Task Management REST API with Nested Comments and Custom Permissions

AUTHOR : VIGNESH CHINTHAKUNTLA

Github : https://github.com/Ch-Vignesh

Description:

- Create a Django project called "TaskManagerAPI."
- Make a "tasks" app with two models:
  - Task: Title, Description, Status ("Pending," "Completed"), Owner (User).
  - Comment: Text, Parent, Author (User), Created Date.
- Build these API endpoints:
  - POST /tasks/ (create a task, JWT required).
  - GET /tasks/ (list user's tasks with nested comments, JWT required).
  - PUT /tasks/{id}/ (edit a task, JWT required, author only).
  - POST /comments/ (add a comment, JWT required).
  - PUT /comments/{id}/ (edit a comment, JWT required, author only).

- Custom permissions: Only task owners edit tasks; only comment authors edit comments.

Deliverable: A secure task management API with comments

Approach

Dependences :

```
pip install django djangorestframework djangorestframework-simplejwt
```

Creating Django project :

```
django-admin startproject TaskManagerAPI

cd TaskManagerAPI

python manage.py runserver

python manage.py startapp tasks
```

project file structure

```
Directory structure:
└── ch-vignesh-task_manager_nested_comments/
├── manage.py
├── TaskManagerAPI/
│   ├── init.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── tasks/
├── init.py
├── admin.py
├── apps.py
├── models.py
├── permissions.py
├── serializers.py
├── signals.py
├── tests.py
├── urls.py
├── views.py
└── migrations/
├── 0001_initial.py
└── init.py
```

update pages one by one

tasks/models.py

```
from django.db import models
from django.conf import settings

class Task(models.Model):
    """
    Task model representing a to-do item with a title, description, status, and ov

    Attributes:
    - title (str): The title of the task.
    - description (str): A detailed description of the task.
    - status (str): The current status of the task, either "Pending" or "Completec
    - owner (ForeignKey): The user who owns the task.

    Methods:
    - __str__(): Returns the title of the task as its string representation.
    """

    STATUS_CHOICES = (
        ('Pending', 'Pending'),
        ('Completed', 'Completed'),
    )
    title = models.CharField(max_length=255)
    description = models.TextField()
    status = models.CharField(max_length=10, choices=STATUS_CHOICES, def
    owner = models.ForeignKey(settings.AUTH_USER_MODEL, related_name='t
    def __str__(self):
        return self.title

class Comment(models.Model):

    """
    Comment model representing user comments on tasks.

    Attributes:
    - task (ForeignKey): The task to which the comment belongs.
    - text (str): The content of the comment.
    - parent (ForeignKey): Optional reference to a parent comment (for nested r
    - author (ForeignKey): The user who created the comment.
    - created_date (DateTimeField): The timestamp when the comment was cre

    Methods:
    - __str__(): Returns a readable string representation of the comment.
    """

    task = models.ForeignKey(Task, related_name='comments', on_delete=moc
    text = models.TextField()
    parent = models.ForeignKey("self", on_delete=models.CASCADE, null=True,
    author = models.ForeignKey(settings.AUTH_USER_MODEL, related_name='
    created_date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'Comment by {self.author} on {self.task}'
```

tasks/permissions.py

```
# tasks/permissions.py
from rest_framework import permissions

class IsOwnerOrReadOnlyTask(permissions.BasePermission):
    """
    Only the owner of the task can edit it.
    """
    """
    Check if the user has permission to perform an action on a specific object.

    This function is used in Django REST framework's permission classes to de
    whether a user has permission to perform a specific action on a given objec

    Parameters:
    - request (Request): The incoming request object containing information ab
    - view (View): The view object that is handling the request.
    - obj (Model): The specific object on which the user is trying to perform the

    Returns:
    - bool: True if the user has permission to perform the action on the object, F
      - For safe HTTP methods (GET, HEAD, OPTIONS), permission is always g
      - For other HTTP methods, permission is granted only if the user is the ov
    """
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        return obj.owner == request.user

class IsAuthorOrReadOnlyComment(permissions.BasePermission):
    """
    Only the author of the comment can edit it.
    """
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        return obj.author == request.user
```

tasks/serializers.py

```
from rest_framework import serializers
from django.contrib.auth.models import User
from .models import Task, Comment
```

```
# comment serialization
class CommentSerializer(serializers.ModelSerializer):
    replies = serializers.SerializerMethodField(read_only=True)
    author = serializers.StringRelatedField(read_only=True)

    class Meta:
        model = Comment
        fields = ['id', 'text', 'parent', 'author', 'created_date', 'replies']
        read_only_fields = ['author', 'created_date', 'replies']

    def get_replies(self, obj):
        if obj.replies.exists():
            return CommentSerializer(obj.replies.all(), many=True).data
        return []

# task serializer
class TaskSerializer(serializers.ModelSerializer):
    owner = serializers.StringRelatedField(read_only=True)
    comments = CommentSerializer(many=True, read_only=True)

    class Meta:
        model = Task
        fields = ['id', 'title', 'description', 'status', 'owner', 'comments']
        read_only_fields = ['owner', 'comments']

# User Registration Serializer
class UserRegisterSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True, required=True, style={'in
    password2 = serializers.CharField(write_only=True, required=True, label="C

    class Meta:
        model = User
        fields = ['username', 'email', 'password', 'password2']

    def validate(self, data):
        if data['password'] != data['password2']:
            raise serializers.ValidationError("Passwords do not match.")
        return data
    def create(self, validated_data):
        validated_data.pop('password2')
        user = User.objects.create_user(**validated_data)
        return user
```

tasks/signals.py

```
from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver
from rest_framework.authtoken.models import Token
from django.contrib.auth.models import User

@receiver(post_save, sender=User)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    """
    This function creates an authentication token for a new user when they are

    Parameters:
    sender (class): The model class sending the signal. In this case, it's the Use
    instance (User): The instance of the User model that triggered the signal.
    created (bool): A boolean indicating whether the instance was created.
    kwargs (dict): Additional keyword arguments passed to the signal handler.

    Returns:
    None
    """
    if created:
        Token.objects.create(user=instance)
```

tasks/urls.py

```
from django.urls import path
from .views import TaskListCreateView, TaskUpdateView, CommentCreateVie


urlpatterns = [
    path('tasks/', TaskListCreateView.as_view(), name='task-list-create'),
    path('tasks/<int:pk>/', TaskUpdateView.as_view(), name='task-update'),
    path('comments/', CommentCreateView.as_view(), name='comment-create'
    path('comments/<int:pk>/', CommentUpdateView.as_view(), name='comme
]
```

tasks/views.py

```

from rest_framework import generics, permissions, status, serializers
from rest_framework.response import Response
from django.contrib.auth.models import User
from .models import Task, Comment
from .serializers import TaskSerializer, CommentSerializer, UserRegisterSeriali
from .permissions import IsOwnerOrReadOnlyTask, IsAuthorOrReadOnlyComm

# Registration View
class RegisterView(generics.CreateAPIView):
    queryset = User.objects.all()
    serializer_class = UserRegisterSerializer
    permission_classes = [permissions.AllowAny]

# tasks
class TaskListCreateView(generics.ListCreateAPIView):
    """
    API view to list all tasks or create a new task.

    - GET: Returns a list of all tasks (optionally filtered by the logged-in user).
    - POST: Creates a new task with the logged-in user as the owner.

    Permissions:
```

```
    - Only authenticated users can access this view.
    """
    serializer_class = TaskSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        # return Task.objects.filter(owner=self.request.user)
        # uncomment this if you want to get user specific tasks
        return Task.objects.all()

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)

class TaskUpdateView(generics.RetrieveUpdateDestroyAPIView):
    """
    API view to retrieve, update, or delete a task.

    - GET: Retrieves a single task by its ID.
    - PUT/PATCH: Updates the task (only allowed for the task owner).
    - DELETE: Deletes the task (only allowed for the task owner).

    Permissions:
    - Only authenticated users can access.
    - Only the owner of the task can update or delete it.
    """
    queryset = Task.objects.all()
    serializer_class = TaskSerializer
    permission_classes = [permissions.IsAuthenticated, IsOwnerOrReadOnlyTa

# Comments

class CommentCreateView(generics.CreateAPIView):
    serializer_class = CommentSerializer
    permission_classes = [permissions.IsAuthenticated]

    def perform_create(self, serializer):
        """
        Saves a new comment instance with the current user as the author and th
        Parameters:
        serializer (CommentSerializer): The serializer instance containing the vali

        Raises:
        serializers.ValidationError: If the 'task' field is missing from the request da

        Returns:
        None: The function does not return a value. It saves the comment instanc
        """
        task_id = self.request.data.get("task")  # Get task_id from request
        if not task_id:
            raise serializers.ValidationError({"task": "This field is required."})

        try:
            task = Task.objects.get(id=task_id)
        except Task.DoesNotExist:
            raise serializers.ValidationError({"task": "Task not found."})

        serializer.save(author=self.request.user, task=task)  # Ensure task is assi


class CommentUpdateView(generics.RetrieveUpdateDestroyAPIView):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer
    permission_classes = [permissions.IsAuthenticated, IsAuthorOrReadOnlyCo
```

tasks/app.py

```
from django.apps import AppConfig

class TasksConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'tasks'
    def ready(self):
        import tasks.signals
```

TaskManagerAPI (project folder)

settings.py

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'tasks'
]
```

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
}
```

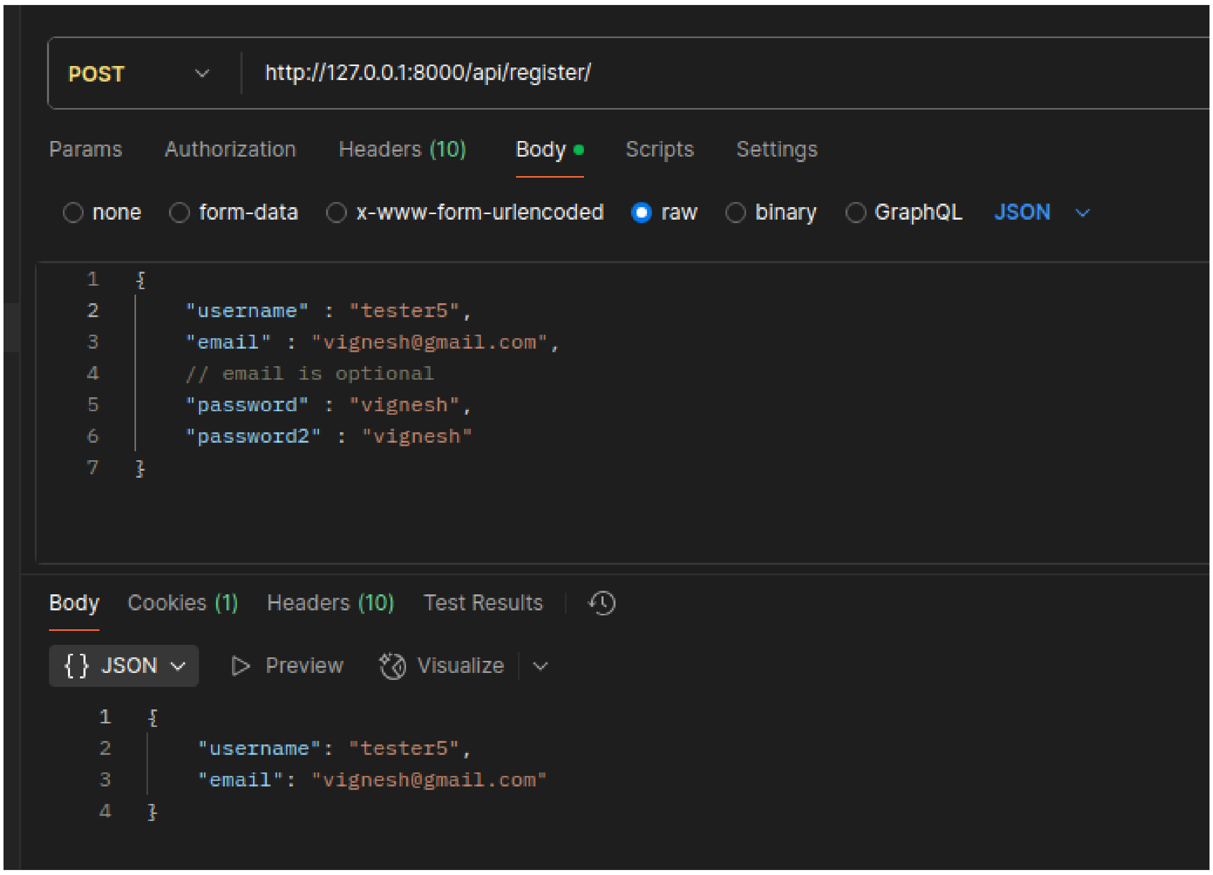
TaskManagerAPI/urls.py

```
from django.contrib import admin
from django.urls import path, include
from tasks.views import RegisterView

from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefr
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('tasks.urls')),
    # JWT endpoints
    path('api/register/', RegisterView.as_view(), name='auth_register'),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresl
]
```

POSTMAN : TESTING THE API



POST http://127.0.0.1:8000/api/register/

```
{
    "username" : "tester5",
    "email" : "vignesh@gmail.com",
    // email is optional
    "password" : "vignesh",
    "password2" : "vignesh"
}
```

Response Body (JSON):

```
{
    "username": "tester5",
    "email": "vignesh@gmail.com"
}
```

STEPS TO AUTOMATE THE ACCESS TOKEN

step 1

create a new environment -

create 3 variables :

1 access_token

`2 refresh_token`

`3 expiry_time`

`keep values empty`

`in collections, on top right select the environment that you created for the token`

`now in collections`

`create a new api call, in that POST http://127.0.0.1:8000/api/token/`

`go to authorization - in the place of token keep {{access_token}}`

`go to body, enter user_id and password`

```
{
"username": "wac",
"password": "web"
}
```

`in scripts`

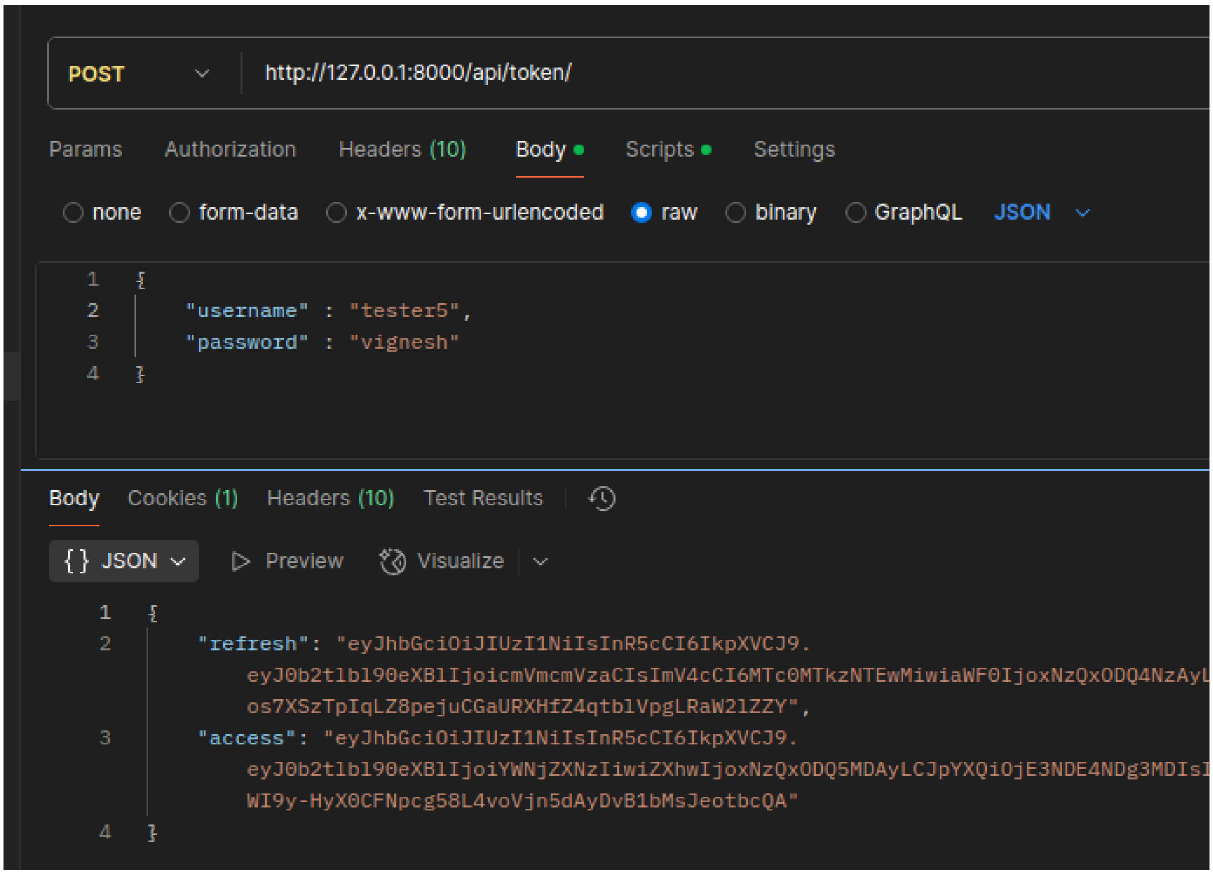
`add the following JSON script`

`for Post-res:`

```
var response = pm.response.json();
if (response.access) {
pm.environment.set("access_token", response.access);
}
if (response.refresh) {
pm.environment.set("refresh_token", response.refresh);
}
```

`click on SEND, it will create refresh key and access key`

login api



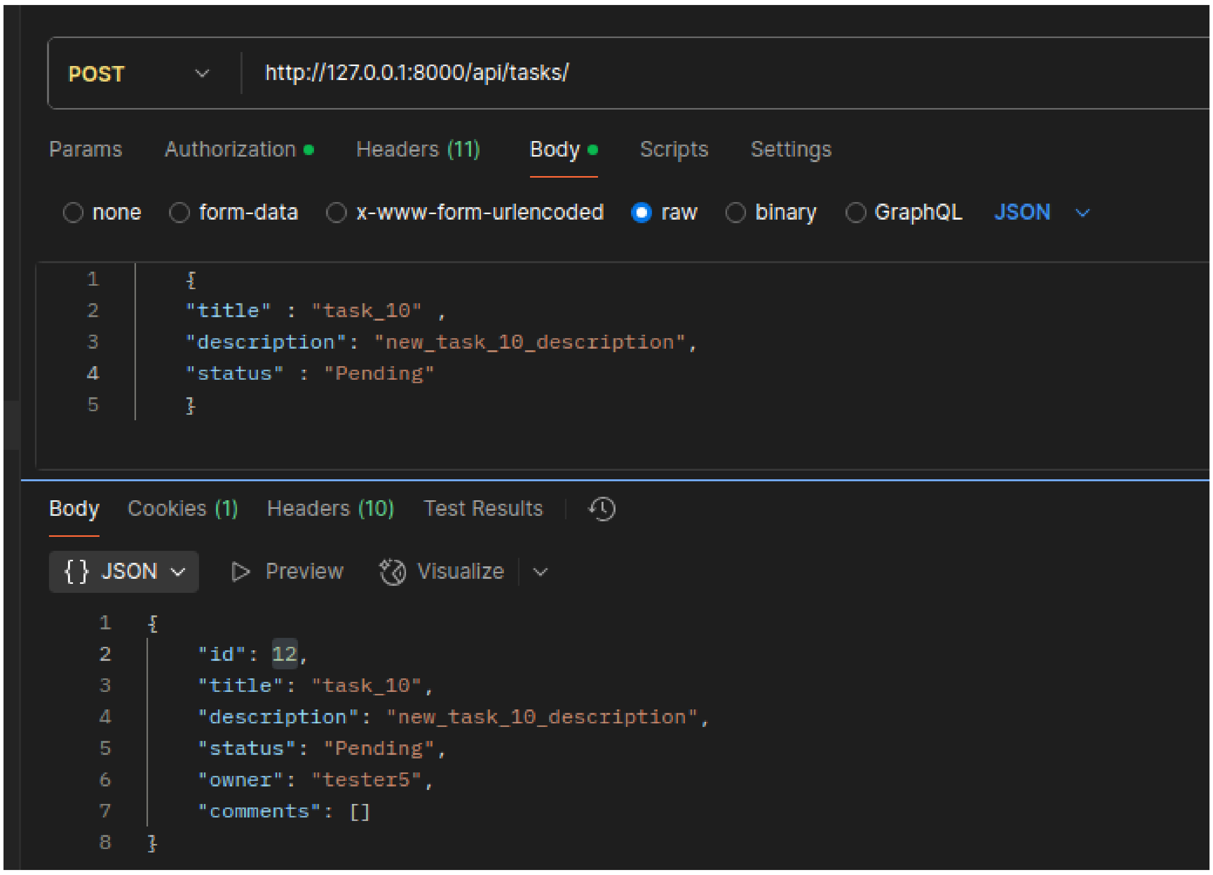
POST http://127.0.0.1:8000/api/token/

Body: [✓ raw] JSON

```
{
    "username" : "tester5",
    "password" : "vignesh"
}
```

```
{
    "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTc0MTkzNTEwMiwiaWF0IjoxNzQxODQ4NzAyL
        os7XSzTpIqLZ8pejuCGaURXHfZ4qtblVpgLRaW2lZZY",
    "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQxODQ5MDAyLCJpYXQiOjE3NDE4NDg3MDIsI
        WI9y-HyX0CFNpcg58L4voVjn5dAyDvB1bMsJeotbcQA"
}
```

post a task



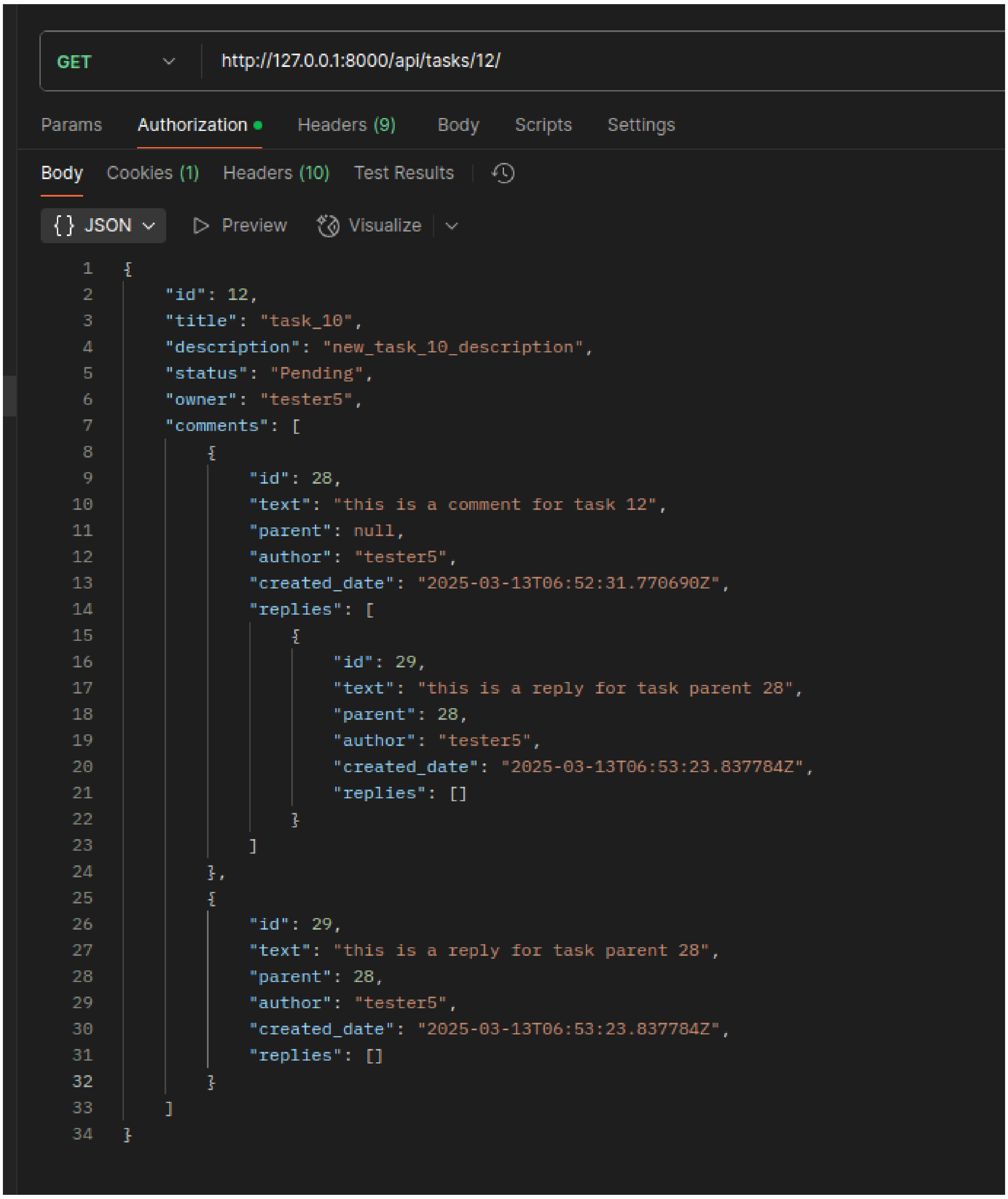
POST http://127.0.0.1:8000/api/tasks/

Body: [✓ raw] JSON

```
{
"title" : "task_10" ,
"description": "new_task_10_description",
"status" : "Pending"
}
```

```
{
    "id": 12,
    "title": "task_10",
    "description": "new_task_10_description",
    "status": "Pending",
    "owner": "tester5",
    "comments": []
}
```

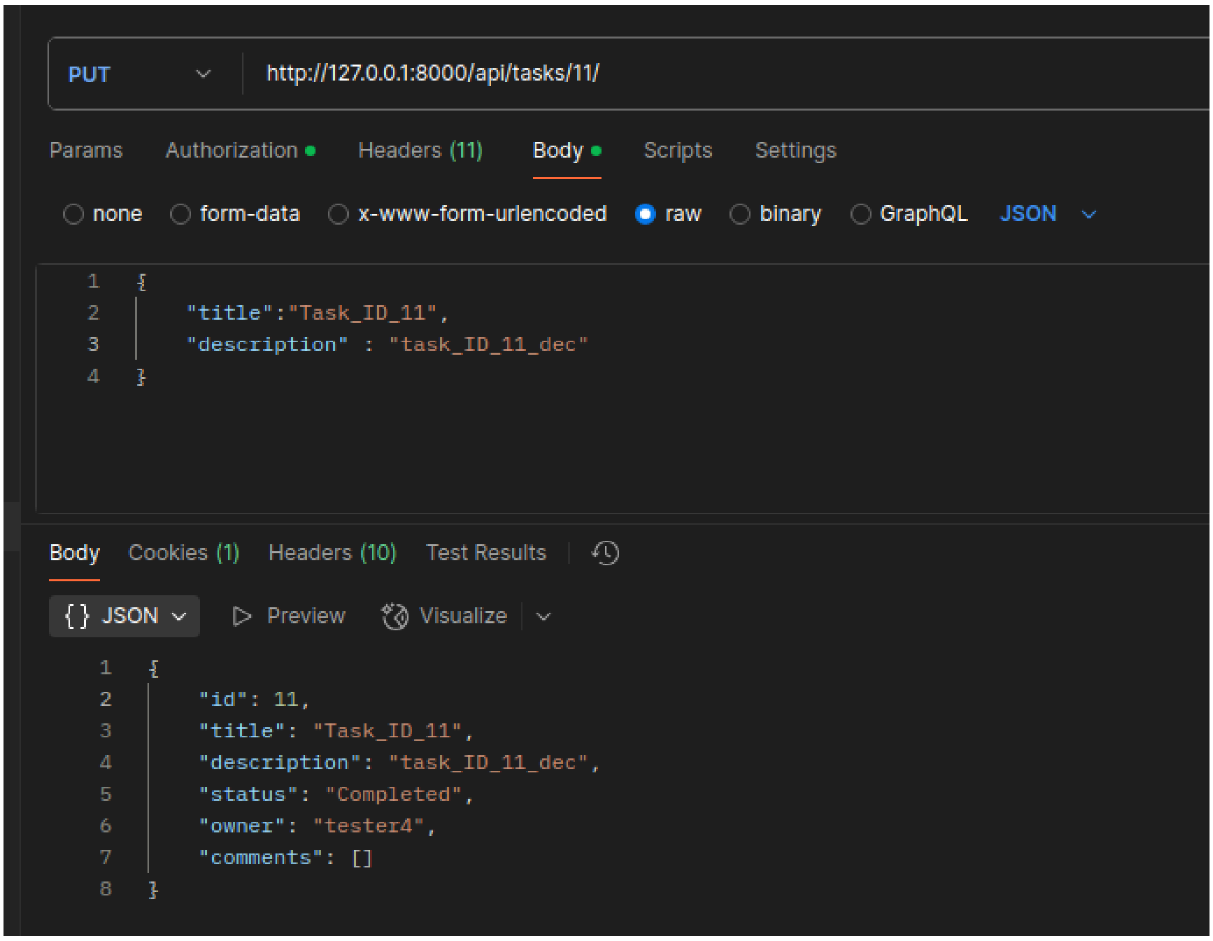
get all tasks



```
GET http://127.0.0.1:8000/api/tasks/12/
```

```
{
    "id": 12,
    "title": "task_10",
    "description": "new_task_10_description",
    "status": "Pending",
    "owner": "tester5",
    "comments": [
        {
            "id": 28,
            "text": "this is a comment for task 12",
            "parent": null,
            "author": "tester5",
            "created_date": "2025-03-13T06:52:31.770690Z",
            "replies": [
                {
                    "id": 29,
                    "text": "this is a reply for task parent 28",
                    "parent": 28,
                    "author": "tester5",
                    "created_date": "2025-03-13T06:53:23.837784Z",
                    "replies": []
                }
            ]
        },
        {
            "id": 29,
            "text": "this is a reply for task parent 28",
            "parent": 28,
            "author": "tester5",
            "created_date": "2025-03-13T06:53:23.837784Z",
            "replies": []
        }
    ]
}
```

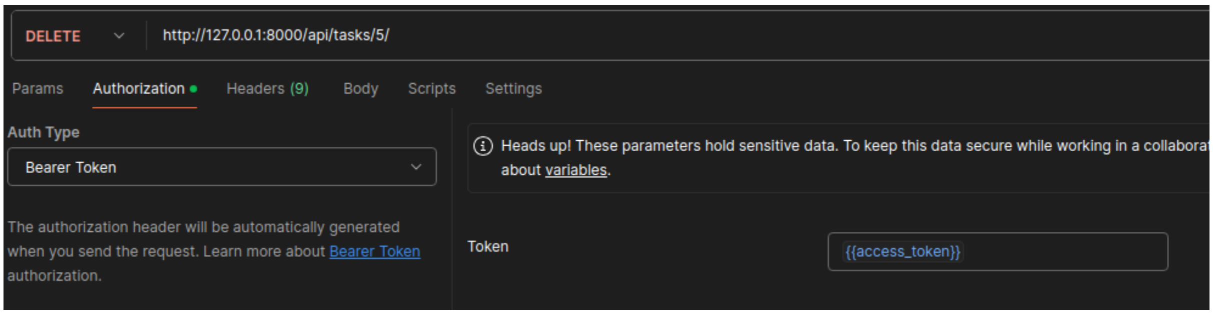
Update a task


PUT http://127.0.0.1:8000/api/tasks/11/

```
{
    "title":"Task_ID_11",
    "description" : "task_ID_11_dec"
}
```

```
{
    "id": 11,
    "title": "Task_ID_11",
    "description": "task_ID_11_dec",
    "status": "Completed",
    "owner": "tester4",
    "comments": []
}
```

delete a task


DELETE http://127.0.0.1:8000/api/tasks/5/

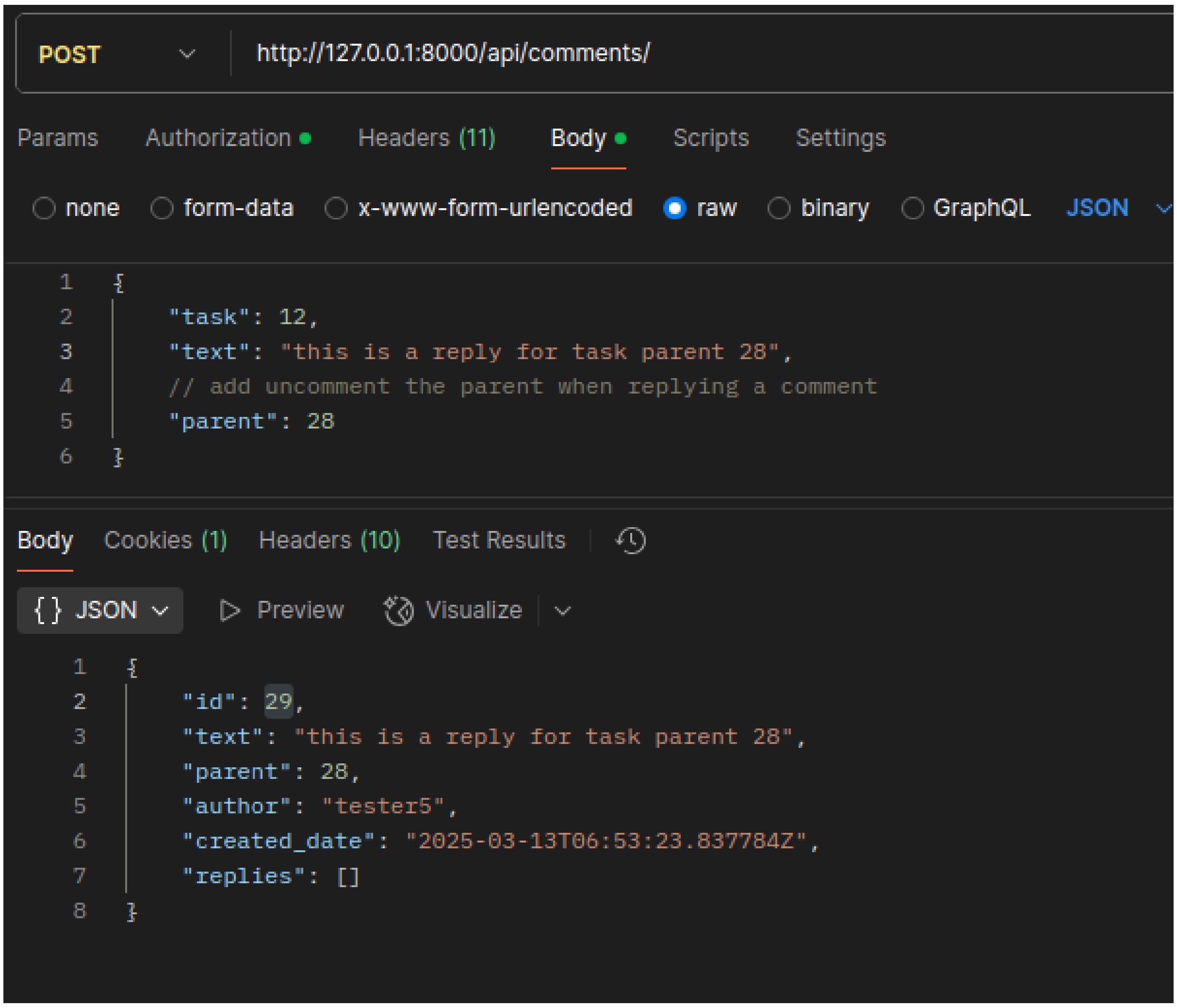
Auth Type
Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collabora
about variables.

Token {{access_token}}

add a comment

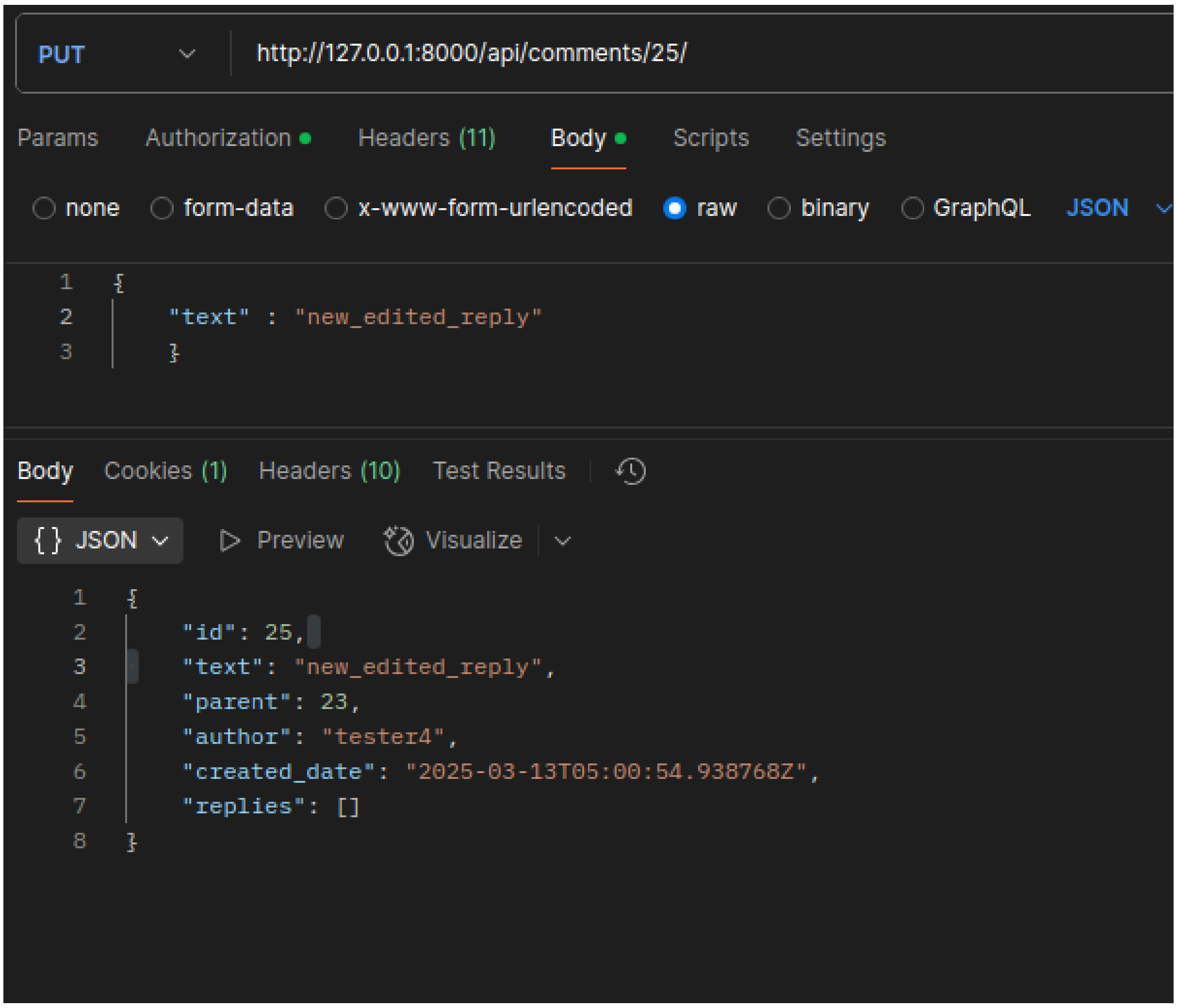


```
http://127.0.0.1:8000/api/comments/
```

```
{
    "task": 12,
    "text": "this is a reply for task parent 28",
    // add uncomment the parent when replying a comment
    "parent": 28
}
```

```
{
    "id": 29,
    "text": "this is a reply for task parent 28",
    "parent": 28,
    "author": "tester5",
    "created_date": "2025-03-13T06:53:23.837784Z",
    "replies": []
}
```

edit a comment



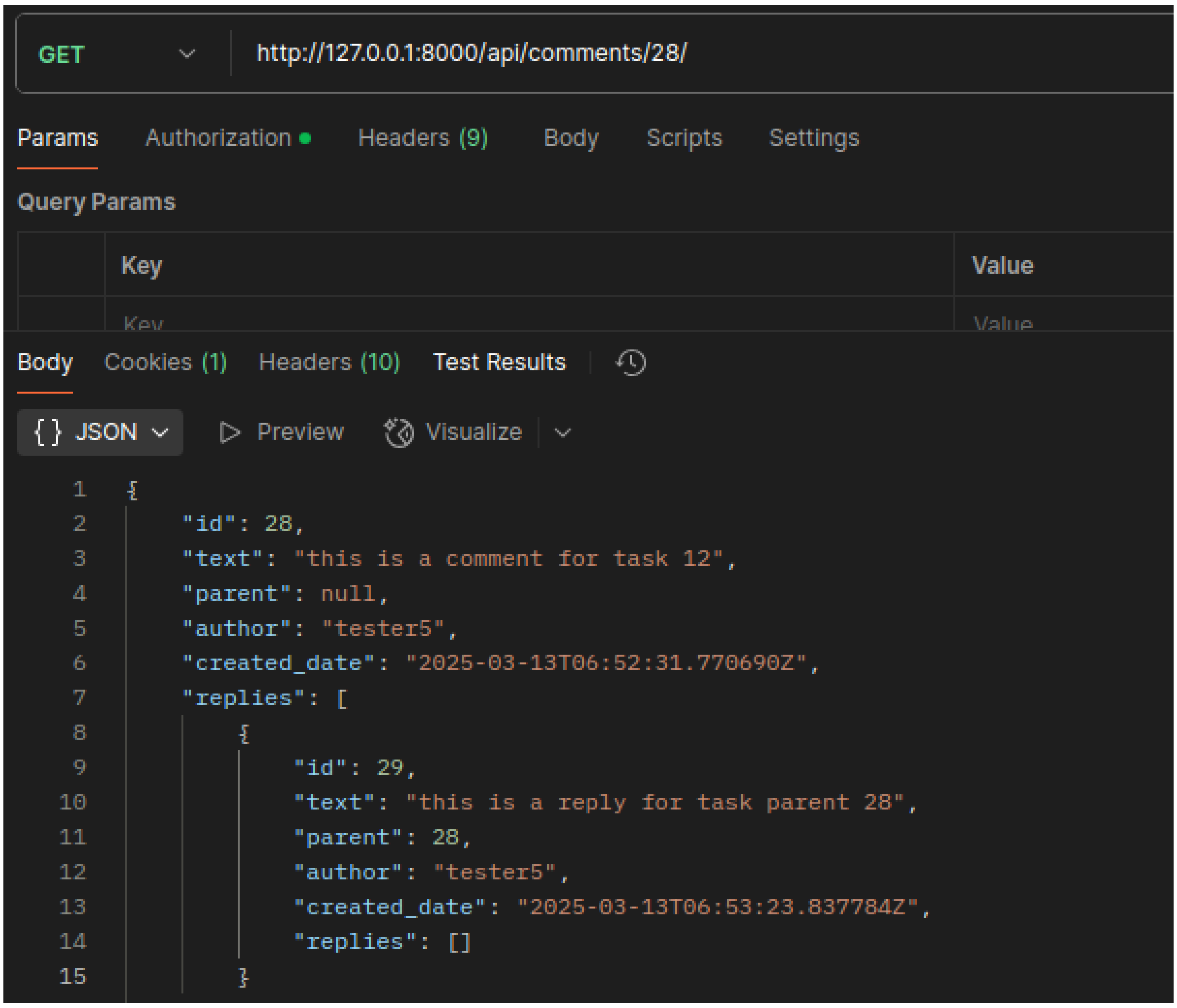
PUT http://127.0.0.1:8000/api/comments/25/

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
    "text" : "new_edited_reply"
    }
```

```
{
    "id": 25,
    "text": "new_edited_reply",
    "parent": 23,
    "author": "tester4",
    "created_date": "2025-03-13T05:00:54.938768Z",
    "replies": []
}
```

get specific comment



http://127.0.0.1:8000/api/comments/28/

```
{
    "id": 28,
    "text": "this is a comment for task 12",
    "parent": null,
    "author": "tester5",
    "created_date": "2025-03-13T06:52:31.770690Z",
    "replies": [
        {
            "id": 29,
            "text": "this is a reply for task parent 28",
            "parent": 28,
            "author": "tester5",
            "created_date": "2025-03-13T06:53:23.837784Z",
            "replies": []
        }
```