1. Python căn bản

2. Tạo virtual environment

python -m venv venv

.\venv\Scripts\activate

3. Cài đặt django

pip install django

4. Cấu trúc project Django

```
myproject/
│
├── manage.py  # Tệp chạy lệnh của Django
├── myproject/  # Thư mục chứa cài đặt dự án
│   ├── __init__.py
│   ├── settings.py  # Cài đặt toàn cục của dự án
│   ├── urls.py      # Định nghĩa URL của dự án
│   ├── asgi.py      # Cài đặt ASGI cho WebSocket (nếu có)
│   └── wsgi.py      # Cài đặt WSGI cho server production
│
├── apps/            # Thư mục chứa các ứng dụng Django
│   ├── __init__.py
│   ├── app1/        # Một ứng dụng trong dự án
│   │   ├── __init__.py
│   │   ├── models.py  # Các mô hình (models) của ứng dụng
│   │   ├── views.py   # Các views của ứng dụng
│   │   ├── urls.py    # URL của ứng dụng
│   │   ├── forms.py   # Các form của ứng dụng
│   │   └── tests.py   # Các test case của ứng dụng
│   │
│   └── app2/        # Ứng dụng khác (tương tự app1)
│
├── static/           # Tệp tĩnh (CSS, JavaScript, images)
│   └── ...
│
└── media/            # Tệp phương tiện do người dùng tải lên (images, videos)
    └── ...
```

5. Xây dựng ứng dụng Django cơ bản (cấu hình url, xây dựng view, tạo template html,...)

6. Tìm hiểu về ORM (ORM là gì, các kiểu dữ liệu ORM, Chạy makemigrations và migrate, các câu lệnh thao tác với ORM...)

- ORM (Object-Relational Mapping): kỹ thuật cho phép làm việc với cơ sở dữ liệu quan hệ mà không cần viết SQL thủ công
- Các kiểu dữ liệu ORM:

  Model: Đại diện cho bảng trong cơ sở dữ liệu

  Field: Các trường dữ liệu trong model, tương ứng với các cột trong bảng

- Các câu lệnh ORM cơ bản:

  python manage.py makemigrations - Tạo các file migration dựa trên sự thay đổi của models

  python manage.py migrate - Áp dụng các file migration vào cơ sở dữ liệu

- Các câu lệnh thao tác với Queryset:

  Tạo mới: Model.objects.create()

  Lấy dữ liệu: Model.objects.all() hoặc Model.objects.filter()

  Cập nhật: Model.objects.filter(id=1).update(field='value')

  Xóa: Model.objects.filter(id=1).delete()

7. Tìm hiểu về Class-based View và function base view

Function-based Views (FBV): Hàm nhận các đối tượng yêu cầu HTTP và trả về respond. Có cấu trúc rõ ràng và đơn giản, tuy nhiên không thể tái sử dụng mã khi có các logic giống nhau trong các views

Class-based Views (CBV): lớp đại diện cho các views, cho phép tái sử dụng mã và tách biệt các logic khác nhau theo phương thức HTTP (GET, POST, PUT, DELETE). Ưu điểm tái sử dụng code bằng cách kế thừa. Một số CBV có sẵn:

- TemplateView: Hiển thị một template với các context.
- ListView: Hiển thị danh sách các đối tượng từ cơ sở dữ liệu.
- DetailView: Hiển thị chi tiết một đối tượng từ cơ sở dữ liệu (1 bài viết, 1 sản phẩm)
- CreateView, UpdateView, DeleteView: Tạo, cập nhật, và xóa đối tượng
- RedirectView: chuyển hướng tới URL khác
- FormView: Hiển thị và xử lý form

8. Authentication và Authorization

Authentication (Xác thực): xác thực người dùng thông qua việc đăng nhập

- Sử dụng module *django.contrib.auth*
- Django có sẵn *LoginView LogoutView* và phương thức *authenticate()* để xử lý việc đăng nhập.

Authorization (Ủy quyền): quá trình kiểm tra xem người dùng đã xác thực có quyền truy cập vào tài nguyên, hành động cụ thể nào đó hay không

- Django cung cấp hệ thống phân quyền thông qua các permissions, groups, và decorators để xác định quyền truy cập của người dùng vào các view
- **Các cách quản lý quyền trong Django:**
    - **Permissions**: Mỗi mô hình trong Django có các quyền mặc định như add, change, delete, view
    - **Groups**: Các nhóm người dùng có thể có quyền giống nhau, quản lí quyền dễ dàng hơn
    - **Decorators**: Django cung cấp các decorator như @login_required, @permission_required để bảo vệ các view, chỉ cho phép người dùng có quyền truy cập nhất định.

9 Cài đặt Django rest framework

pip install djangorestframework

Django REST Framework (DRF): triển khai các web service dựa trên RESTful, giúp người dùng giao tiếp với ứng dụng Django qua các yêu cầu HTTP và nhận phản hồi dữ ở dạng JSON

Các tính năng chính của Django REST Framework:

1. **Serializer**: Chuyển đổi dữ liệu giữa các mô hình Django và các định dạng JSON, XML. Serializer giúp việc làm việc với dữ liệu trở nên dễ dàng hơn, ví dụ như chuyển đổi dữ liệu từ mô hình thành JSON để gửi về client.
2. **ViewSets và Routers**: Giúp quản lý các endpoint API một cách linh hoạt và dễ dàng. ViewSets tự động xử lý các phương thức HTTP. Routers giúp tạo các URL cho các viewsets mà không cần phải tự cấu hình từng URL
3. **Authentication và Authorization**: DRF cung cấp các cơ chế xác thực và phân quyền gồm session authentication, token authentication
4. **Thư viện cho Pagination, Filtering, và Sorting**: DRF hỗ trợ phân trang (pagination), lọc (filtering), và sắp xếp (sorting) dữ liệu cho API dễ dàng, giúp người dùng quản lý dữ liệu khi có số lượng lớn
5. **Browsable API**: cung cấp một giao diện web để tương tác với các API, giúp dễ dàng kiểm tra và thao tác với các endpoint API ngay trong trình duyệt

6. **Support for ViewSets and Generic Views**: DRF cung cấp nhiều lớp view cơ bản như ModelViewSet và GenericAPIView, giúp tạo các API CRUD nhanh chóng mà không cần phải viết nhiều mã lệnh.

10. Serializers, API view, ViewSet, Router

Serializers: hoạt động giống như các "bộ chuyển đổi" dữ liệu, giúp chuyển đổi dữ liệu từ mô hình Django thành định dạng dễ dàng xử lý (JSON) hoặc ngược lại. Xác thực dữ liệu đầu vào (như kiểm tra kiểu dữ liệu, yêu cầu trường bắt buộc)

API Views:

- Là cách tạo ra các view trong DRF để xử lý các yêu cầu HTTP và trả về phản hồi dạng JSON.
- Ghi đè get() post() để xử lí yêu cầu
- Có sẵn các công cụ của DRF như xác thực, phân quyền, và xử lý JSON.

ViewSets là cách viết ngắn gọn hơn so với APIView, giúp tự động hóa việc xử lý các phương thức CRUD. Thay vì phải viết riêng từng phương thức cho GET, POST, PUT, DELETE có thể sử dụng ViewSet và DRF sẽ tự động cung cấp các hành động này

- **ModelViewSet**: Là một trong các subclass của ViewSet, hỗ trợ đầy đủ các phương thức CRUD.
- **ReadOnlyModelViewSet**: Chỉ hỗ trợ các hành động GET (chỉ đọc).

Router: trong DRF giúp tự động tạo ra các URL cho các ViewSets mà không cần phải định nghĩa từng URL thủ công. DRF cung cấp các loại router như **DefaultRouter** và **SimpleRouter**. **DefaultRouter** tự động thêm một endpoint để xem danh sách API. Chỉ cần đăng ký *viewset* với router và DRF sẽ tạo URL cho các hành động CRUD.

11. Tìm hiểu về Authentication (Token, Session, JWT)

Token Authentication là phương thức xác thực đơn giản nhất trong các API RESTful. Người dùng sẽ nhận được một **token** sau khi đăng nhập, và token này sẽ được gửi kèm trong các yêu cầu tiếp theo để xác minh người dùng. Token là một chuỗi ký tự duy nhất, có thể được sử dụng để xác thực mà không cần gửi lại mật khẩu.

Cách thức hoạt động:

- Người dùng gửi tên người dùng và mật khẩu qua API (sử dụng phương thức **POST**).
- Hệ thống kiểm tra tính hợp lệ của thông tin và nếu hợp lệ, hệ thống tạo một **token** duy nhất cho người dùng.
- Token này sẽ được trả về và người dùng sẽ sử dụng token này trong các yêu cầu tiếp theo, gửi kèm trong **Authorization Header**

Session Authentication là phương thức xác thực mặc định của Django. Thay vì gửi token, người dùng sẽ sử dụng **cookie session** để duy trì trạng thái đăng nhập. Khi người dùng đăng nhập thành công, Django tạo ra một session ID và lưu trữ nó trong cookie của trình duyệt.

Cách thức hoạt động:

- Người dùng gửi tên người dùng và mật khẩu qua **POST**.
- Nếu thông tin hợp lệ, Django tạo một session cho người dùng và lưu session ID vào trong **cookies** của trình duyệt.
- Mỗi lần người dùng gửi yêu cầu tới server, session ID sẽ tự động được gửi kèm trong cookie và Django sẽ kiểm tra session để xác minh người dùng.

JWT Authentication: phương thức xác thực mạnh mẽ và phổ. JWT là một chuỗi mã hóa chứa thông tin về người dùng, thời gian hết hạn, và các quyền truy cập. JWT thường được sử dụng trong các API phân tán, nơi không muốn sử dụng session-based xác thực.

Cách thức hoạt động:

- Người dùng gửi tên người dùng và mật khẩu để đăng nhập.
- Server xác thực thông tin và tạo một **JWT** cho người dùng. JWT này có thể chứa thông tin như user_id, exp,…
- Sau khi người dùng nhận được JWT, họ gửi token này trong phần **Authorization Header** của các request
- Server sẽ giải mã token và xác nhận tính hợp lệ của nó trước khi xử lý yêu cầu.

12. Permissions & Throttling

Permissions trong DRF sử dụng để xác định liệu người dùng có quyền truy cập vào một endpoint API cụ thể hay không. Đây là cơ chế để bảo vệ các API, chỉ cho phép những người dùng có quyền hợp lệ truy cập vào tài nguyên của ứng dụng.

Có thể cấu hình ở mức global trong settings.py

Các lớp Permission trong DRF

- **IsAuthenticated**: Yêu cầu người dùng phải được xác thực.
- **IsAdminUser**: Chỉ cho phép người dùng là admin.
- **IsAuthenticatedOrReadOnly**: Cho phép đọc (GET, HEAD, OPTIONS) cho mọi người, nhưng yêu cầu xác thực cho các hành động khác (POST, PUT, DELETE).
- **AllowAny**: Cho phép tất cả người dùng (được sử dụng khi không có hạn chế nào về quyền truy cập).
- **IsOwner**: Tùy chỉnh để chỉ cho phép người dùng sở hữu đối tượng có quyền truy cập.

Throttling là cơ chế kiểm soát tần suất yêu cầu từ người dùng đến API, nhằm ngăn ngừa việctấn công Ddos. Trong DRF, throttling giúp giới hạn số lượng yêu cầu mà một người dùng có thể gửi trong một khoảng thời gian nhất định.

Có thể cấu hình ở mức global trong settings.py

DRF cung cấp một số loại throttling mặc định, bao gồm:

- **UserRateThrottle**: Giới hạn số lượng yêu cầu mà mỗi người dùng có thể thực hiện trong một khoảng thời gian.
- **AnonRateThrottle**: Giới hạn số lượng yêu cầu mà người dùng chưa đăng nhập (anonymous users) có thể thực hiện trong một khoảng thời gian.
- **ScopedRateThrottle**: Giới hạn yêu cầu dựa trên các phạm vi khác nhau (scope), cho phép bạn kiểm soát yêu cầu cho từng loại API cụ thể.

13. Quản lý Static và Media files

Static files bao gồm các tệp như CSS, js, hình ảnh và các tệp tĩnh khác mà không thay đổi trong suốt thời gian chạy của ứng dụng. Cần cấu hình trong settings.py và sử dụng templatetag {% static %}

Khi triển khai ứng dụng lên môi trường production, cần thu thập tất cả các tệp static từ các thư mục khác nhau vào một thư mục duy nhất: python manage.py collectstatic

Media files là các tệp do người dùng tải lên, như hình ảnh, video, tài liệu,... Những tệp này thường thay đổi và có thể được lưu trữ trong cơ sở dữ liệu hoặc dưới dạng các tệp thực trên hệ thống file.

Trong môi trường production, Django không phục vụ trực tiếp các tệp static và media. Thay vào đó cần một web server (Nginx, Apache) để phục vụ các tệp này

14. Unittest

Các test case trong Django được đặt trong tệp tests.py của mỗi app trong dự án.

sử dụng các lớp kế thừa từ django.test.TestCase để viết các unit test, cung cấp các phương thức như setUp(), tearDown() và các phương thức kiểm tra (assert)

Các phương thức kiểm thử cơ bản

assertEqual(a, b): Kiểm tra xem a và b có bằng nhau không.

assertNotEqual(a, b): Kiểm tra xem a và b có khác nhau không.

assertTrue(x): Kiểm tra xem x có phải là True không.

assertFalse(x): Kiểm tra xem x có phải là False không.

assertIsNone(x): Kiểm tra xem x có phải là None không.

assertIsNotNone(x): Kiểm tra xem x có phải là khác None không.

assertIn(a, b): Kiểm tra xem a có nằm trong b không.

assertNotIn(a, b): Kiểm tra xem a có không nằm trong b không.

- Chạy các test trong Django: *python manage.py test*
- **TestView và testForm**

15. Triển khai ứng dụng trên các môi trường (nginx, gunicorn,...)

- **Gunicorn** (Green Unicorn) là một WSGI server được sử dụng để chạy ứng dụng Django trong môi trường production.

pip install gunicorn

gunicorn --workers 3 myproject.wsgi:application

- **Nginx** là một web server và reverse proxy rất mạnh mẽ, thường được sử dụng kết hợp với Gunicorn để phục vụ ứng dụng Django trong môi trường production.

sudo apt update

sudo apt install nginx

- cần tạo một file cấu hình mới cho Nginx để reverse proxy các yêu cầu đến Gunicorn

sudo nano /etc/nginx/sites-available/myproject

- Cấu hình Gunicorn với Systemd để chạy Gunicorn như một dịch vụ hệ thống.

Cấu hình Static và Media Files

16. Tạo socket server với Python socketIO

Socket.IO trong Python giúp tạo các ứng dụng thời gian thực, với việc sử dụng thư viện python-socketio để tạo server và giao tiếp giữa client và server qua WebSocket hoặc các giao thức khác

Django Channels là một công cụ để làm việc với WebSocket trong Django, và có thể tích hợp với **python-socketio**

django-socketio là một giải pháp cũ để tích hợp Socket.IO vào Django

- cài đặt:

pip install python-socketio

pip install gevent eventlet

pip install channels

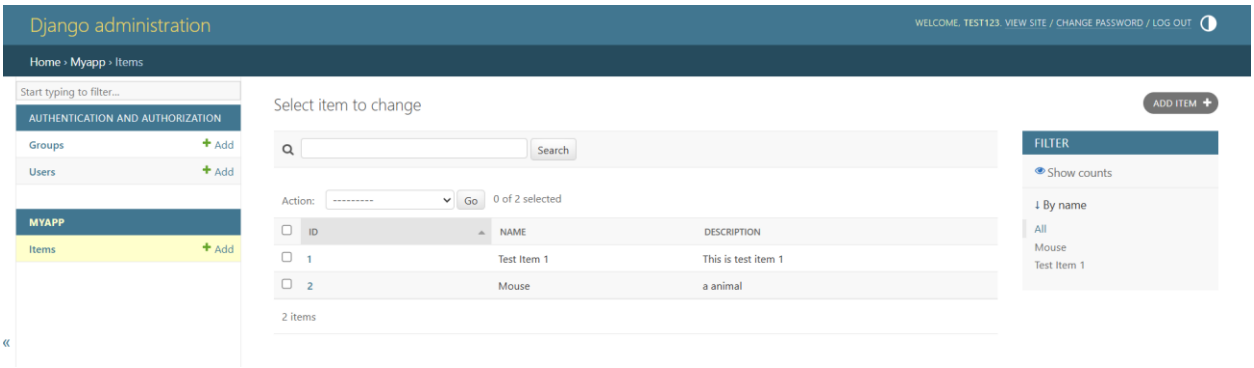
Socket server được khởi tạo bằng socketio.Server().

connect(), message(), và disconnect() là các hàm lắng nghe các sự kiện như kết nối, nhận tin nhắn, và ngắt kết nối.

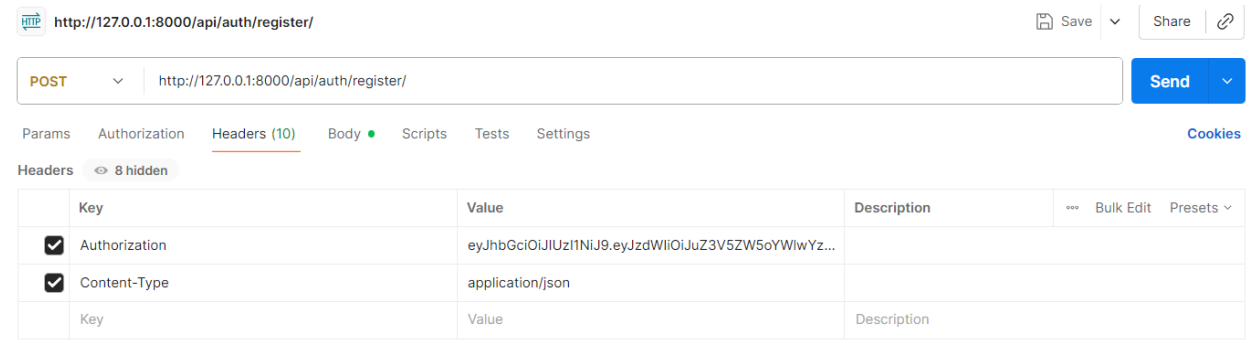
gevent và **eventlet** giúp server có thể xử lý các kết nối đồng thời.

Test DEMO với Postman

Django Admin

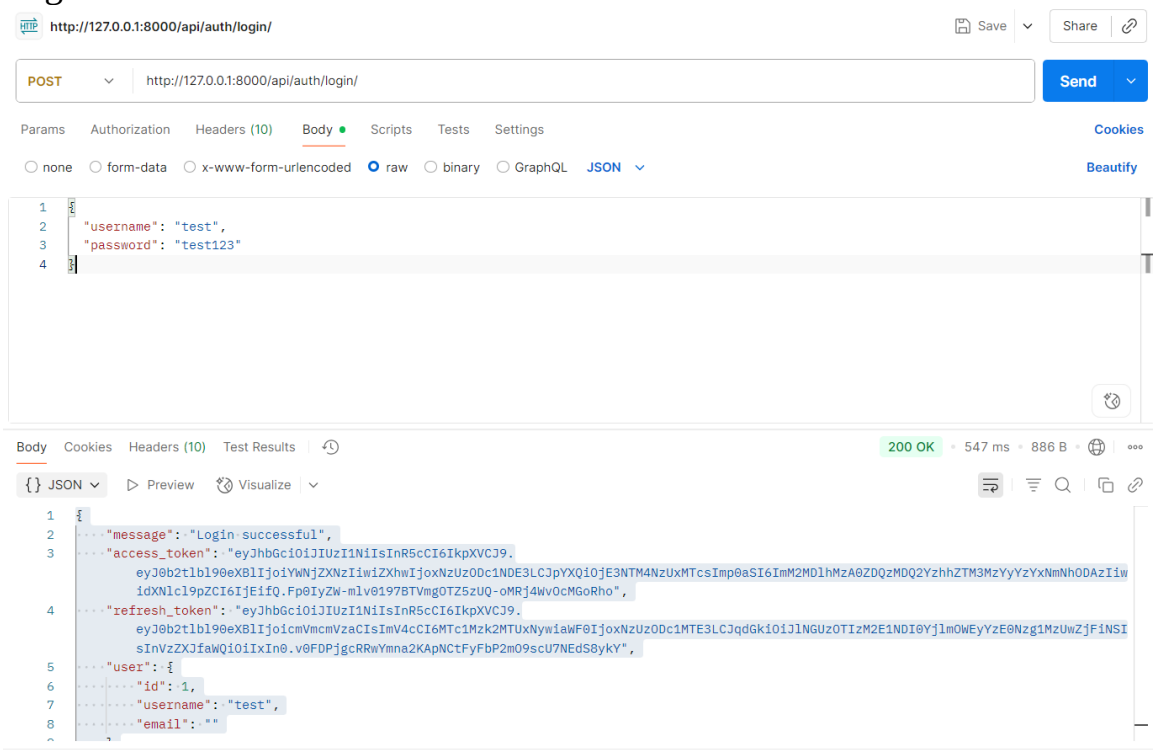


Register:

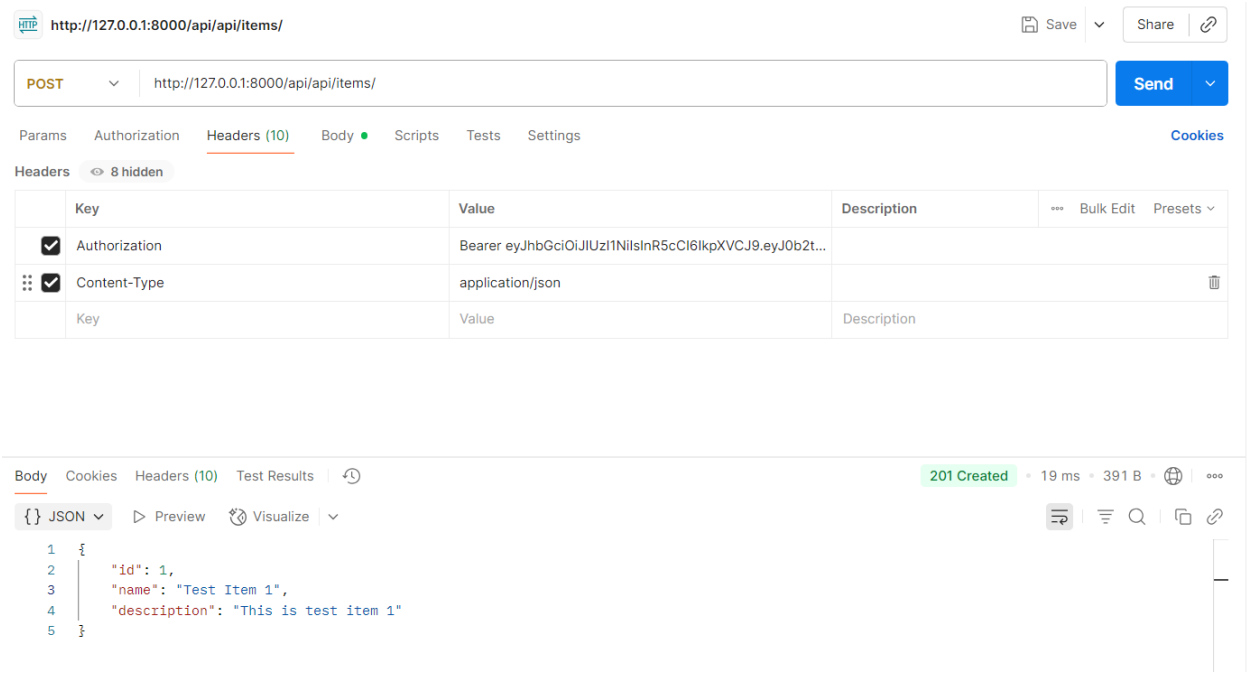

http://127.0.0.1:8000/api/auth/register/
Save
Share
POST
http://127.0.0.1:8000/api/auth/register/
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
Headers
8 hidden
Key
Value
Description
Bulk Edit
Presets
Authorization
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJuZ3V5ZW5oYWlwYz...
Content-Type
application/json
Key
Value
Description


Body
Cookies
Headers (10)
Test Results
201 Created
648 ms
903 B
JSON
Preview
Visualize
{
"message": "User registered successfully",
"access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzUzODc1MzA3LCJpYXQiOjE3NTM4NzUwMDcsImp0aSI6ImY1NmQzMDRlNmIzNjRjMjM4NjIzZWY5YjRlOGFjZjg1IiwidXNlcl9pZCI6IjEifQ.euxjbORnQ_dOMx_3-U1TZY46sxj9m_wByQHHr5CSjA8",
"refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTc1Mzk2MTQwNywiaWF0IjoxNzUzODc1MDA3LCJqdGkiOiJhZDVjMjQ4ZDc4ZmY0YzEyODMwMzJhYTBkNzFlNjY3NyIsInVzZXJfaWQiOiIxIn0.nH3j-xwt1Z6MEm-Zv7fmwnKVXWcHBRNU8DvvgQq4n80",
"user": {
"id": 1,
"username": "test",
"email": ""

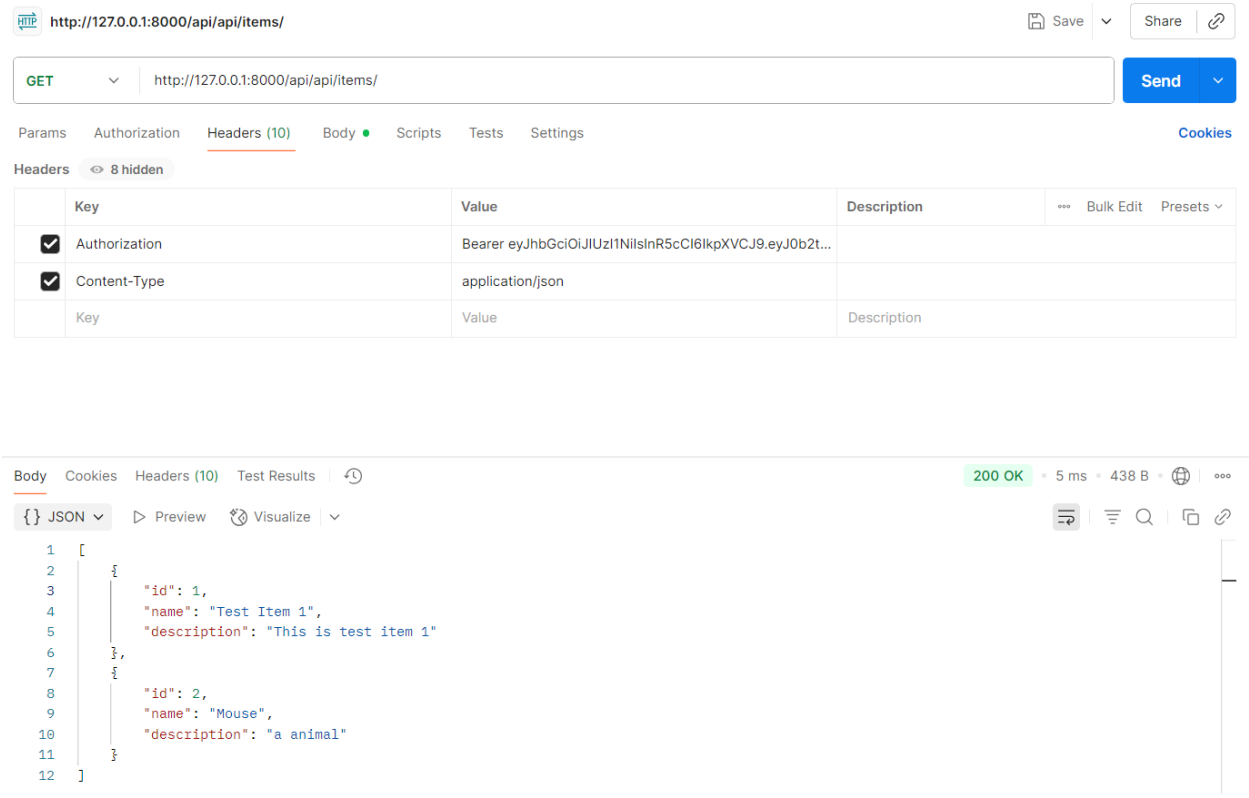
Login:


http://127.0.0.1:8000/api/auth/login/
Save
Share
POST
http://127.0.0.1:8000/api/auth/login/
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"username": "test",
"password": "test123"
}
Body
Cookies
Headers (10)
Test Results
200 OK
547 ms
886 B
JSON
Preview
Visualize
{
"message": "Login successful",
"access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzUzODc1NDE3LCJpYXQiOjE3NTM4NzUxMTcsImp0aSI6ImM2MDlhMzA0ZDQzMDQ2YzhhZTM3MzYyYzYxNmNhODAzIiwidXNlcl9pZCI6IjEifQ.Fp0IyZW-mlv0197BTVmgOTZ5zUQ-oMRj4WvOcMGoRho",
"refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTc1Mzk2MTUxNywiaWF0IjoxNzUzODc1MTE3LCJqdGkiOiJlNGUzOTIzM2E1NDI0YjlmOWEyYzE0Nzg1MzUwZjFiNSIsInVzZXJfaWQiOiIxIn0.v0FDPjgcRRwYmna2KApNCtFyFbP2mO9scU7NEdS8ykY",
"user": {
"id": 1,
"username": "test",
"email": ""

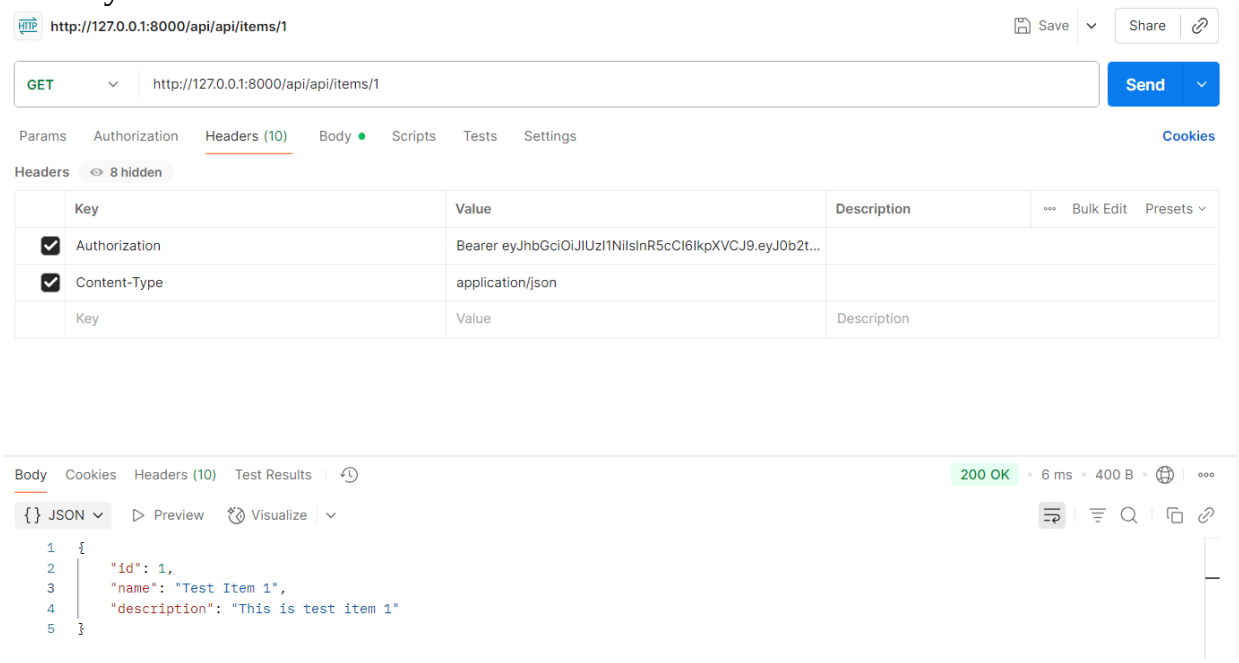
POST item:



GET all items:



GET by id



NỘI DUNG BỔ SUNG KHÁC

Một số mã trạng thái HTTP phổ biến:

- **200 OK**: Yêu cầu thành công, và dữ liệu (nếu có) sẽ được trả về. Đây là mã trạng thái mặc định khi không chỉ định mã trạng thái.
- **201 Created**: Yêu cầu đã được thực thi và tài nguyên mới đã được tạo.
- **400 Bad Request**: Yêu cầu không hợp lệ, có thể do thiếu dữ liệu hoặc dữ liệu sai định dạng.
- **403 Forbidden**: Người dùng không có quyền truy cập tài nguyên, dù tài nguyên đó có tồn tại hay không.
- **404 Not Found**: không tìm thấy tài nguyên
- **500 Internal Server Error**: Lỗi phía server, thường là do server gặp sự cố hoặc lỗi trong mã nguồn.

Cách dùng JsonResponse:

```
from django.http import JsonResponse

return JsonResponse(data, safe=True, json_dumps_params=None,
content_type='application/json')
```

Cách dùng path() trong url.py: khai báo tên "name" để sử dụng trong các thẻ tag {% url 'name' %} nhằm mục đích dễ dàng chỉnh sửa đường dẫn ở nhiều nơi khác nhau trong mã.

path(route, view, kwargs=None, name=None)

Các methods của HTTP:

- **GET**: Lấy dữ liệu (không thay đổi).
- **POST (**Không idempotent**)**: Gửi dữ liệu để tạo mới tài nguyên, tạo nhiều bản ghi giống nhau nếu không có cơ chế kiểm tra
- **PUT**: Cập nhật tài nguyên (toàn bộ).
- **PATCH**: Cập nhật tài nguyên (một phần) thay vì toàn bộ như PUT
- **DELETE**: Xóa tài nguyên.
- **HEAD**: Lấy header của tài nguyên (giống GET nhưng không có body).
- **OPTIONS**: Kiểm tra các phương thức hỗ trợ cho tài nguyên.
- **TRACE**: Kiểm tra đường đi của yêu cầu (dùng chủ yếu cho debugging).

Cấu trúc của hàm `authenticate()`:

user = authenticate(request, username='username', password='password')

login(request, user)

- chỉ trả về người dùng nếu tên người dùng và mật khẩu hợp lệ. Nếu sai, nó sẽ trả về `None`
- Hàm **`login()`** sẽ thiết lập session cho người dùng, giúp họ duy trì trạng thái đăng nhập trong suốt phiên làm việc.