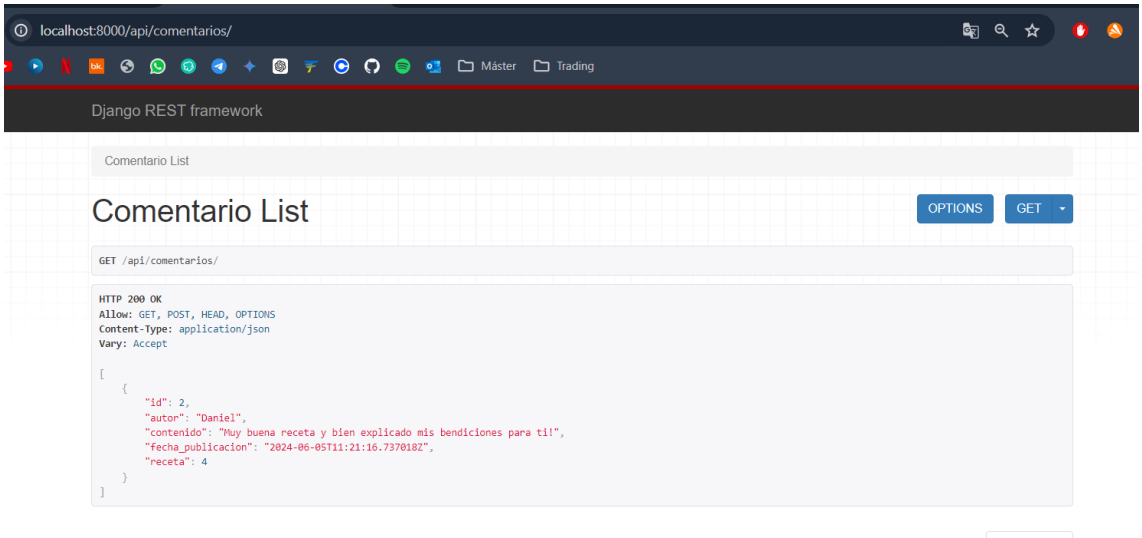


Listar y crear comentarios: http://localhost:8000/api/comentarios/

Creamos el comentario con el método POST, relacionado con la receta de la paella valenciana:

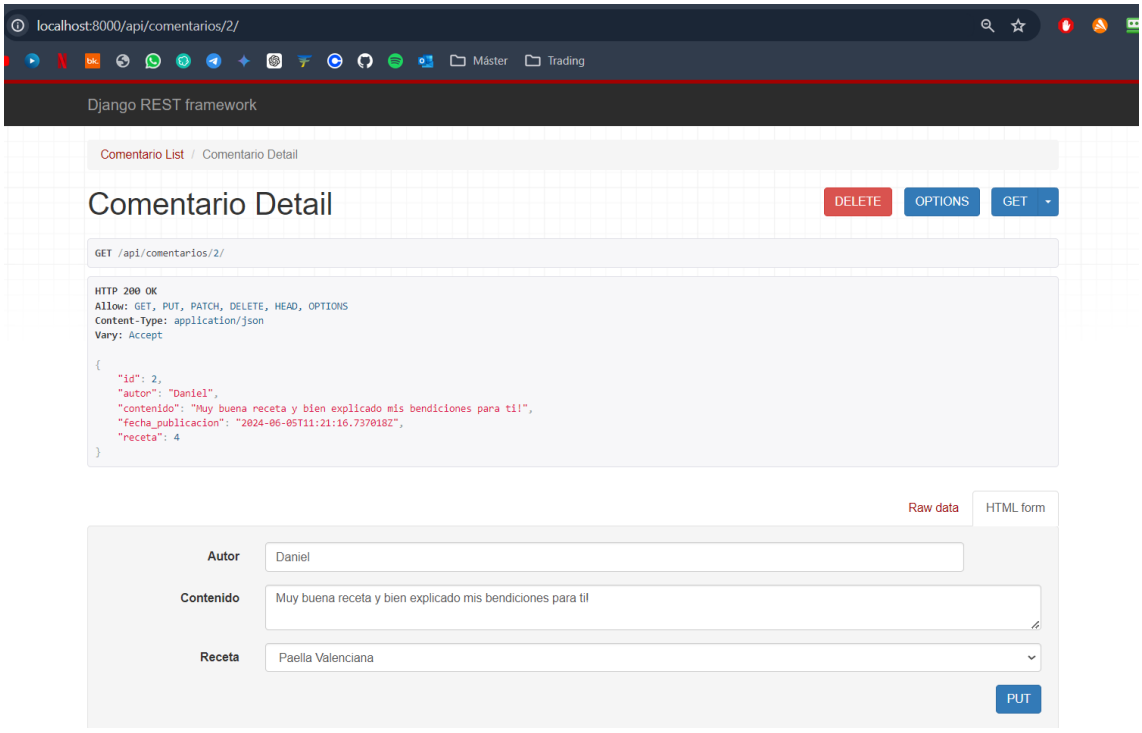


Y lo mostramos con el método GET:

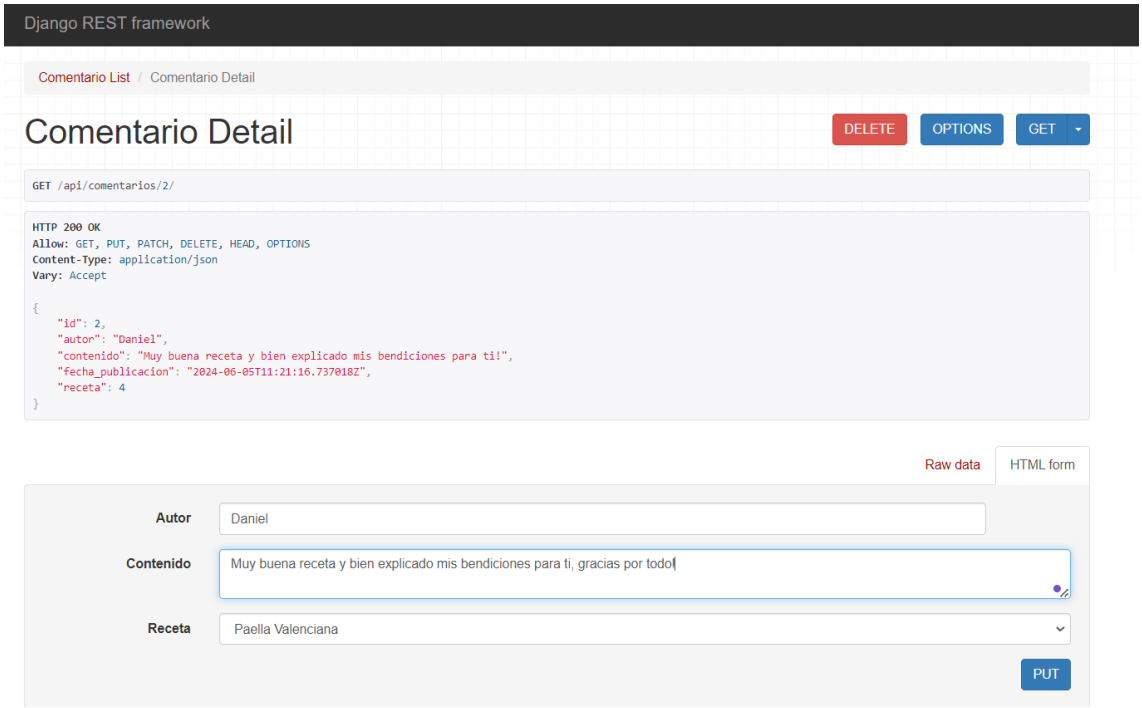


Detallar, actualizar y eliminar un comentario:
http://localhost:8000/api/comentarios/<id>/

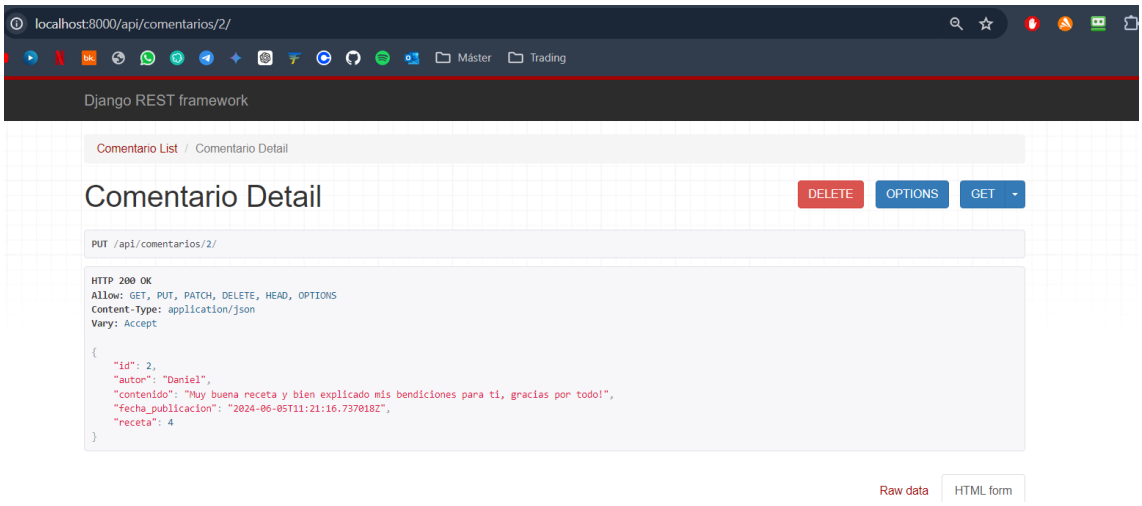
Por otro lado, detallamos los comentarios:



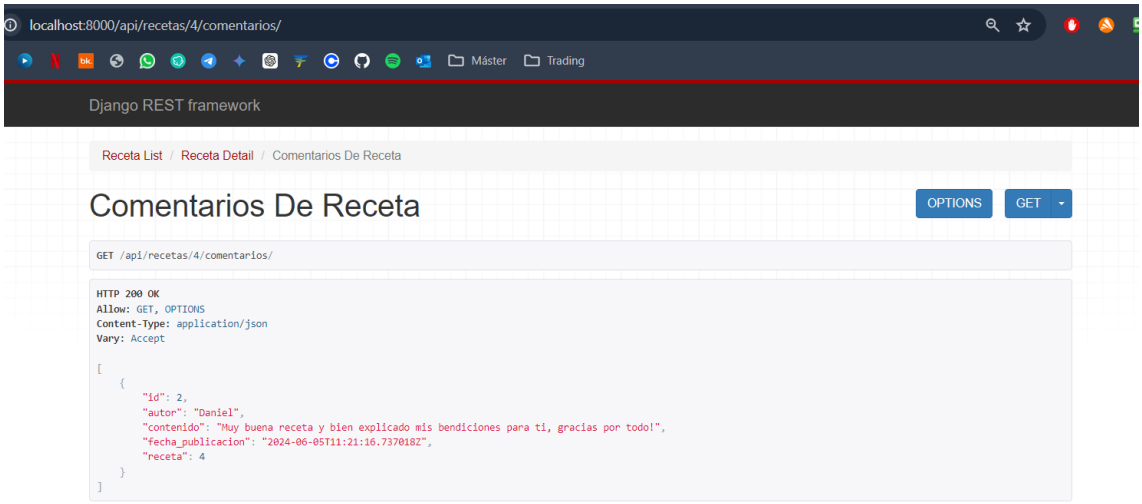
Y hacemos una modificación del contenido del comentario con el método PUT



Modificado correctamente:

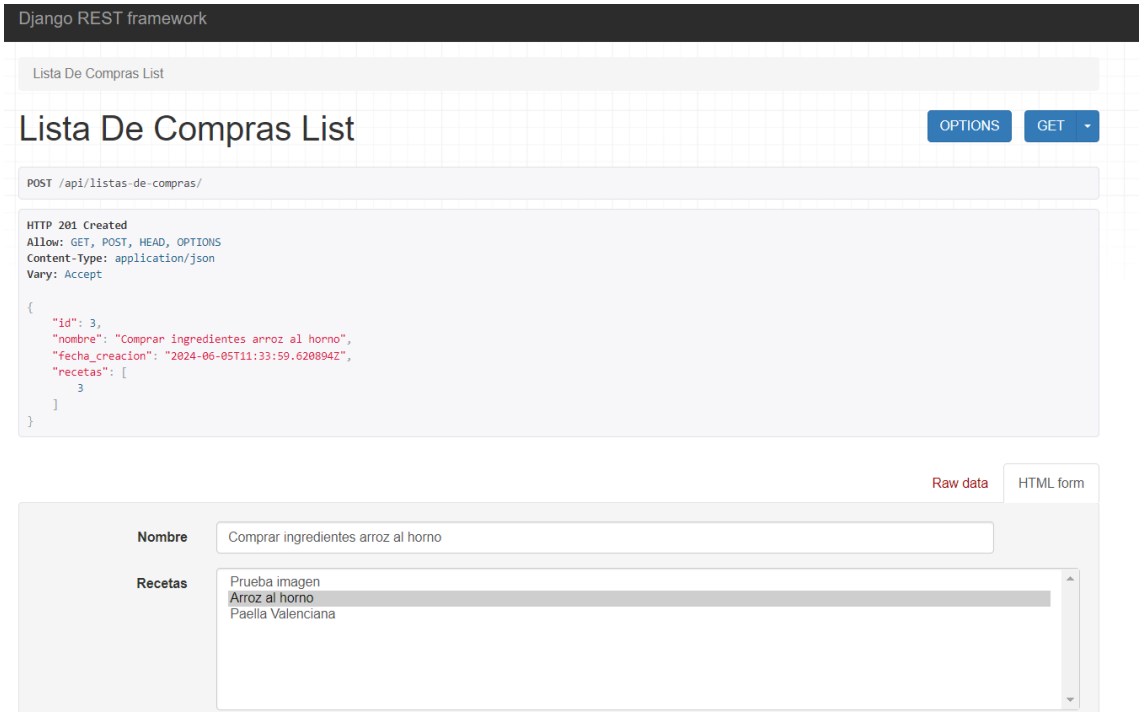


Listar comentarios de una receta específica:
http://localhost:8000/api/recetas/<receta_id>/comentarios/



Listar y crear listas de compras: http://localhost:8000/api/listas-de-compras/

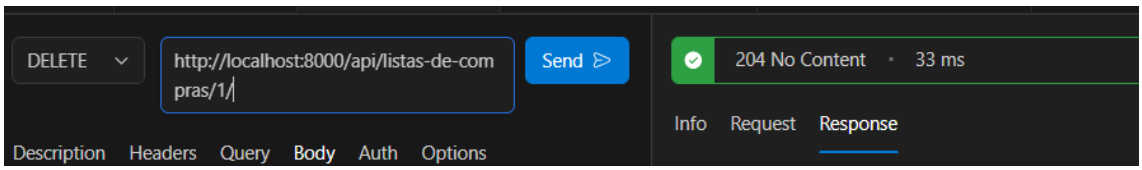
Creamos una lista de la compra para la receta del arroz al horno



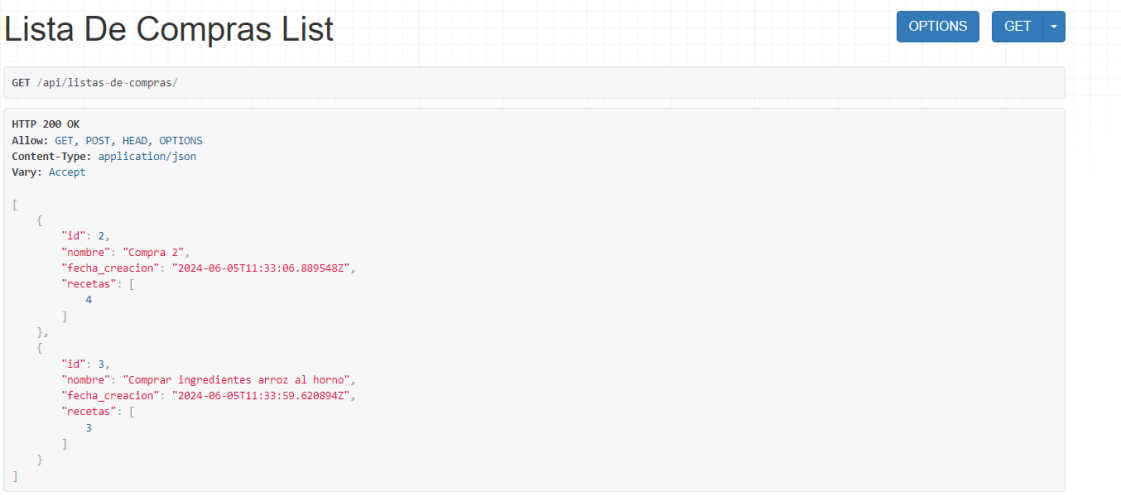
Detallar, actualizar y eliminar una lista de compras:
http://localhost:8000/api/listas-de-compras/<id>/

Por ejemplo, el primer caso de lista de compra lo cree vacío a modo de test:

Lo eliminamos de la siguiente manera:



Vemos que ya se ha eliminado correctamente:



- **Aplicación desafios:**

Modelos: Desafio, Participación

```python
class Desafio(models.Model):
    titulo = models.CharField(max_length=255)
    descripcion = models.TextField()
    fecha_inicio = models.DateField()
    fecha_fin = models.DateField()

    def __str__(self):
        return self.titulo

class Participacion(models.Model):
    receta = models.ForeignKey(Receta, on_delete=models.CASCADE)
    desafio = models.ForeignKey(Desafio, related_name='participaciones', on_delete=models.CASCADE)
    fecha_participacion = models.DateTimeField(auto_now_add=True)
    votos = models.PositiveIntegerField(default=0)

    def __str__(self):
        return f'Participación de {self.receta.nombre} en {self.desafio.titulo}'
```

Serializadores: Serializadores para cada modelo.

```python
class DesafioSerializer(serializers.ModelSerializer):
    class Meta:
        model = Desafio
        fields = '__all__'

class ParticipacionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Participacion
        fields = '__all__'
```

Vistas Genéricas: Vistas para listar, crear, actualizar y eliminar desafíos y participaciones.

```python
# Vistas genéricas Desafio y Participacion
class DesafioList(generics.ListCreateAPIView):
    queryset = Desafio.objects.all()
    serializer_class = DesafioSerializer

class DesafioDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Desafio.objects.all()
    serializer_class = DesafioSerializer

class ParticipacionList(generics.ListCreateAPIView):
    queryset = Participacion.objects.all()
    serializer_class = ParticipacionSerializer

class ParticipacionDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Participacion.objects.all()
    serializer_class = ParticipacionSerializer
```

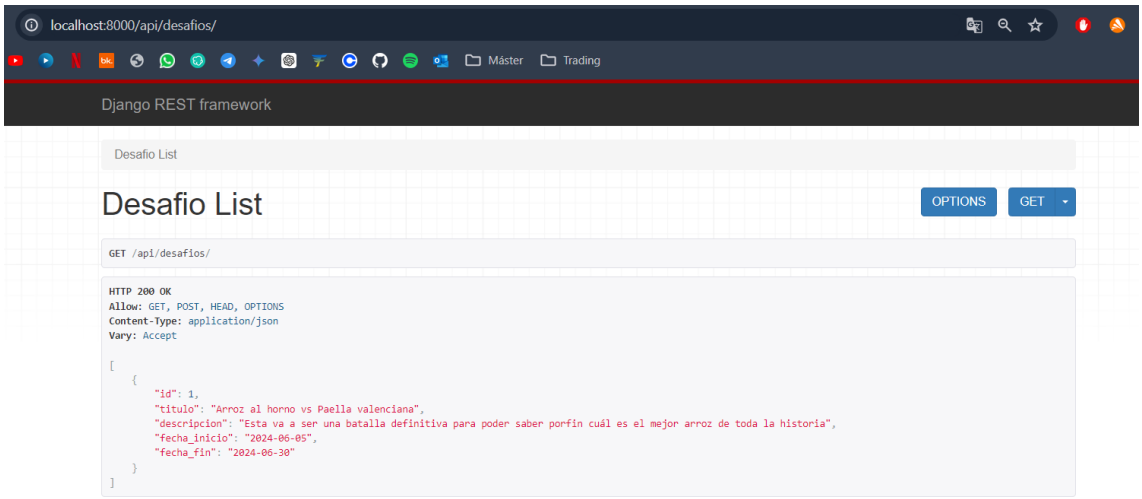
Vistas Personalizadas: Vista para votar por una participación.

```python
# Vista de api_view para votar en participaciones
@api_view(['POST'])
def votar_participacion(request, participacion_id):
    try:
        participacion = Participacion.objects.get(id=participacion_id)
        participacion.votos += 1
        participacion.save()
        return Response({'status': 'voto registrado'}, status=status.HTTP_200_OK)
    except Participacion.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
```

URLs:

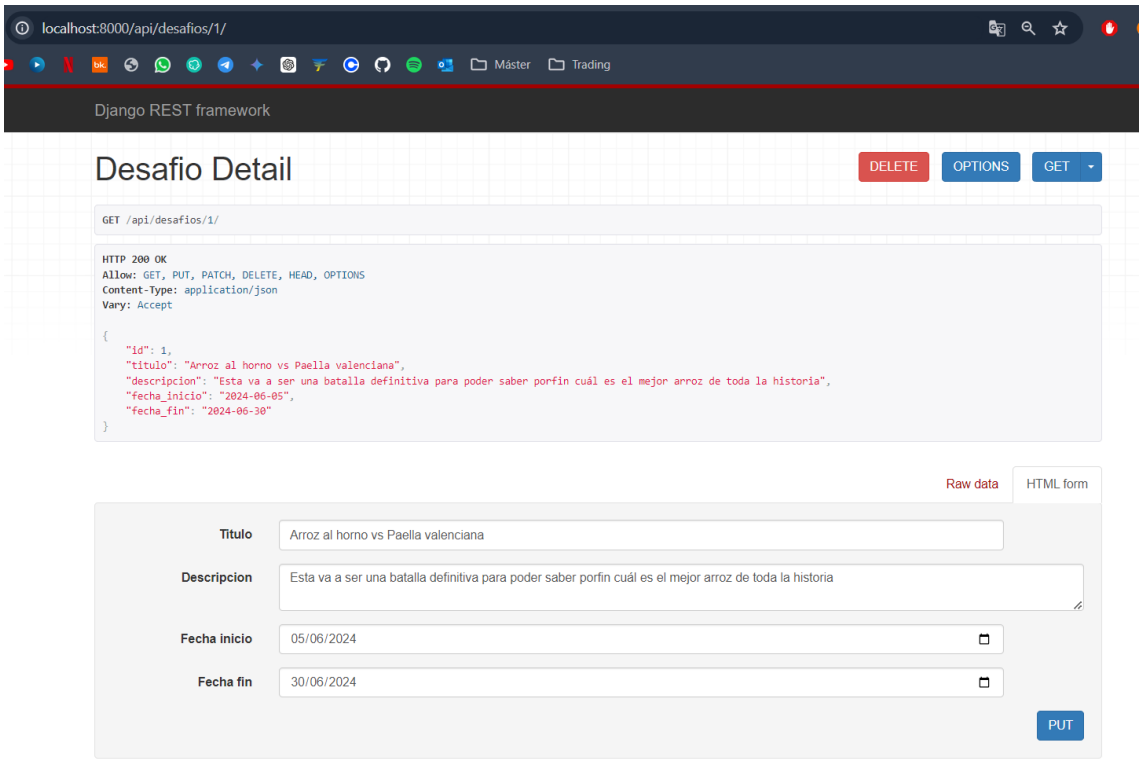
Listar y crear desafíos: http://localhost:8000/api/desafios/

Previamente hemos creado un desafío con el método POST y lo mostramos con el método GET:



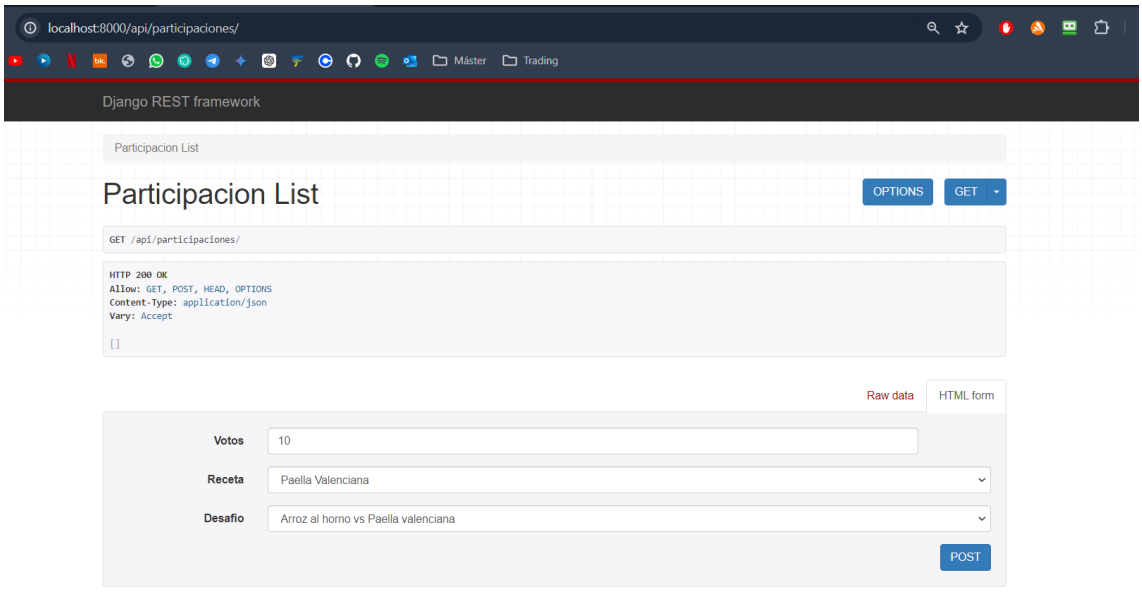
Detallar, actualizar y eliminar un desafío: http://localhost:8000/api/desafios/<id>/

Mostramos detalladamente el desafío por la búsqueda por id con el método GET:

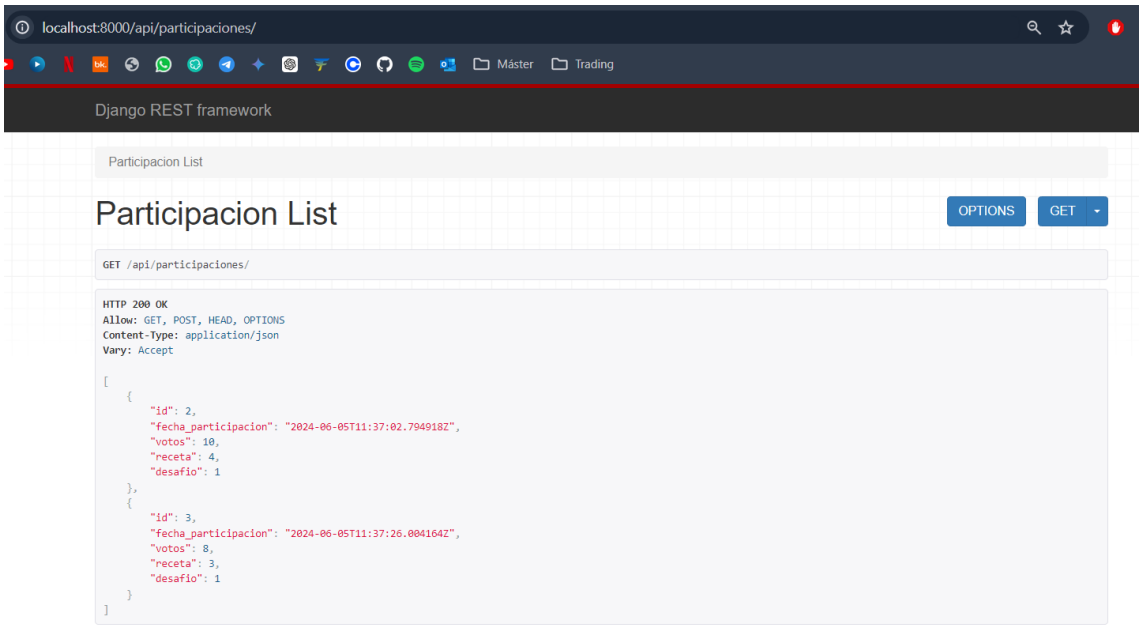


Listar y crear participaciones: http://localhost:8000/api/participaciones/

Creamos participaciones con el método POST:

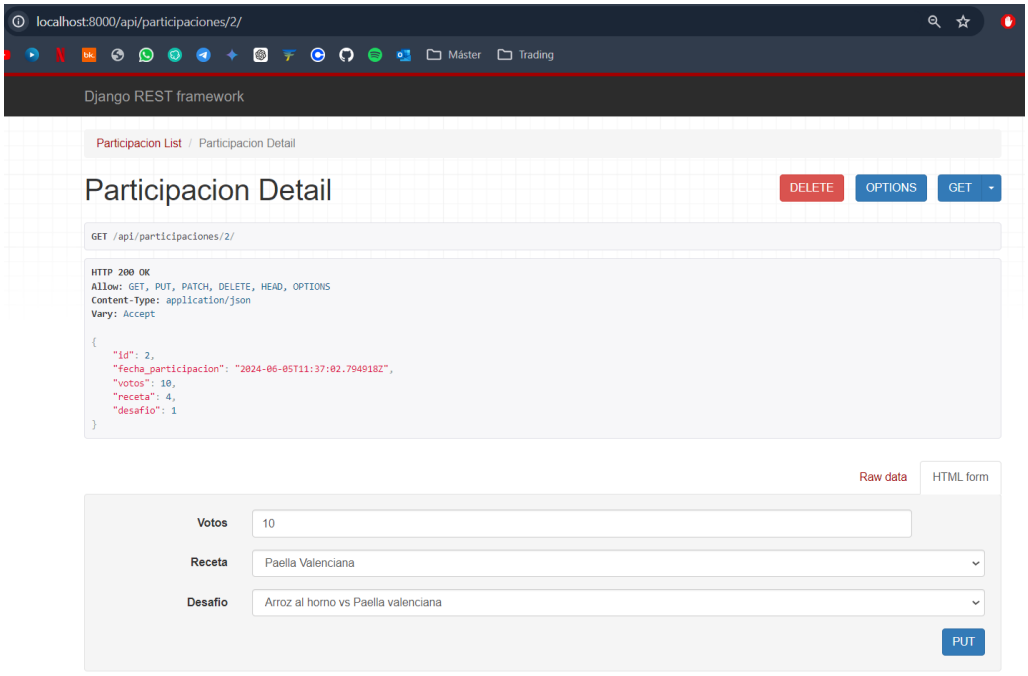


Mostramos el listado de las participaciones al desafío.



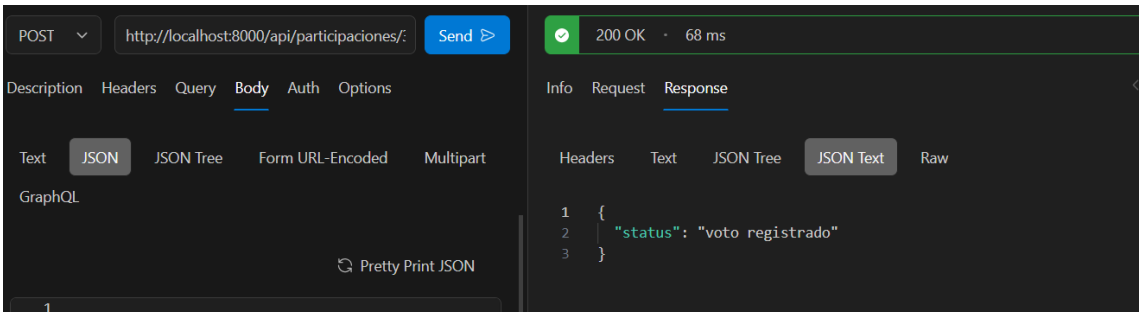
Detallar, actualizar y eliminar una participación:
http://localhost:8000/api/participaciones/<id>/

Mostramos detalladamente la participación en el desafío:



Votar por una participación:
http://localhost:8000/api/participaciones/<participacion_id>/votar/

Gracias al método POST y el api_view personalizado que hemos creado, podemos votar a una participación indicada, como es este caso:



Como vemos, la participación 3 aumentaría en 1 su voto

Conclusiones

Este proyecto de cocina ofrece una plataforma versátil para gestionar recetas, comentarios, listas de compras y desafíos culinarios. La implementación de vistas genéricas y personalizadas, junto con una API REST robusta, permite una interacción fluida y eficiente con los datos. Las funcionalidades añadidas, como la votación en desafíos y la gestión de listas de compras, enriquecen la experiencia del usuario, haciendo de esta aplicación una herramienta completa para los entusiastas de la cocina.

La estructura modular del proyecto facilita su escalabilidad y mantenimiento, permitiendo la incorporación de nuevas características de manera ordenada y eficiente. El uso de herramientas como Postman y cURL para probar y consumir la API asegura que las interacciones sean claras y manejables, tanto para desarrolladores como para usuarios finales.

Enlace del código

https://github.com/danisentamans/Entregables-BACKEND/tree/main/Entregable3