**BÁO CÁO XÂY DỰNG WEB APP TO-DO LIST**

**Mục lục**

[1. Lựa chọn công nghệ 3](#_Toc184413376)

[1.1. Chọn ngôn ngữ lập trình 3](#_Toc184413377)

[1.2. Chọn framework cho frontend 3](#_Toc184413378)

[1.3. Chọn cơ sở dữ liệu 3](#_Toc184413379)

[1.4. Chọn môi trường chạy server 3](#_Toc184413380)

[2. Xử lý backend 4](#_Toc184413381)

[2.1. Thiết lập môi trường 4](#_Toc184413382)

[2.1.1. Tạo môi trường ảo 4](#_Toc184413383)

[2.1.2. Cài đặt Django và Django REST Framework và cơ sở dữ liệu PostgreSQL: 4](#_Toc184413384)

[2.1.3. Tạo dự án Django 4](#_Toc184413385)

[2.2. Thiết lập settings.py 5](#_Toc184413386)

[2.2.1. Đăng ký ứng dụng trong danh sách INSTALLED\_APPS 5](#_Toc184413387)

[2.2.2. Cấu hình CORS cho phép frontend React giao tiếp với backend Django 5](#_Toc184413388)

[2.2.3. Cấu hình DATABASE trong setting.py 6](#_Toc184413389)

[2.2.4. Chạy lệnh thiết lập cơ sở dữ liệu 9](#_Toc184413390)

[2.3. Tạo tài khoản quản trị viên 9](#_Toc184413391)

[2.4. Tạo model Task trong tasks/models.py 9](#_Toc184413392)

[2.5. Tạo Serializers (nếu dùng Django REST Framework) 10](#_Toc184413393)

[2.6. Tạo Views (tasks/views.py) 10](#_Toc184413394)

[2.7. Cấu hình URL để truy cập API: 11](#_Toc184413395)

[2.8. Chạy thử backend 12](#_Toc184413396)

# Lựa chọn công nghệ

Trong quá trình xây dựng phần mềm quản lý To-Do List, việc lựa chọn công nghệ phù hợp là một bước quan trọng, đảm bảo dự án được phát triển một cách hiệu quả, dễ bảo trì và có khả năng mở rộng trong tương lai.

## Chọn ngôn ngữ lập trình

Lựa chọn ngôn ngữ Python cho backend. Vì Python là ngôn ngữ lập trình phổ biến, dễ học, phù hợp với người mới bắt đầu. Python có cộng đồng lớn, tài liệu phong phú và nhiều thư viện hỗ trợ.

Framework sử dụng Django, một framework mạnh mẽ, có nhiều tính năng tích hợp sẵn (ORM, bảo mật, quản trị).

## Chọn framework cho frontend

Framework lựa chọn cho frontend là React. React là thư viện frontend mạnh mẽ, phổ biến và linh hoạt, được sử dụng để xây dựng giao diện người dùng hiện đại. Cung cấp khả năng xây dựng giao diện dạng component, giúp dễ bảo trì và tái sử dụng mã nguồn. Có một hệ sinh thái lớn, dễ dàng tích hợp với các thư viện và công cụ khác.

Để sử dụng được React cần cài đặt Nodejs. Nodejs là môi trường chạy JavaScript bên ngoài trình duyệt. React sử dụng Node.js để quản lý gói (packages) và chạy các công cụ phát triển.

## Chọn cơ sở dữ liệu

Lựa chọn PostgreSQL. PostgreSQL là cơ sở dữ liệu mã nguồn mở mạnh mẽ, hỗ trợ tốt cho các ứng dụng cần xử lý dữ liệu phức tạp. PostgreSQL tích hợp tốt với Django thông qua công cụ ORM (Object-Relational Mapping). Và hỗ trợ các tính năng như transaction, indexing, và JSON data, giúp dễ dàng mở rộng khi cần.

## Chọn môi trường chạy server

Dự án lựa chọn sử dụng **Server cục bộ (Local Server)** trong giai đoạn phát triển vì:

* Không tốn chi phí thuê server đám mây hoặc hosting, phù hợp với nhóm không có ngân sách.
* Đáp ứng đủ nhu cầu kiểm tra và phát triển ứng dụng trong môi trường nội bộ.

# Xử lý backend

Phần backend của dự án được xây dựng bằng **Django**, một framework mạnh mẽ và dễ sử dụng trong việc phát triển các ứng dụng web. Nhiệm vụ chính của backend là cung cấp API RESTful để xử lý các thao tác quản lý tasks như tạo, xem, chỉnh sửa và xóa.

## Tạo thư mục dự án

### Tạo môi trường ảo

Để đảm bảo dự án sử dụng đúng phiên bản thư viện và tránh xung đột với các dự án khác, ta nên tạo một môi trường riêng dành cho dự án. Một môi trường ảo Python đã được tạo bằng lệnh:

python -m venv tên\_môi\_trường\_ảo

Sau đó, cần phải kích hoạt môi trường ảo bằng lệnh:

.\tên\_môi\_trường\_ảo\Scripts\Activate.ps1

### Cài đặt các thư viện cần thiết cho môi trường ảo

Sau khi kích hoạt môi trường ảo, các thư viện cần thiết được cài đặt thông qua pip:

pip install django djangorestframework psycopg2-binary

* Django: Framework chính để xây dựng backend.
* Django REST Framework: Công cụ hỗ trợ xây dựng API RESTful.
* Psycopg2-binary: Thư viện giúp kết nối Django với cơ sở dữ liệu PostgreSQL.

Sau khi cài đặt, các cấu hình cơ sở dữ liệu trong file settings.py được cập nhật để sử dụng PostgreSQL thay vì SQLite mặc định.

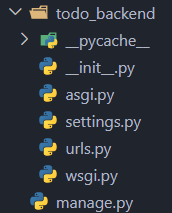
### Tạo dự án Django

Sau khi hoàn thành quá trình thiết lập môi trường và cài đặt các thư viện cần thiết, bước tiếp theo là khởi tạo dự án Django. Việc tạo một dự án Django mới sẽ cung cấp cho chúng ta cấu trúc thư mục cơ bản và các file cấu hình cần thiết để bắt đầu phát triển backend của ứng dụng.

Khởi tạo dự án bằng câu lệnh: django-admin startproject project\_name .

Câu lệnh trên sẽ tạo các file và thư mục dự án trực tiếp trong thư mục hiện tại

**Cấu trúc thư mục được tạo như sau:**

****

Ý nghĩa của từng file:

* manage.py: giúp ta tương tác Project qua các **command**(như là tạo tài khoản admin, tạo database, chạy server ảo, ...), vì vậy không nên chỉnh sửa ở đây.
* \_init\_.py:
* settting.py: đây là file cấu hình project.
* urls.py:
* asgi.py:
* wsgi.py:

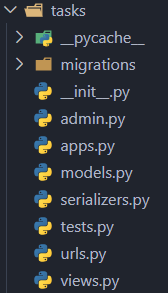
Sau khi đã khởi tạo dự án Django, bước tiếp theo là tạo ứng dụng (app) trong dự án. Để làm điều này, cần chuyển hướng đến thư mục chứa file manage.py của dự án. Ta có thể thực hiện điều này bằng cách sử dụng câu lệnh sau: cd current\_directory

Tiếp theo, để tạo ứng dụng muốn lập trình, sử dụng lệnh sau:

python manage.py startapp app\_name

Trong đó, app\_name là tên của ứng dụng muốn tạo. Đối với dự án này, chúng ta sẽ đặt tên ứng dụng là **‘tasks’**. Tên này được chọn vì nó phản ánh trực tiếp chức năng chính của ứng dụng, đó là quản lý các công việc (tasks). Tuy nhiên, tên của ứng dụng có thể được thay đổi tùy thuộc vào mục đích và chức năng của dự án đang phát triển.

Cấu trúc của thư mục được tạo ra như sau:



Ý nghĩa của từng file:

## Cấu hình file settings.py

### Đăng ký ứng dụng trong danh sách INSTALLED\_APPS

Để ứng dụng tasks hoạt động trong dự án Django, cần thêm tên ứng dụng vào danh sách INSTALLED\_APPS trong file settings.py. Đây là bước quan trọng để Django có thể nhận diện và quản lý ứng dụng.

* Mở file settings.py trong thư mục gốc của dự án (thư mục chứa manage.py).
* Tìm phần INSTALLED\_APPS và thêm tên ứng dụng tasks vào danh sách này. Cụ thể, cần thêm dòng sau vào danh sách:

# Ứng dụng To-Do List

'tasks', # Tên ứng dụng vừa đặt (ứng dụng chứa model To-Do List)

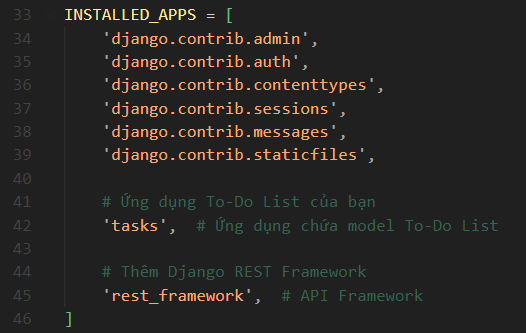
Tiếp theo, ta thêm **'rest\_framework',** đây là tên của ứng dụng Django REST Framework trong hệ thống Django. Để sử dụng DRF trong dự án, cần thêm nó vào danh sách INSTALLED\_APPS

# Thêm Django REST Framework

'rest\_framework', # API Framework

Khi ta thêm 'rest\_framework' vào INSTALLED\_APPS, Django sẽ biết rằng bạn muốn sử dụng các tính năng của Django REST Framework, như:

1. **Xây dựng các API endpoint:** Sử dụng DRF để dễ dàng tạo các API cho các thao tác CRUD (Create, Read, Update, Delete).
2. **Serialization:** DRF hỗ trợ việc chuyển đổi dữ liệu giữa các đối tượng Python và các định dạng dễ đọc như JSON.
3. **Authentication và Authorization:** DRF cung cấp các phương pháp bảo mật, bao gồm xác thực người dùng và phân quyền.



Sau khi thực hiện những lệnh trên, Django sẽ nhận diện và sử dụng ứng dụng tasks để quản lý các tính năng của To-Do List, cũng như tích hợp với Django REST Framework để phát triển API.

### Cấu hình CORS cho phép frontend React giao tiếp với backend Django

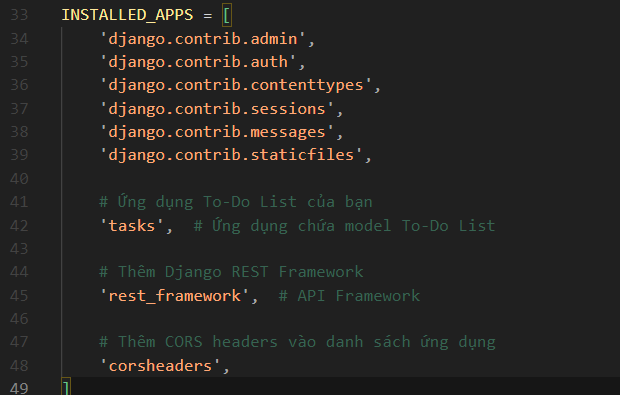
Trong quá trình phát triển ứng dụng web, frontend (React) và backend (Django) thường chạy trên các cổng (port) khác nhau. Ví dụ, React thường chạy trên port 3000, trong khi Django chạy trên port 8000. Điều này có thể gây ra lỗi liên quan đến **CORS (Cross-Origin Resource Sharing)** khi frontend cố gắng gửi các yêu cầu API đến backend. Để giải quyết vấn đề này, cần cấu hình CORS trong Django để cho phép giao tiếp giữa frontend và backend.

Để cài đặt CORS, cần cài đặt thư viện **django-cors-headers**. Chạy lệnh sau để cài đặt vào môi trường ảo

pip install django-cors-headers

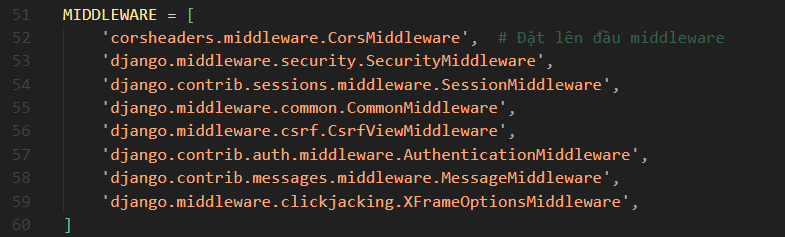
Sau khi cài đặt thành công, bạn cần đăng ký ứng dụng corsheaders trong file settings.py để Django có thể sử dụng. Nghĩa là, tthêm dòng lệnh sau vào INSTALLED\_APPS:

'corsheaders',



Tiếp theo, thêm CorsMiddleware vào danh sách MIDDLEWARE trong file settings.py, ngay trước dòng CommonMiddleware:

'corsheaders.middleware.CorsMiddleware',



Nếu muốn cho phép tất cả các domain gửi yêu cầu đến API, có thể thêm:

CORS\_ALLOW\_ALL\_ORIGINS = True

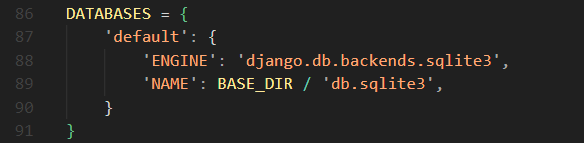


Sau khi hoàn tất các bước trên, Django sẽ cho phép các yêu cầu từ frontend React gửi đến backend mà không gặp lỗi CORS. Đây là một bước quan trọng để đảm bảo frontend và backend có thể giao tiếp với nhau trong quá trình phát triển ứng dụng.

### Cấu hình DATABASE trong setting.py

Trong Django, cấu hình cơ sở dữ liệu (database) được thực hiện thông qua **DATABASES** trong file **settings.py**. Django hỗ trợ nhiều loại cơ sở dữ liệu, bao gồm SQLite, PostgreSQL, MySQL, và Oracle.

Đây là dạng mặc định, cơ sở dữ liệu SQLite:



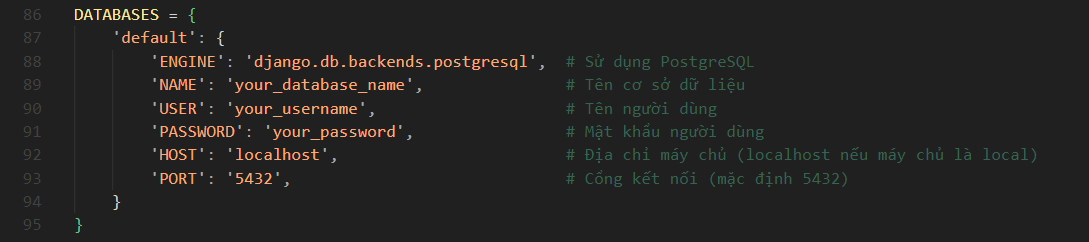
Nhưng ta không sử dụng SQLite, ta chọn sử dụng PostgreSQL:

Nếu muốn sử dụng **PostgreSQL**, cần thay đổi ENGINE và thêm các thông tin về kết nối (user, password, host, port).

Trước tiên, cần đảm bảo đã cài đặt thư viện PostgreSQL cho Django:

pip install psycopg2-binary django-cors-headers djangorestframework

Nhưng câu lệnh này đã được thực hiện ở trên, nên không cần thực hiện lại. Sau đó, thay đổi cấu hình cơ sở dữ liệu trong **settings.py** như sau:



Trước khi có thể chạy được ứng dụng Django, cần tạo cơ sở dữ liệu và người dùng tương ứng. Dưới đây là cách thực hiện:

Quay trở lại giai đoạn run file .exe, ta đã set up xong PostgreSQL. Mở cmd lên, gõ lệnh:

psql -U postgres để chạy PostgreSQL với quyền admin.

Sau đó, tạo cơ sở dữ liệu mới bằng dòng lệnh sau đây:

CREATE DATABASE your\_database\_name; // ví dụ thay your\_database\_name thành todo\_ds

Sau đó, tạo người dùng bằng câu lệnh sau:

CREATE USER your\_username WITH PASSWORD 'your\_password';

Ví dụ:

CREATE USER todo\_user WITH PASSWORD 'mypassword';

Sau đó, cấp quyền cho người dùng trên cơ sở dữ liệu bằng câu lệnh sau:

GRANT ALL PRIVILEGES ON DATABASE your\_database\_name TO your\_username;

Ví dụ:

GRANT ALL PRIVILEGES ON DATABASE todo\_db TO todo\_user;

Tiếp theo, ta phải kết nối vào database đã tạo bằng câu lệnh sau đây:

\c your\_database\_name

Ví dụ:

\c todo\_db

Sau đó, cấp quyền trên schema public bắng câu lệnh này:

GRANT ALL PRIVILEGES ON SCHEMA public TO your\_username;

Ví dụ:

GRANT ALL PRIVILEGES ON SCHEMA public TO todo\_user;

Cuối cùng, gõ lệnh sau để thoát khỏi PostgreSQL:

\q

Tiếp theo, trong file **settings.py**, cập nhật lại phần **DATABASES** như sau (dùng thông tin đã tạo ở bước trên):

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql', # Sử dụng PostgreSQL

'NAME': 'todo\_db', # Tên cơ sở dữ liệu

'USER': 'todo\_user', # Tên người dùng

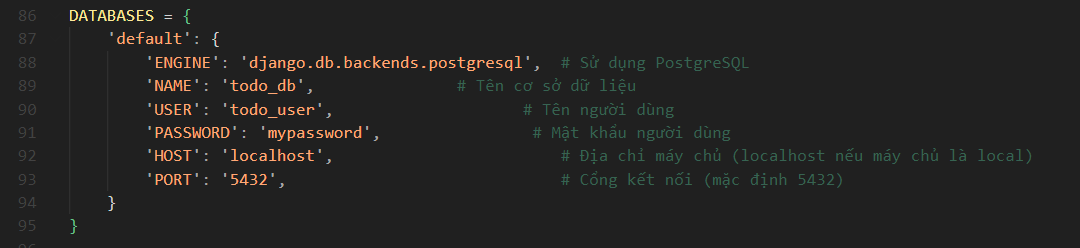
'PASSWORD': 'mypassword', # Mật khẩu người dùng

'HOST': 'localhost', # Máy chủ (nếu cục bộ thì dùng 'localhost')

'PORT': '5432', # Cổng (mặc định là 5432)

}

}



### Chạy lệnh thiết lập cơ sở dữ liệu

Sau khi cấu hình cơ sở dữ liệu trong file settings.py, bước tiếp theo là kết nối Django với cơ sở dữ liệu PostgreSQL và thiết lập các bảng cần thiết. Để thực hiện, chúng ta sử dụng hai lệnh quan trọng: **makemigrations** và **migrate**.

Django sẽ chuẩn bị các file migration để mô tả cách tạo bảng:

python manage.py makemigrations

Lệnh trên cần phải được chạy trong thư mục chứa file manage.py.

**Bước 2: Áp dụng migration**

Sau khi tạo file migration, cần áp dụng chúng lên cơ sở dữ liệu PostgreSQL. Chạy lệnh sau:

python manage.py migrate

Lệnh này đọc các file migration đã được tạo ra và áp dụng chúng lên cơ sở dữ liệu. Kết quả là các bảng sẽ được tạo hoặc sửa đổi theo định nghĩa trong model.

## Tạo model Task trong tasks/models.py

Model trong Django là phần quan trọng để định nghĩa cấu trúc dữ liệu mà bạn sẽ lưu trữ trong cơ sở dữ liệu. Mỗi Model trong Django đại diện cho một bảng trong cơ sở dữ liệu. Các trường trong Model sẽ tương ứng với các cột trong bảng đó.

Một Model trong Django là một lớp (class) trong Python kế thừa từ django.db.models.Model. Mỗi trường dữ liệu trong lớp này sẽ được định nghĩa như là một thuộc tính (attribute) của lớp, và mỗi thuộc tính này sẽ là một trường (field) trong bảng cơ sở dữ liệu.

Đầu tiên, bạn cần tạo một model Task để lưu trữ các công việc cần làm. Model này sẽ có các trường như tên công việc, trạng thái hoàn thành, ngày hết hạn, v.v.

Trong thư mục tasks, mở file models.py và định nghĩa model Task.

Các kiểu trường phổ biến trong Django:

|  |  |
| --- | --- |
| **Kiểu trường** | **Mô tả** |
| CharField | Trường lưu chuỗi ký tự. Thường được sử dụng cho tên hoặc trạng thái. Yêu cầu max\_length. |
| TextField | |  | | --- | | Trường lưu chuỗi văn bản dài. Không giới hạn độ dài. |  |  | | --- | |  | |
| IntegerField | Trường lưu số nguyên. |
| DateField | Lưu giá trị ngày. Đại diện trong Python bởi datetime.date |
| DateTimeField | Lưu ngày và giờ. Đại diện trong Python bởi datetime.datetime. |
| DecimalField | Lưu số thập phân với độ chính xác cố định. Cần chỉ định max\_digits (tổng số chữ số) và decimal\_places (số chữ số thập phân). |
| FloatField | Lưu số thực (dạng float trong Python). |
| EmailField | |  | | --- | | Một CharField kiểm tra giá trị hợp lệ của địa chỉ email. |  |  | | --- | |  | |
| TextField | Trường văn bản lớn. Dùng để lưu nội dung dài như bài viết. |
| GenericIPAddressField | |  | | --- | | Lưu địa chỉ IPv4 hoặc IPv6 dưới dạng chuỗi (vd: 192.0.2.30 hoặc 2a02:42fe::4). |  |  | | --- | |  | |
| SlugField | Lưu chuỗi văn bản an toàn cho URL (chỉ chứa chữ cái, số, dấu gạch dưới, dấu gạch ngang). |
| URLField | Một CharField dùng để lưu URL, kiểm tra hợp lệ với URLValidator. |

Các kiểu quan hệ trường trong Django:

|  |  |
| --- | --- |
| **Kiểu quan hệ** | **Mô tả** |
| ForeignKey | Thiết lập mối quan hệ **nhiều - một** (many-to-one).  Liên kết nhiều bản ghi trong một model tới một bản ghi trong model khác.  Yêu cầu:  - Đối số đầu tiên là class mà model đang liên kết.  - Đối số on\_delete để chỉ định hành vi khi đối tượng liên quan bị xóa. |
| ManyToManyField | Thiết lập mối quan hệ **nhiều - nhiều** (many-to-many). Liên kết nhiều bản ghi trong một model tới nhiều bản ghi trong model khác  Yêu cầu đối số đầu tiên là class mà model đang liên kết. Có thể dùng cho quan hệ đệ quy hoặc quan hệ trễ (lazy relationships). |
| OneToOneField | Thiết lập mối quan hệ **một - một** (one-to-one). Liên kết duy nhất giữa hai model (One-to-One). Tương tự ForeignKey với unique=True, nhưng phía “reverse” trả về duy nhất một đối tượng liên quan. |



A computer screen shot of text

Description automatically generated

A screen shot of a computer program

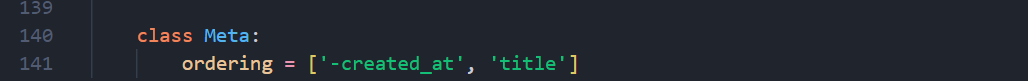
Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated



Sau khi đi code xong, Django sẽ tự động chuyển đổi Model thành các bảng cơ sở dữ liệu bằng cách sử dụng migrations bằng hai câu lệnh sau đây:

python manage.py makemigrations

python manage.py migrate

## Tạo file serializers.py (nếu dùng Django REST Framework)

Sau khi áp dụng migrations trong Django, không nhất thiết phải tạo file serializers ngay lập tức, trừ khi có nhu cầu triển khai API để giao tiếp với frontend hoặc các hệ thống khác qua HTTP (ví dụ như dùng Django REST Framework để xây dựng API).

Serializers chỉ cần khi cần chuyển đổi dữ liệu từ các mô hình Django thành định dạng JSON hoặc XML để gửi qua API hoặc chuyển từ dữ liệu nhận được từ API thành đối tượng Django. Nếu không xây dựng một API mà chỉ sử dụng ứng dụng web thông thường với giao diện HTML, thì không cần phải tạo file serializers.

Tóm lại, Serializer là công cụ trong Django REST Framework (DRF) để chuyển đổi dữ liệu:

* **Từ Model sang JSON/XML:** Gửi dữ liệu từ backend đến frontend hoặc API client.
* **Từ JSON/XML sang Model:** Nhận dữ liệu từ frontend hoặc API client và lưu vào cơ sở dữ liệu.

Tuy nhiên, nếu ta đang phát triển ứng dụng web có backend API, sẽ cần sử dụng Django REST Framework (DRF) để dễ dàng xây dựng API để giao tiếp với frontend React. React và các frontend hiện đại thường giao tiếp với backend qua các API (thường là RESTful API).

Ta cần tạo file serializers.py (trong thư mục app\_name nếu chưa có), sau đó định nghĩa Serializer cho các Model đã tạo.

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

**A screen shot of a computer program

Description automatically generated**

**A computer screen shot of text

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

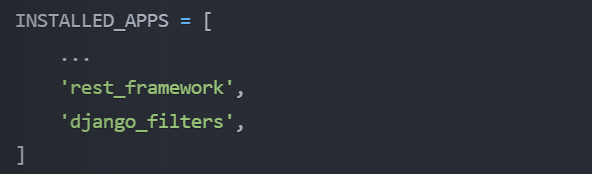
## Tạo Views (tasks/views.py)

Sau khi đã tạo serializers, tạo **ViewSets** hoặc **API views** để cung cấp API cho các mô hình.

Trong file code views.py này, ta có phải sử dụng thêm Django Filter, mà ở trên chưa cài vào môi trường, nên bây giờ phải thực hiện câu lệnh:

pip install django-filter

Sau đó, phải chỉnh sửa cấu hình file setting.py lại, bằng cách thêm 'django\_filters' vào trong danh sách INSTALLED\_APP:



Sau đó, thêm đoạn code này vào file setting.py:

REST\_FRAMEWORK = {

'DEFAULT\_FILTER\_BACKENDS': [

'django\_filters.rest\_framework.DjangoFilterBackend',

],

}

A screen shot of a computer

Description automatically generated

Dưới đây là nội dung file views.py:

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer screen shot of text

Description automatically generated

A screen shot of a computer code

Description automatically generated

Sau khi làm xong các bước trên, file views.py đã có thể chạy được.

## Cấu hình URL để truy cập API:

Trong thư mục app\_name cụ thể là tasks, tạo file urls.py (nếu chưa có) và thêm đường dẫn sau:

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import TaskViewSet, CategoryViewSet

# Tạo router cho ViewSets

router = DefaultRouter()

# Đăng ký các ViewSet

router.register(*r*'tasks', TaskViewSet, *basename*='task')

router.register(*r*'categories', CategoryViewSet, *basename*='category')

urlpatterns = [

    # Đường dẫn chính của API

    path('api/v1/', include(router.*urls*)),

    # Endpoint tùy chỉnh (nếu cần)

    # Ví dụ: path('api/v1/tasks/overdue/', TaskViewSet.as\_view({'get': 'overdue'}), name='task-overdue'),

]

# Các endpoint sẽ được tạo ra:

# GET    /api/v1/tasks/           - Danh sách tasks

# POST   /api/v1/tasks/           - Tạo task mới

# GET    /api/v1/tasks/{id}/      - Chi tiết task

# PUT    /api/v1/tasks/{id}/      - Cập nhật toàn bộ task

# PATCH  /api/v1/tasks/{id}/      - Cập nhật một phần task

# DELETE /api/v1/tasks/{id}/      - Xóa task

# Các custom actions:

# GET    /api/v1/tasks/overdue/   - Danh sách tasks quá hạn

# GET    /api/v1/tasks/statistics/ - Thống kê tasks

# PATCH  /api/v1/tasks/{id}/change\_status/ - Thay đổi trạng thái task

# Tương tự cho Categories

Tiếp theo, trong thư mục dự án, cụ thể là to\_do\_backend, mở file urls.py (đã có sẵn) và kết nối tasks/urls.py vào todo\_backend/urls.py:

"""

URL configuration for todo\_backend project.

The `urlpatterns` list routes URLs to views. For more information please see:

    https://docs.djangoproject.com/en/5.1/topics/http/urls/

Examples:

Function views

    1. Add an import:  from my\_app import views

    2. Add a URL to urlpatterns:  path('', views.home, name='home')

Class-based views

    1. Add an import:  from other\_app.views import Home

    2. Add a URL to urlpatterns:  path('', Home.as\_view(), name='home')

Including another URLconf

    1. Import the include() function: from django.urls import include, path

    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.*site*.*urls*),

    path('', include('tasks.urls')),

]

Và cuối cùng, trong file setting.py, cấu hình lại CORS

A screen shot of a computer

Description automatically generated

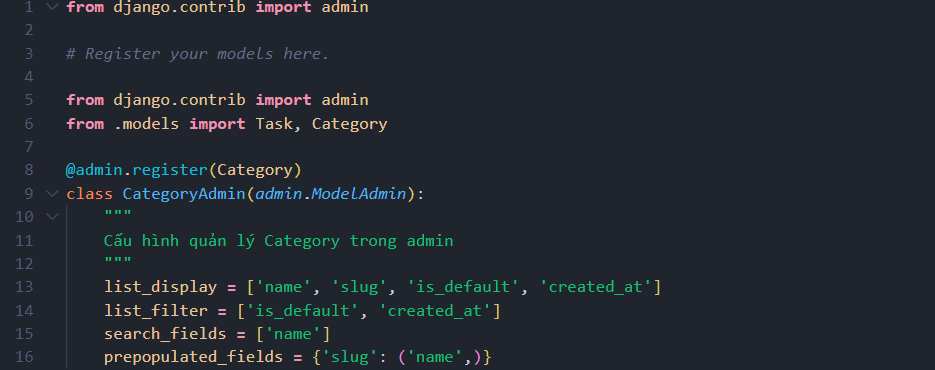
## Cấu hình file admin.py

Một trong những phần mạnh nhất của Django là giao diện admin. Django admin site đọc metadata từ các mô hình của chúng ta để cung cấp giao diện nhanh chóng, tập trung vào mô hình, nơi người dùng đáng tin cậy có thể quản lý nội dung trên trang web.

Django cung cấp một giao diện quản trị mặc định giúp ta dễ dàng quản lý dữ liệu trong cơ sở dữ liệu mà không cần tự xây dựng giao diện thủ công. File admin.py là nơi ta định nghĩa cách các model của bạn sẽ được hiển thị và quản lý trong giao diện này.

* **Đăng ký model:** Khi bạn đăng ký một model trong admin.py, Django sẽ biết rằng model đó cần xuất hiện trên giao diện admin.
* **Tùy chỉnh hiển thị:** Bạn có thể điều chỉnh:
  + Trường nào được hiển thị trong danh sách.
  + Trường nào có thể tìm kiếm, lọc hoặc chỉnh sửa.
  + Thêm các hành động (actions) nhanh, như xóa hàng loạt, đánh dấu hoàn thành.

Nếu không đăng ký model trong admin.py, bạn sẽ không thể quản lý model đó qua giao diện admin.



A screen shot of a computer program

Description automatically generated

A computer screen shot of green text

Description automatically generated

## Tạo tài khoản quản trị viên

Để truy cập vào trang quản trị của Django (Django Admin), cần tạo một tài khoản quản trị viên:

python manage.py createsuperuser

Hệ thống sẽ yêu cầu nhập:

* **Tên người dùng** (username): admin
* **Email** (không bắt buộc): trammaiquynhds@gmail.com
* **Mật khẩu:** 20092005 (phải xác thực lần hai)

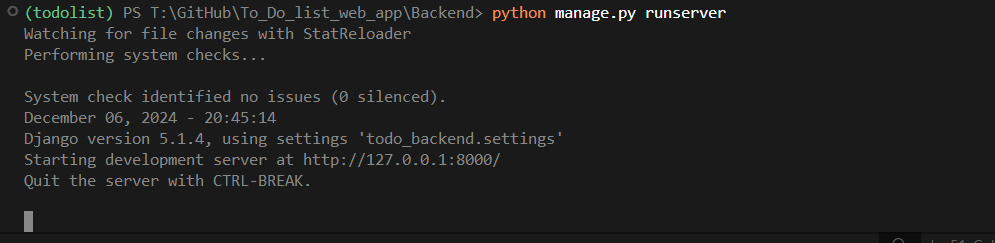
**Sau khi tạo tài khoản admin ta truy cập vào url cua**

## Chạy thử backend

Để đảm bảo không có bug, ta chạy thử backend trước khi qua xử lý front end bằng đoạn code sau:

python manage.py runserver

Tôi đã chạy thử thành công, màn hình terminal xuất hiện:



Ta có thể mở trình duyệt và truy cập vào địa chỉ sau để kiểm tra ứng dụng:

<http://127.0.0.1:8000/api/v1/tasks/>

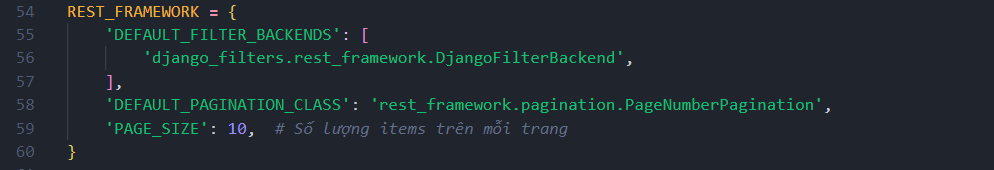
# Mở rộng

## Phân trang

Để quản lý **phân trang (pagination)** trong Django REST Framework (DRF), ta có thể làm theo các bước sau:

**1. Kích hoạt Pagination**

Pagination trong DRF được quản lý thông qua cài đặt DEFAULT\_PAGINATION\_CLASS. Bạn cần thêm cấu hình này vào file settings.py của dự án:



# Xử lý frontend

## Tạo thư mục dự án