

STUDI KASUS PORTAL BERITA

DJANGO REST API



RIZQI MAULANA

Table of contents

Kata Pengantar	3
Lisensi	4
Tentang Buku	5
Studi Kasus	6
Perancangan	6
Struktur Database	8
Persiapan dan Instalasi	10
Instalasi Python	10
Instalasi PostgreSQL	11
Instalasi Django dan Package Pendukungnya	13
Instalasi Postman	15
Instalasi Pycharm (Optional)	15
Instalasi GIT	16
Membuat Proyek Django	18
Membuat Proyek Baru	18
Pengaturan Konfigurasi	20
Models dan Migrations	23
Models	23
Migrations	25
Dashboard Admin	28
Modul Category	30
Modul User	32
Modul News	36
Rest API	43
API Endpoint Category	47
Category Serializers	48
Category Views	49
CategoryUrls	50

API Endpoint News	53
News Serializers	56
News Views	60
NewsUrls	65
Authentication	68
Testing API	79
Postman	80
Unit Testing	88
API Documentation	92
Mendeploy Aplikasi	105
Membuat Repository pada Github	105
Deploy ke Shared Hosting	112
Membuat Username dan Database PostgreSQL	112
Setup Aplikasi Python	113



Kata Pengantar

Bismillahirrahmannirrahiim. Assalamu'alaikum Warahmatullahi Wabarakatuh.

Alhamdulillah, segala puji dan syukur penulis panjatkan kehadiran Tuhan Yang Maha Esa karena berkat limpahan karunia-Nya, penulis dapat menyelesaikan penulisan buku ini. Shalawat dan salam untuk baginda Nabi Muhammad Saw. Tak lupa penulis juga mengucapkan terima kasih yang sebesar-besarnya kepada orang tua, keluarga,rekan-rekan dari tim Santrikoding yang telah mendukung penulis untuk menyelesaikan buku ini.

Penulis juga ingin menyampaikan terima kasih secara khusus untuk istri dan anak tercinta karena tak henti-hentinya menjadi penyemangat dan inspirasi untuk penulisan buku ini. Penulis menyadari apabila dalam penyusunan buku ini terdapat kekurangan, tetapi penulis meyakini sepenuhnya bahwa sekecil apapun buku ini tetap memberikan manfaat.

Akhir kata guna penyempurnaan buku ini kritik dan saran dari pembaca sangat penulis nantikan.

Wassalamu'alaikum Warahmatullahi Wabarakatuh.

Pekalongan, Februari 2021

Rizqi Maulana



Lisensi

Buku ini menggunakan license personal, yang artinya buku ini hanya boleh digunakan dan di baca untuk seseorang yang sudah membelinya. Selain pemilik license dari buku ini tidak diperbolehkan menggunakan apalagi sampai menyebarluaskan tanpa izin dari penulis.

Dan untuk pemilik license dari buku ini juga tidak di perbolehkan menyebarkan dan memperjualbelikan lagi kepada seseorang.

*Ilmu akan menjadi bermanfaat dan berkah ketika kita mempelajarinya dengan sesuatu yang halal
(tidak bajakan)*



Tentang Buku

Buku ini akan membahas pembuatan Rest API serta Dashboard Admin untuk sebuah web portal berita. Beberapa teknologi yang dipakai diantaranya: untuk database menggunakan PostgreSQL sedangkan untuk backendnya menggunakan Framework yang berbasis bahasa Python yaitu Django. Buku ini sendiri kami tujuhan untuk tingkat pemula sampai menengah. Sebelum mempelajari buku ini, kami harapkan pembaca sudah menguasai atau paling tidak pernah mempelajari dasar-dasar bahasa pemrograman Python.

Materi disusun secara step by step mulai dari persiapan dan instalasi sampai deploying aplikasi. Secara garis besar materi buku ini akan lebih banyak berkutat dalam membuat dashboard admin untuk mengelola data berita dan juga membuat rest api untuk data berita yang ada. Penulis juga menambahkan materi untuk membuat dokumentasi dari rest api tersebut agar nantinya akan mudah digunakan untuk tahap pengembangan seperti untuk mengintegrasikannya dengan aplikasi mobile.

Studi kasus yang dipakai dalam buku ini terbilang tidak terlalu rumit karena kami menyesuaikan dengan target pembaca buku ini (pemula sampai menengah). Tapi kami berharap buku ini menjadi pondasi awal yang baik bagi pembaca dalam mengembangkan aplikasi web menggunakan Django.

Studi Kasus

Perancangan

Dalam buku ini kita akan membuat sebuah proyek berdasarkan sebuah studi kasus. Kita akan merancang sebuah portal berita/*news* sederhana dimana nantinya di dalam proyek tersebut terdapat dua bagian penting yang akan kita bangun:

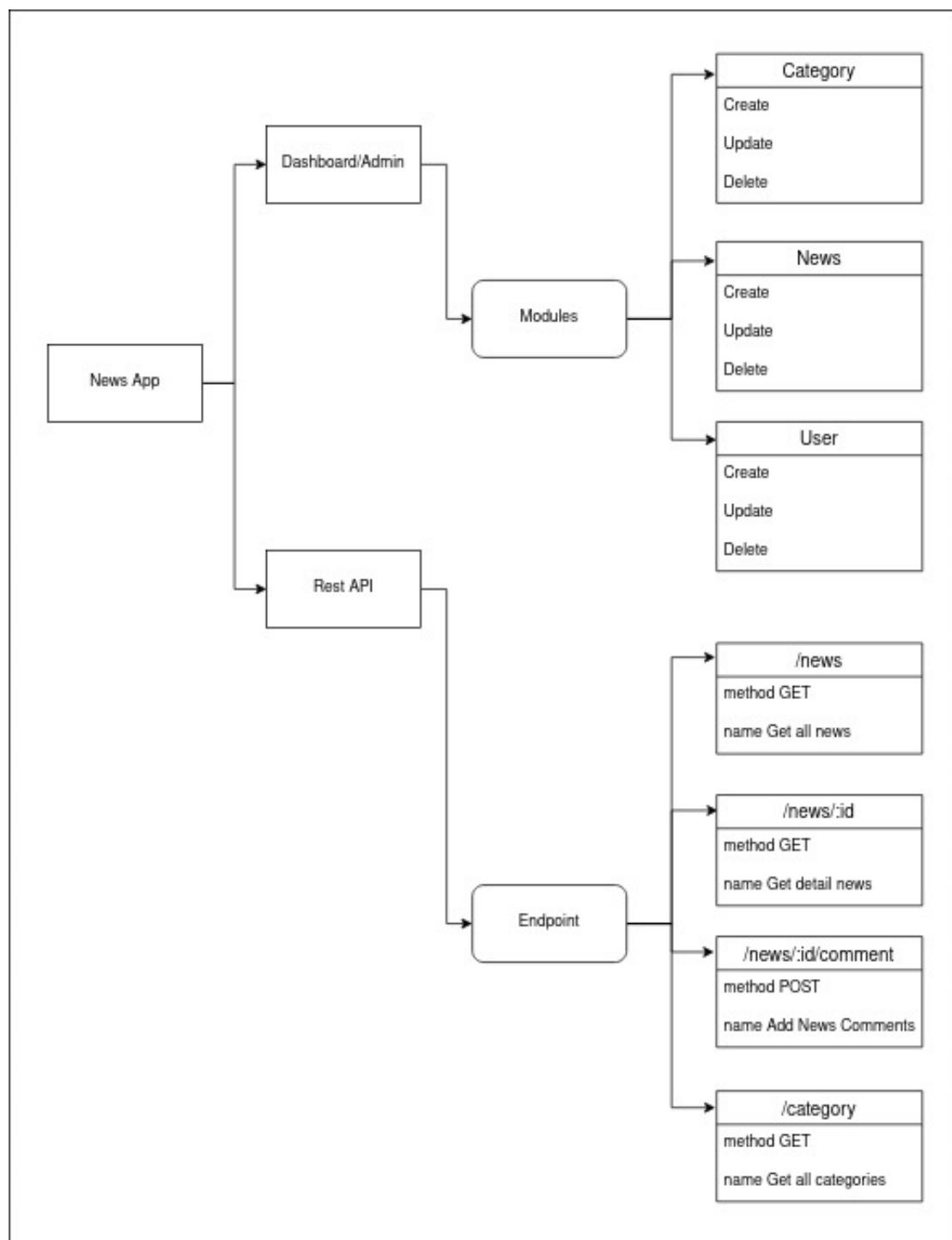
1. Dashboard Admin

Dalam sebuah portal berita tentunya kita perlu membuat sebuah fitur bagi admin dan *author* untuk menuliskan berita dan dipublish pada *platform* tersebut. Nantinya pada fitur ini *admin* ataupun *author* bisa mengelola data-data yang dibutuhkan untuk sebuah portal berita seperti kategori, berita dan juga membuat *user* atau akun baru bagi *author* agar dia bisa mempunyai akses untuk menuliskan berita ke dalam portal tersebut.

2. Rest API

Pada bagian ini kita akan fokus merancang dan membangun sebuah **Rest API** untuk proyek portal berita. Apa itu Rest API? secara sederhana bisa dikatakan sebagai sebuah *interface* yang bisa diakses melalui protokol **HTTP** dan bertugas menghubungkan satu aplikasi dengan aplikasi lainnya. Jadi Rest API ini nantinya bisa kita gunakan untuk menampilkan berita pada sebuah halaman website atau bisa menghubungkan portal berita dengan sebuah aplikasi **mobile (android/ios)**.

Tentunya karena buku ini dikhurasukan untuk pemula maka proyek yang akan kita buat tidak sekompel/serumit dengan sebuah portal berita sungguhan yang sudah ada. Gambar di bawah ini sedikit menjelaskan bagian-bagian dari proyek yang akan kita buat:



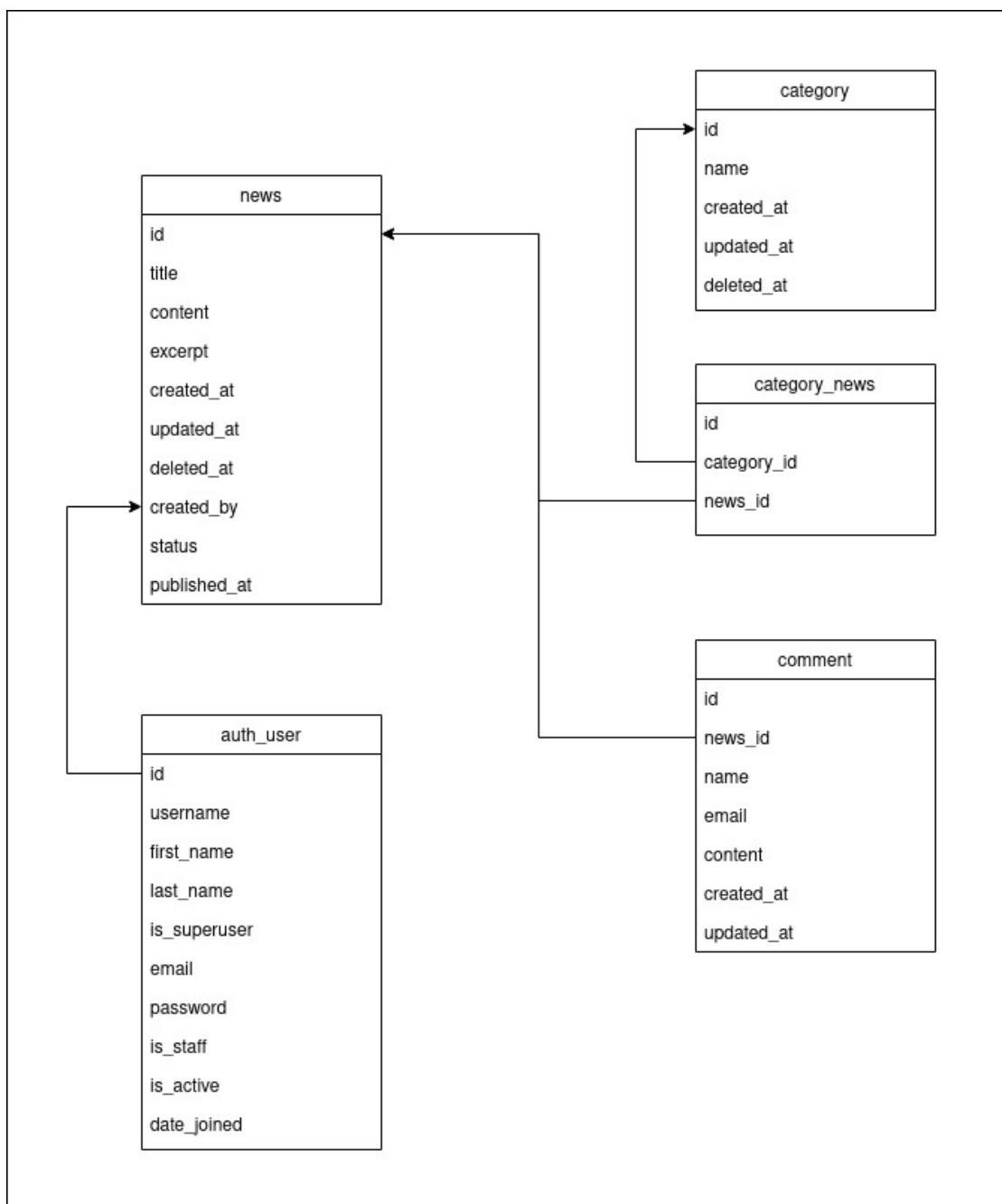
Gambar 1: Rancangan News App

Struktur Database

Pada tahap ini sebenarnya *optional* (kalian bisa melewati bagian ini). Tetapi alangkah baiknya jika kita terlebih dahulu merancang struktur database dari proyek yang akan kita buat. Kita perlu menganalisa data apa saja yang nantinya akan kita tampilkan atau kita simpan ke dalam *database*. Sebagai contoh pada studi kasus proyek kita tentang membuat sebuah portal berita, maka data seperti data berita, kategori dan penulis perlu kita jabarkan detailnya dari setiap datanya.

Di bawah ini contoh struktur *database* untuk portal berita yang akan kita buat:





Gambar 2: Struktur Database News App

Persiapan dan Instalasi

Sebelum memulai membangun sistem portal berita dengan framework **Django**, ada beberapa program yang perlu kita instal. Sistem operasi yang kita gunakan pada proses penulisan buku ini adalah **Ubuntu 20.04**. Jadi langkah-langkah instalasi yang akan dijelaskan dalam buku ini tentunya adalah langkah-langkah instalasi pada sistem operasi Ubuntu 20.04. Jika kamu menggunakan sistem operasi yang sama tapi dengan versi yang berbeda atau sistem operasi yang berbeda (**Windows atau Mac OS**), maka kamu perlu menyesuaikan dengan sistem operasi yang kamu pakai.

Pada setiap langkah-langkah instalasi akan kami sertakan link referensi untuk menginstal pada sistem operasi yang lain. kamu juga bisa bertanya pada grup konsultasi yang kami sediakan apabila kamu mengalami kendala pada materi persiapan dan instalasi ini.

Instalasi Python

Sistem operasi **Ubuntu 20.04** secara *default* sudah terinstal **Python** tetapi biasanya versi yang terinstal bukan versi yang terbaru. Di sini kita akan menginstal dengan versi yang terbaru yaitu 3.9.0 (versi python 3 terbaru pada saat buku ini ditulis). Silahkan ikuti langkah-langkah instalasinya di bawah ini:

Perbarui **package list** dari Ubuntu.

```
sudo apt update && sudo apt install software-properties-common
```

Tambahkan **PPA deadsnake** pada sistem.

```
sudo add-apt-repository ppa:deadsnakes/ppa
```

Dan terakhir instal Python versi 3.9.0.

```
sudo apt install python3.9
```

Untuk mengecek apakah Python sukses terinstal, jalankan perintah berikut.

```
python3.9 --version
```

Atau cukup dengan perintah

```
python --version
```

Catatan: Selanjutnya jika ada perintah `python3.9` maka kamu bisa juga menggantinya dengan `python` atau menyesuaikan versi yang kamu pakai. contohnya: `python3.8` , `python3.7` dst. contoh:

```
python3.9 -m venv env
```

bisa kamu ganti perintahnya seperti ini.

```
python -m venv env
```

Referensi untuk sistem operasi lain:

- **Windows:** <https://docs.python.org/3.9/using/windows.html>
- **Mac OS:** <https://docs.python.org/3.9/using/mac.html>

Instalasi PostgreSQL

PostgreSQL merupakan sistem manajemen relational database yang bersifat open source. Bagi kamu yang sudah belajar tentang database mungkin lebih familiar dengan MySQL dibanding PostgreSQL. Di sini kita akan memilih PostgreSQL sebagai sistem manajemen database dari sistem yang kita buat. Tetapi jika kamu merasa lebih nyaman menggunakan MySQL, itu bukan masalah karena nantinya sebagian besar pengolahan data seperti membaca, menambah, mengubah, dan menghapus akan dihadir oleh Django sebagai framework yang nantinya akan kita gunakan. Kamu nantinya cukup menyesuaikan saja dengan database pilihan kamu seperti dalam pengaturan setting konfigurasi koneksi ke database.

Untuk menginstal PostgreSQL pada Ubuntu 20.04 Silahkan buka terminal/command prompt dan jalankan perintah-perintah di bawah ini:

Pertama kita perlu membuat file konfigurasi untuk repository PostgreSQL.

```
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" >
↪ /etc/apt/sources.list.d/pgdg.list'
```

Masukkan repository signing key.

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
```

Update package list pada sistem kita.

```
sudo apt-get update
```

Terakhir ketikkan perintah di bawah ini untuk menginstal PostgreSQL versi terbaru.

```
sudo apt-get -y install postgresql
```

Untuk memeriksa apakah PostgreSQL sudah terinstal dengan benar, kita bisa mengeceknya dengan dua cara. Pertama dengan mengecek versi PostgreSQL yang terinstal pada sistem operasi kita.

```
psql --version
```

Kedua dengan mencoba masuk ke dalam PostgreSQL shell.

```
sudo -u postgres psql
```

Jika tampilan terminal/command prompt kamu berubah seperti ini, maka dipastikan PostgreSQL sudah terinstal dengan benar.

```
sakukode@sakukode:~$ sudo -u postgres psql
psql (13.1 (Ubuntu 13.1-1.pgdg20.04+1))
Type "help" for help.

postgres=#
```

Sebagai tambahan kita juga akan menginstal pgadmin4 sebagai management tools untuk PostgreSQL.

```
sudo apt-get install pgadmin4
```

Referensi untuk sistem operasi lain:

- <https://www.postgresql.org/download/windows/>
- <https://www.postgresql.org/download/macosx/>

Instalasi Django dan Package Pendukungnya

Kita akan menginstal **Django** dengan **package** pendukungnya menggunakan **tools package manager** di **Python** yaitu **PIP** dan juga kita akan menginstalnya di dalam sebuah folder proyek baru dengan **environment** khusus menggunakan **Virtual Environments** dari Python. Buka kembali terminal/command prompt:

Buat sebuah **folder** baru dengan nama **newsproject** dan masuk ke dalam **folder** tersebut.

```
mkdir newsproject && cd newsproject
```

Selanjutnya kita perlu menambahkan Virtual Environment pada **folder** tersebut.

```
python3.9 -m venv env
```

Aktifikan Virtual Environment dengan menjalankan perintah berikut.

```
source env/bin/activate
```

Untuk pengguna Sistem Operasi Windows. bisa menggunakan perintah berikut untuk mengaktifkan Virtual Environment.

```
env\Scripts\activate.bat
```

Tambahkan **file** baru dengan nama **requirements.txt** di dalamnya untuk menuliskan daftar **package** apa saja yang akan kita instal.

```
touch requirements.txt
```

Untuk pengguna Sistem Operasi Windows. bisa menggunakan perintah berikut.

```
type NUL > requirements.txt
```

Ubah file requirements.txt dan tuliskan *package-package* yang akan kita instal kemudian simpan kembali.

requirements.txt

```
Django == 3.1.4
psycopg2-binary == 2.8.6
djangorestframework == 3.12.2
Pillow == 8.0.1
django-filter == 2.4.0
drf-yasg
```

Setelah itu jalankan perintah berikut untuk menginstal *package-package* tersebut.

```
pip install -r requirements.txt
```

Berikut ini adalah *package-package* yang harus kita instal:

Nama Package	Versi	Deskripsi
Django	3.1.4	<i>web framework</i> yang nantinya akan kita gunakan sebagai <i>core</i> atau bagian inti dalam membangun proyek kita.
pyscopg2	2.8.6	<i>driver</i> untuk <i>database PostgreSQL</i> pada Python .
djangorestframe-work	3.12.2	<i>package</i> yang akan kita gunakan untuk membuat Rest API pada proyek kita.
Pillow	8.0.1	<i>package</i> yang akan kita gunakan untuk mengupload file gambar pada proyek kita.
django-filter	2.4.0	<i>package</i> untuk fitur filtering pada <i>package</i> djangorestframework
drf-yasg		<i>package</i> untuk mengintegrasikan <i>documentation API tool (Swagger & Redoc)</i> dengan <i>package</i> djangorestframework

Untuk mengecek apakah semua *package* sudah terinstal, bisa dengan menjalankan perintah berikut.

```
pip list
```

Jika instalasi berjalan sukses, seharusnya akan tampil seperti ini.

```
(.venv) sakukode@sakukode:~/newsproject$ pip list
Package           Version
-----
asgiref          3.3.1
Django            3.1.4
djangorestframework 3.12.2
pip               20.2.3
psycopg2-binary   2.8.6
pytz              2020.5
setuptools        49.2.1
sqlparse          0.4.1
```

Instalasi Postman

Postman merupakan *tool* yang nantinya akan kita gunakan untuk melakukan *testing Rest API* secara manual. Untuk menginstal Postman pada **Ubuntu 20.04** sangatlah mudah. Kita tinggal menjalankan perintah berikut dari *terminal*.

```
sudo snap install postman
```

Referensi untuk sistem operasi lain:

- <https://www.postman.com/downloads/> (Windows dan Mac OS)

Instalasi Pycharm (Optional)

Pycharm merupakan sebuah IDE yang dikembangkan khusus bagi para developer Python. Sebenarnya penggunaan Pycharm sebagai code editor dalam buku ini bersifat optional (tidak wajib). Kamu bisa menggunakan IDE atau code editor favorit kamu lainnya. Jadi silahkan lewati materi ini jika kamu lebih nyaman menggunakan IDE atau code editor pilihan kamu.

Untuk instalasi Pycharm pada Ubuntu 20.04, pertama kita perlu mendownload [tarball/installer-nya](#) terlebih dahulu. Ekstrak atau unpack *installer* ke dalam folder `/opt` pada sistem dengan menjalankan perintah berikut pada terminal dari *folder* lokasi *installer*-nya.

```
sudo tar xzf pycharm-*.tar.gz -C /opt/
```

Kemudian masuk ke dalam *subfolder bin* di dalam folder Pycharm *installer* yang sudah kita ekstrak.

```
cd /opt/pycharm-*/bin
```

Jalankan **pycharm.sh** dari *subfolder bin*

```
sh pycharm.sh
```

Atau jika kamu menggunakan Sistem Operasi Ubuntu 16.04 atau versi di atasnya kamu bisa menginstal Pycharm dengan **snap packages**. Jalankan perintah berikut untuk menginstal Pycharm dengan snap packages:

```
sudo snap install pycharm-community --classic
```

Referensi untuk sistem operasi lain:

- <https://www.jetbrains.com/pycharm/download/#section=windows>
- <https://www.jetbrains.com/pycharm/download/#section=mac>

Instalasi GIT

Git adalah *version control system* yang digunakan para developer untuk mengembangkan software secara bersama-bersama. Fungsi utama git yaitu mengatur versi dari source code program anda dengan mengasih tanda baris dan code mana yang ditambah atau diganti. Pada tutorial ini, GIT akan kita gunakan dalam proses deployment dimana nantinya kita tidak perlu mengcopy atau mengupload satu per satu file atau folder dari proyek kita. Untuk menginstal GIT pada Ubuntu 20.04 langkahnya sangat mudah. Pertama kita perlu mengupdate apt-repository dari sistem.

```
sudo apt-get update
```

Jalankan perintah di bawah ini untuk menginstal git.

```
sudo apt-get install git
```

Untuk mengecek GIT sudah sukses terinstal bisa dengan menggunakan perintah berikut.

```
git --version
```

Outputnya akan seperti di bawah ini jika GIT sudah terinstal.

```
sakukode@sakukode:~$ git --version
git version 2.25.1
```

Referensi untuk sistem operasi lain:

- <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

Membuat Proyek Django

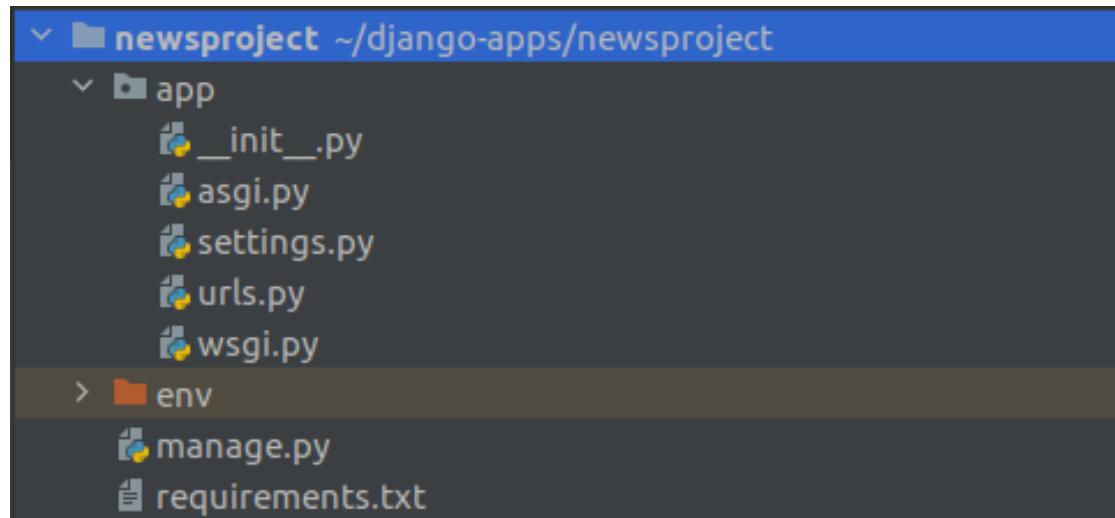
Membuat Proyek Baru

Pada materi sebelumnya kita sudah berhasil menginstal **Django** dan juga *package-package* pendukungnya pada *folder newsproject*. Selanjutnya pada materi ini kita akan memulai membuat sebuah proyek baru dengan *framework* Django dan melakukan pengaturan pada konfigurasinya.

Untuk membuat sebuah proyek baru dengan Django, ketikkan perintah berikut pada *terminal/command prompt* dari dalam folder newsproject.

```
django-admin startproject app .
```

Jika tidak terjadi kesalahan, seharusnya akan muncul sebuah folder baru dengan nama **app** dan juga beberapa file seperti gambar di bawah ini.



Gambar 3: Struktur folder dan file Proyek Django

Cek dan verifikasi proyek baru Django yang sudah kita buat apakah bisa berjalan atau tidak. Jalankan perintah berikut.

```
python3.9 manage.py runserver
```

Akan muncul tampilan seperti di bawah ini jika proyek Django berhasil dijalankan.

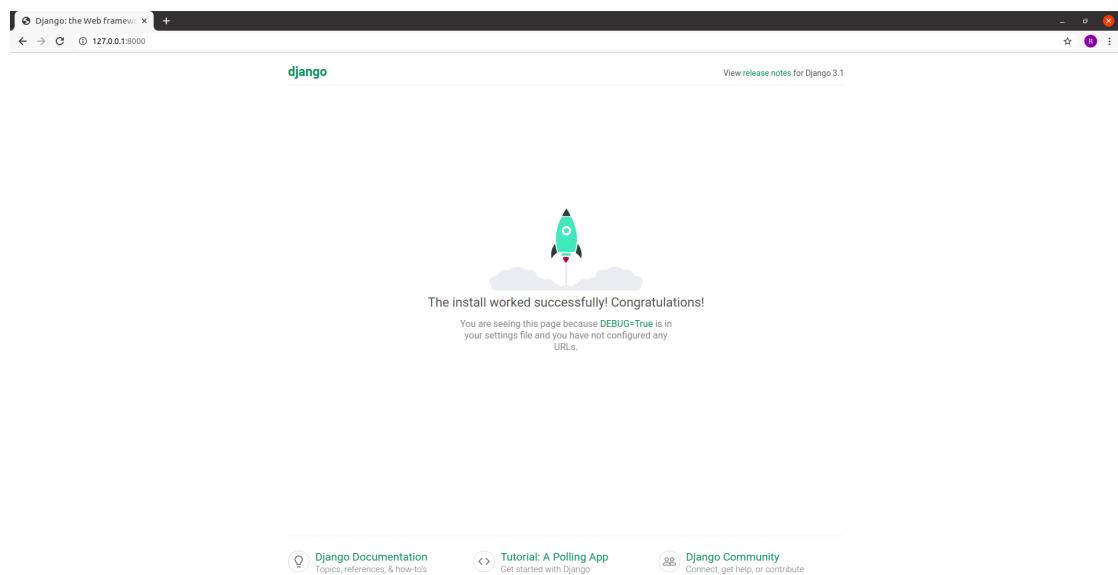
```
(env) sakukode@sakukode:~/django-apps/newsproject$ python3.9 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for
→ app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

December 25, 2020 - 23:07:54
Django version 3.1.4, using settings 'newsapp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Buka alamat <http://127.0.0.1:8000> dengan *browser* maka akan muncul halaman *web* seperti ini.



Gambar 4: Halaman Default Django

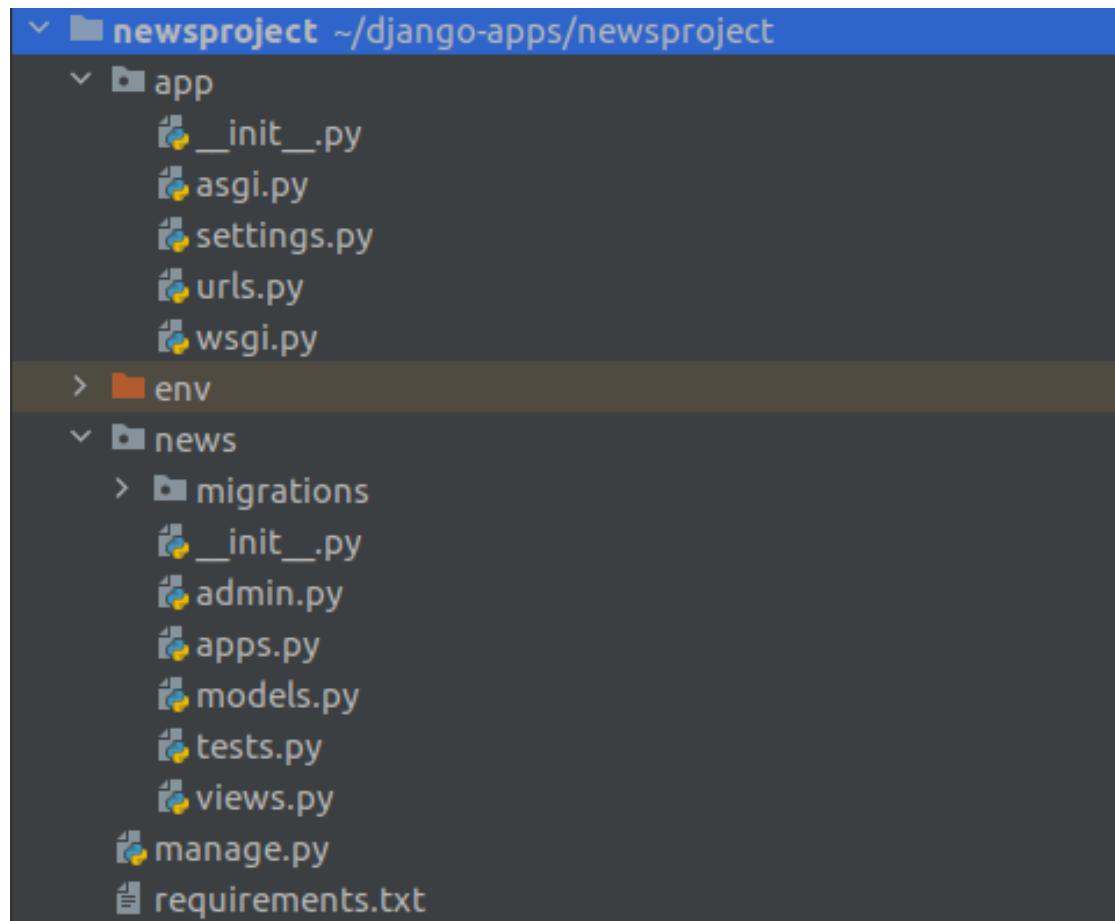
Untuk menghentikan proyek Django yang sedang berjalan, tekan kombinasi tombol **Ctrl + C** pada *keyboard*.

Selain membuat proyek baru pada Django, kita juga perlu membuat sebuah aplikasi di dalam proyek baru tersebut. Setiap proyek Django bisa memiliki banyak aplikasi di dalamnya. Untuk

membuat aplikasi di dalam proyek Django, ketikkan perintah berikut pada *terminal/command prompt*.

```
django-admin startapp news
```

Jika proses di atas berhasil, maka akan muncul sebuah *folder* baru lagi dengan nama **news** di dalam *folder* **newsproject**.



Gambar 5: Struktur folder dan file Aplikasi Django

Pengaturan Konfigurasi

Ada beberapa konfigurasi yang perlu kita atur di dalam proyek Django yang sudah kita buat. Buka file **settings.py** yang ada di lokasi *newsproject/app/settings.py*. Buka file tersebut dan ubah pada beberapa bagian.

Pertama kita perlu menambahkan aplikasi yang sudah kita buat di dalam proyek Django kita. Ubah bagian pengaturan di bawah ini.

```
INSTALLED_APPS = [
    # aplikasi kita
    'news.apps.NewsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Kedua kita perlu menyesuaikan pengaturan *database* pada bagian ini.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Secara *default* Django menggunakan *database* sqlite, sedangkan pada buku ini kita menggunakan PostgreSQL. Ubah dan sesuaikan dengan *driver database* yang kamu gunakan (MySQL, PostgreSQL, Oracle dll), nama *database*, *username*, *password*, *host*, dan *port* yang digunakan. Contoh konfigurasi yang kami gunakan.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'newsapp',
        'USER': 'postgres',
        'PASSWORD': '123456',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Terakhir kita akan mengatur *time zone* atau zona waktu dari proyek Django yang kita buat. ubah bagian ini.

```
TIME_ZONE = 'UTC'
```

Menjadi seperti ini

```
TIME_ZONE = 'Asia/Jakarta'
```



Models dan Migrations

Models

Models merupakan sebuah *class* pada Django yang berisi tentang informasi data pada *database* seperti nama tabel dan kolom dan juga perilaku lainnya seperti relasi antar tabel. Umumnya setiap satu *class* Models memetakan atau mewakili satu tabel pada *database*. Dengan referensi dari struktur database yang sudah kita buat pada materi sebelumnya, kita akan membuat beberapa Models yang akan memetakan tabel-tabel yang sudah kita definisikan antara lain Models untuk tabel category, news, news_categories, dan comment. Sedangkan untuk tabel auth_user itu akan otomatis tergenerate secara oleh Django.

Buka file **models.py** yang ada di lokasi *newsproject/news/models.py* dan ubah file tersebut menjadi seperti kode di bawah ini.

newsproject/news/models.py

```
from django.db import models
from django.contrib.auth.models import User

# Model untuk tabel category
class Category(models.Model):
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # set nama tabel
    class Meta:
        db_table = 'category'
        verbose_name_plural = "Category"

    def __str__(self):
        return self.name

# Model untuk tabel news
class News(models.Model):
    class NewsStatus(models.IntegerChoices):
        draft = 1
```

```
published = 2

title = models.CharField(max_length=255)
cover = models.ImageField(upload_to='images')
content = models.TextField()
excerpt = models.TextField()
status = models.IntegerField(choices=NewsStatus.choices)
published_at = models.DateTimeField(null=True)
created_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)

# set relasi ke tabel user and category
user = models.ForeignKey(User, on_delete=models.CASCADE)
categories = models.ManyToManyField(Category)

# set nama tabel
class Meta:
    db_table = 'news'
    verbose_name_plural = "News"

def __str__(self):
    return self.title

# Model untuk tabel comment
class Comment(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # set relasi ke tabel news
    news = models.ForeignKey(News, on_delete=models.CASCADE)

    # set nama tabel
    class Meta:
        db_table = 'comment'
```

Migrations

Migrations dibutuhkan untuk menggenerate/membuat tabel yang sudah kita definisikan sebagai Model pada tahap sebelumnya ke dalam *database*. Sebelum masuk ke tahap membuat dan menjalankan Migrations, kita perlu memastikan beberapa hal terlebih dahulu.

- Cek apakah nama *database*, *user*, *password*, *host*, dan *port* pada pengaturan konfigurasi proyek sudah benar.
- Pastikan kita sudah membuat *database* dengan nama sesuai yang ada pada konfigurasi proyek. Jika belum ada, silahkan ikuti langkah di bawah ini untuk membuat *database* baru pada PostgreSQL. Buka *terminal/command prompt* dan jalankan perintah ini.

```
sudo -u postgres psql postgres
```

Perintah di atas adalah perintah untuk masuk ke dalam PostgreSQL Shell. jika berhasil masuk, seharusnya tampilan *terminal/command prompt* kita akan seperti ini.

```
sakukode@sakukode:~$ sudo -u postgres psql postgres
psql (13.1 (Ubuntu 13.1-1.pgdg20.04+1))
Type "help" for help.

postgres=#
```

Untuk membuat *database* baru sangatlah mudah, ketikkan perintah berikut pada PostgreSQL Shell.

```
CREATE DATABASE newsapp;
```

Untuk keluar dari PostgreSQL Shell, ketikkan perintah.

```
exit
```

Oke Buka kembali *terminal/command prompt* dan jalankan perintah berikut untuk membuat Migrations pada Django.

```
python3.9 manage.py makemigrations news
```

Pada perintah di atas kita menggunakan perintah pada Django untuk membuat *file* Migrations baru untuk aplikasi news. Cek ke dalam *folder* *newsproject/news/migrations*. Seharusnya akan

muncul sebuah *file* baru dengan nama seperti *0001_initial.py*. Jika belum muncul *file* tersebut silahkan cek kembali *file* *models.py*, pastikan kamu sudah mengubahnya dengan benar.

Dan ketikkan perintah di bawah ini di terminal untuk menjalankan Migrations yang sudah kita buat.

```
python3.9 manage.py migrate
```

Jika Migrations berhasil dijalankan dan tidak ada *error*, tampilan di *terminal* kurang lebih akan seperti ini.

```
(env) sakukode@sakukode:~/django-apps/newsproject$ python3.9 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, newsapp, sessions
Running migrations:
  Applying newsapp.0001_initial... OK
```

Untuk memastikan bahwa tabel benar-benar sudah tergenerate, kita akan mengeceknya melalui PostgreSQL Shell. Masuk kembali PostgreSQL Shell dengan perintah berikut.

```
sudo -u postgres psql postgres
```

Setelah masuk ke PostgreSQL Shell pilih *database* dari proyek kita dengan perintah.

```
\c newsapp
```

Kemudian ketikkan perintah berikut untuk melihat daftar tabel di dalam *database* kita.

```
\dt
```

Cek apakah pada daftar sudah ada tabel category,comment,news,news_categories. Jika sudah ada, berarti migrations yang kita buat sudah berhasil dijalankan.

```
          List of relations
 Schema |      Name      | Type | Owner
-----+-----+-----+-----
```

```
public | auth_group           | table | postgres
public | auth_group_permissions | table | postgres
public | auth_permission       | table | postgres
public | auth_user             | table | postgres
public | auth_user_groups     | table | postgres
public | auth_user_user_permissions | table | postgres
public | category              | table | postgres
public | comment               | table | postgres
public | django_admin_log      | table | postgres
public | django_content_type   | table | postgres
public | django_migrations     | table | postgres
public | django_session         | table | postgres
public | news                   | table | postgres
public | news_categories       | table | postgres
(14 rows)
```



Dashboard Admin

Django Admin merupakan fitur yang bisa menggenerate sebuah halaman Dashboard Admin dengan praktis dan mudah. Halaman Admin ini bisa digunakan oleh *admin* atau *staff* dari sebuah proyek atau aplikasi yang kita buat untuk mengelola konten yang ada di dalam aplikasi tersebut. Seperti pada studi kasus kita dalam membangun proyek portal berita. Tentunya kita perlu memiliki halaman Admin untuk mengelola berita yang akan dipublish kepada pembaca.

Halaman Admin pada Django bersifat *private* atau terproteksi sehingga kita memerlukan *user* untuk mengaksesnya. Untuk membuat *user* admin kita bisa membuatnya dengan menjalankan perintah berikut pada *terminal/command prompt*.

```
python3.9 manage.py createsuperuser
```

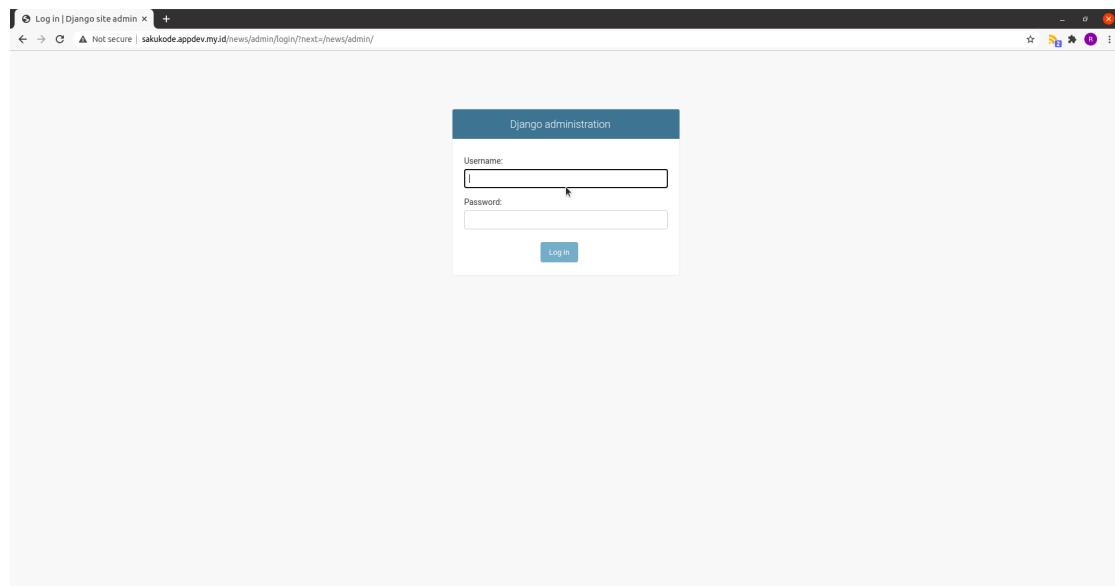
akan muncul *dialog* dan kita diharuskan memasukkan *username*, *email* dan *password* yang akan kita gunakan.

```
(env) sakukode@sakukode:~/myprojects/newsproject$ python3.9 manage.py createsuperuser
Username (leave blank to use 'sakukode'): admin
Email address: sakukode@gmail.com
Password:
Password (again):
Superuser created successfully.
(env) sakukode@sakukode:~/myprojects/newsproject$
```

Setelah itu jalankan proyek Django kita.

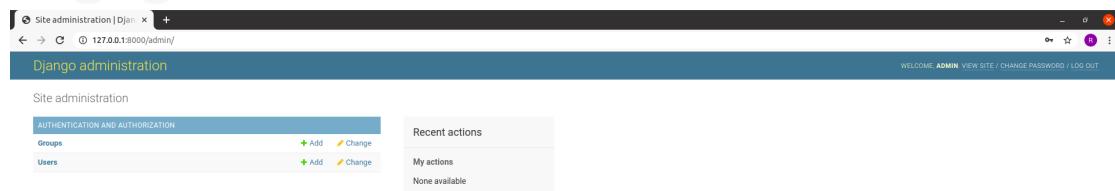
```
python3.9 manage.py runserver
```

Dan buka alamat *url* berikut pada *browser* <http://127.0.0.1:8000/admin>. Akan muncul halaman untuk *login* ke Django Admin.



Gambar 6: Halaman Login Admin News App

Silahkan *login* menggunakan *username* dan *password* yang sudah kita buat sebelumnya. Setelah berhasil *login*, kita akan diarahkan menuju halaman beranda Django Admin. Pada halaman beranda kita bisa melihat ada 2 menu yang sudah ada secara *default* yaitu menu untuk modul Groups dan Users. Selanjutnya kita akan menambahkan menu untuk modul lainnya.



Gambar 7: Halaman Beranda Admin News App

Modul Category

Untuk menambahkan modul lain ke Django Admin kita perlu mengubah file **admin.py** (*newsproject/news/admin.py*). Tambahkan atau daftarkan modul *Category* pada Django Admin dengan menambahkan kode berikut.

newsproject/news/admin.py

```
from django.contrib import admin
# import class model Category
from .models import Category

# Menambahkan modul Category ke Admin
class CategoryAdmin(admin.ModelAdmin):
    # Menampilkan kolom "name" dan "created_at" pada halaman Category list di Admin
    list_display = ('name', 'created_at')
    # Menambahkan fitur pencarian berdasarkan kolom "name"
    search_fields = ['name']

# Mendaftarkan CategoryAdmin
admin.site.register(Category, CategoryAdmin)
```

Berikut penjelasan dari kode di atas. Ada 3 bagian yang akan kita bahas. Yang pertama pada bagian ini.

```
# import class model Category
from .models import Category
```

Pada kode di atas, kita melakukan *import class Category* model yang sudah kita definisikan pada file *models.py*.

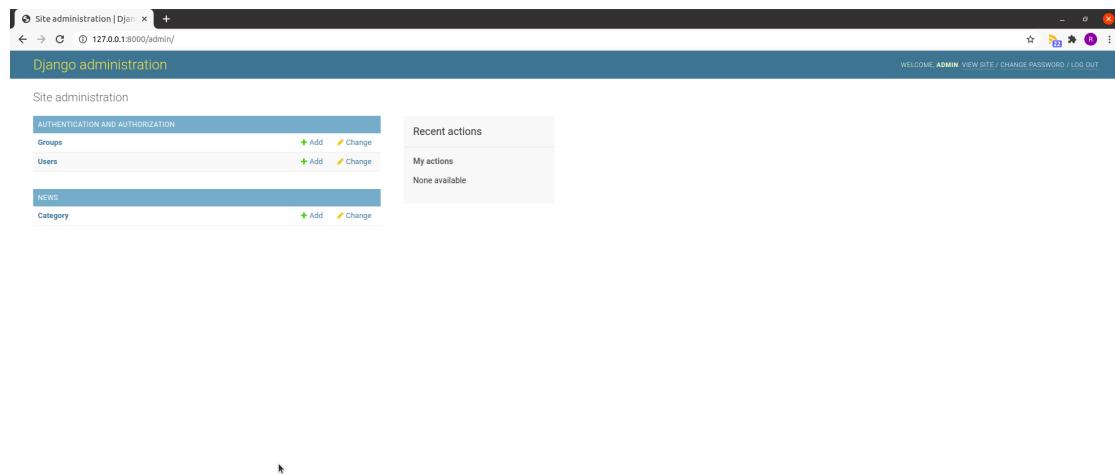
```
# Menambahkan modul Category ke Admin
class CategoryAdmin(admin.ModelAdmin):
    # Menampilkan kolom "name" dan "created_at" pada halaman Category list di Admin
    list_display = ('name', 'created_at')
    # Menambahkan fitur pencarian berdasarkan kolom "name"
    search_fields = ['name']
```

Kemudian pada bagian kedua (kode di atas). Kita membuat sebuah *class CategoryAdmin* yang merupakan turunan dari *class admin.ModelAdmin*. Kita perlu membuat *class* ini jika kita ingin menampilkan modul *Category* pada halaman Django Admin. Di dalam *class* itu kita juga mengeset apa saja kolom yang akan tampil di halaman daftar *Category* dan menambahkan fitur pencarian berdasarkan kolom *name*.

```
# Mendaftarkan CategoryAdmin
admin.site.register(Category, CategoryAdmin)
```

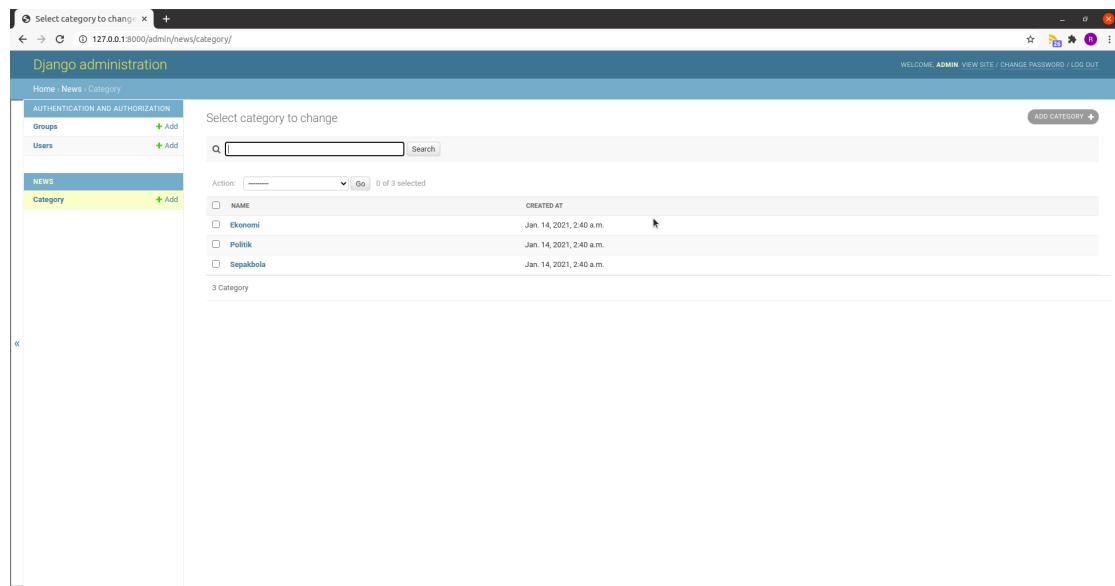
Dan bagian terakhir dengan satu baris kode di atas berfungsi untuk mendaftarkan *class CategoryAdmin* yang sudah kita buat sebelumnya ke Django Admin.

Buka kembali alamat <http://127.0.0.1:8000/admin> dengan *browser* dan *refresh* maka pada halaman beranda Admin akan otomatis muncul sebuah menu baru *Category* di bawah grup menu **NEWS**.



Gambar 8: Halaman Beranda Admin News App

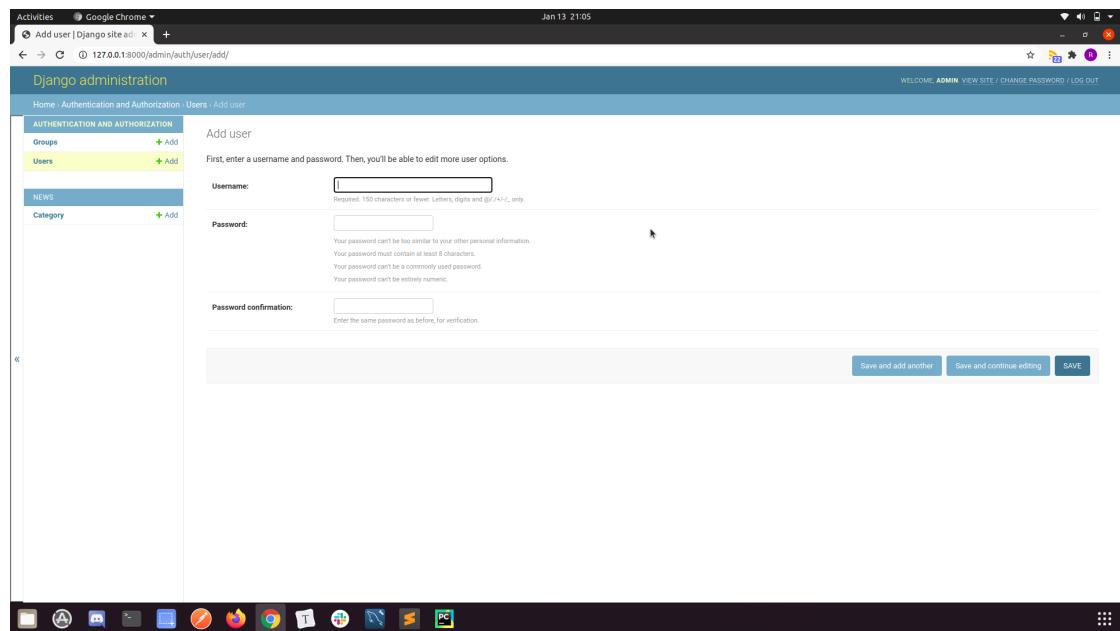
Klik pada menu **Category** dan kita akan diarahkan ke halaman modul *Category*. dari halaman tersebut kita bisa melihat daftar data *Category* dan juga bisa melakukan pengolahan data lain seperti menambah, mengubah dan menghapus data.



Gambar 9: Halaman Category Admin News App

Modul User

Secara default Django sudah menggenerate modul User pada halaman Django Admin. Sehingga jika kita lihat pada halaman beranda Admin sudah terdapat menu untuk modul User. Untuk itu di sini kita hanya akan sedikit mengubah pada bagian *form* User nya saja. Pada gambar di bawah ini hanya ada kolom *Username*, *Password*, dan *Password Confirmation*. Kita akan menambahkan 3 kolom lagi yaitu *First Name*, *Last Name* dan juga *Email Address*.



Gambar 10: Halaman User Form Admin News App

Buka kembali file `newsproject/news/admin.py` dan ubah kodennya menjadi seperti di bawah ini.

newsproject/news/admin.py

```
from django.contrib import admin
# import class yang dibutuhkan untuk memodifikasi form pada User Admin
from django.contrib.auth.admin import UserAdmin
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from django.contrib.auth.models import User
# import class model Category
from .models import Category

# Menambahkan modul Category ke Admin
class CategoryAdmin(admin.ModelAdmin):
    # Menampilkan kolom "name" dan "created_at" pada halaman Category list di Admin
    list_display = ('name', 'created_at')
    # Menambahkan fitur pencarian berdasarkan kolom "name"
    search_fields = ['name']
```

```
# Mendaftarkan CategoryAdmin
admin.site.register(Category, CategoryAdmin)

# Membuat Custom Form untuk Form Add User
class UserCreateForm(UserCreationForm):

    class Meta:
        model = User
        fields = ('username', 'first_name', 'last_name', )

# Membuat Custom Form untuk Form Update User
class UserUpdateForm(UserChangeForm):

    class Meta:
        model = User
        fields = ('username', 'first_name', 'last_name', )

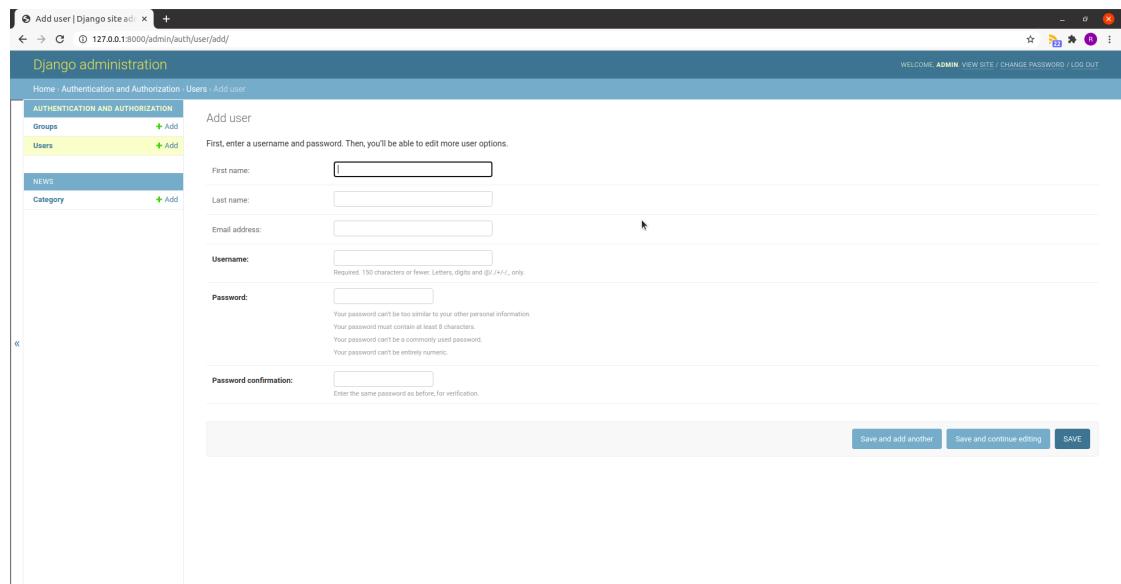
# Mengeset Custom User Form ke modul User Admin
class UserAdmin(UserAdmin):
    add_form = UserCreateForm
    form = UserUpdateForm
    prepopulated_fields = {'username': ('first_name', 'last_name', )}

    add_fieldsets = (
        (None, {
            'classes': ('wide',),
            'fields': ('first_name', 'last_name', 'email', 'username', 'password1', 'password2', ),
        }),
    )

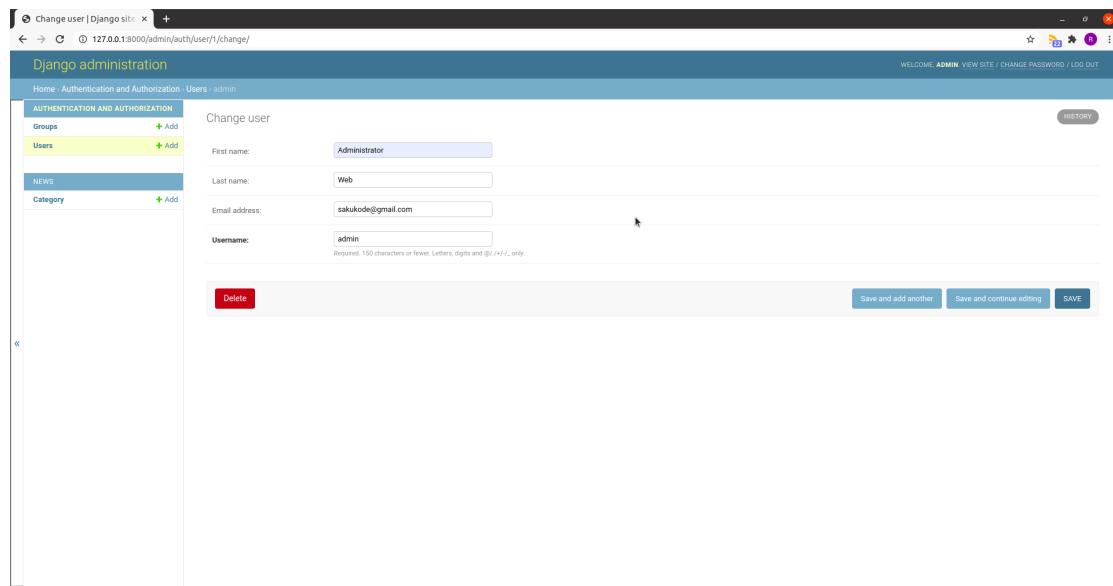
    fieldsets = (
        (None, {
            'classes': ('wide',),
            'fields': ('first_name', 'last_name', 'email', 'username', ),
        }),
    )
```

```
# Mendaftarkan ulang User Admin
admin.site.unregister(User)
admin.site.register(User, UserAdmin)
```

Coba buka kembali halaman *form User* di Halaman Django Admin (<http://127.0.0.1:8000/admin/auth/user/add/>) dan kita akan melihat 3 tambahan kolom baru (*First Name, Last Name, Email Address*).



Gambar 11: Halaman Create User Form Admin News App



Gambar 12: Halaman Update User Form Admin News App

Modul News

Modul terakhir yang akan kita tambahkan ke dalam Django Admin adalah modul *News/Berita*. Modul *News* memiliki kolom yang cukup banyak dan akan ada beberapa perbedaan dibanding modul-modul sebelumnya. Sedikit rumit tapi tidak perlu khawatir karena kita akan mempelajarinya dengan cara yang bisa mudah untuk dipahami. Sama seperti langkah sebelumnya, kita perlu mengubah lagi file *newsproject/news/admin.py* dan menambahkan beberapa baris kode untuk menambahkan modul *News* ke dalam Django Admin.

newsproject/news/admin.py

```
from django.contrib import admin
# import class yang dibutuhkan untuk memodifikasi form pada User Admin
from django.contrib.auth.admin import UserAdmin
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from django.contrib.auth.models import User
# import helper timezone untuk mengeset waktu published_at
from django.utils import timezone

# import class model Category
from .models import Category
```

```
# import class model News
from .models import News


# Menambahkan modul Category ke Admin
class CategoryAdmin(admin.ModelAdmin):
    # Menampilkan kolom "name" dan "created_at" pada halaman Category list di Admin
    list_display = ('name', 'created_at')
    # Menambahkan fitur pencarian berdasarkan kolom "name"
    search_fields = ['name']


# Mendaftarkan CategoryAdmin
admin.site.register(Category, CategoryAdmin)


# Membuat Custom Form untuk Form Add User
class UserCreateForm(UserCreationForm):

    class Meta:
        model = User
        fields = ('username', 'first_name', 'last_name', ) 


# Membuat Custom Form untuk Form Update User
class UserUpdateForm(UserChangeForm):

    class Meta:
        model = User
        fields = ('username', 'first_name', 'last_name', ) 


# Mengeset Custom User Form ke modul User Admin
class UserAdmin(UserAdmin):
    add_form = UserCreateForm
    form = UserUpdateForm
    prepopulated_fields = {'username': ('first_name', 'last_name', )}

    add_fieldsets = (
        (None, {
```

```
'classes': ('wide',),
'fields': ('first_name', 'last_name', 'email', 'username', 'password1', 'password2', ),
}),
)

fieldsets = (
(None, {
'classes': ('wide',),
'fields': ('first_name', 'last_name', 'email', 'username', ),
}),
)

# Mendaftarkan ulang User Admin
admin.site.unregister(User)
admin.site.register(User, UserAdmin)

# Menambahkan modul News ke Admin
class NewsAdmin(admin.ModelAdmin):
    # Menampilkan kolom "title", "status", "user" dan "created_at" pada halaman News list di Admin
    list_display = ('title', 'status', 'user', 'created_at')
    # Menampilkan kolom "title", "content", "excerpt", "cover", "status", dan "categories"
    # pada halaman Add & Update News di Admin
    fields = ('title', 'content', 'excerpt', 'cover', 'status', 'categories')
    # Menambahkan fitur pencarian berdasarkan kolom "title"
    search_fields = ['title']
    # Menambahkan fitur filter data berdasarkan kolom "status"
    list_filter = ('status', )

    # override method save_model untuk menyimpan data user dan published_at secara otomatis.
    def save_model(self, request, obj, form, change):
        # menyimpan kolom "user" berdasarkan user yang login
        if not obj.user_id:
            obj.user = request.user

        if form.cleaned_data.get('status') == 1:
            # mengeset kolom "published_at" dengan nilai null jika status bernilai 1 (draft)
            obj.published_at = None
        else:
            obj.published_at = datetime.datetime.now()
```

```
# mengeset kolom "published_at" dengan tanggal&waktu sekarang jika status bernilai 2 (published)
obj.published_at = timezone.now()

obj.save()

# Mendaftarkan NewsAdmin
admin.site.register(News, NewsAdmin)
```

Baris-baris kode di atas adalah perubahan kode yang kita tulis di dalam *file admin.py*. Ada beberapa bagian yang akan kita bahas pada perubahan kode di atas. Untuk bagian pertama kita menambahkan 2 baris kode berikut di bagian *import class*.

```
# import helper timezone untuk mengeset waktu published_at
from django.utils import timezone

...
# import class model News
from .models import News
```

Baris pertama di atas kita gunakan untuk mengimport sebuah *package* atau *helper timezone*. *Helper* ini kita gunakan untuk menggenerate tanggal waktu sekarang (*now*) dan nantinya akan kita gunakan pada bagian berikutnya sedangkan pada baris selanjutnya di atas adalah kode untuk mengimport *class model News* karena modul *News* sendiri merupakan interpretasi dari model/tabel *News*. Pada bagian kedua kita membuat sebuah *class* bernama *NewsAdmin*.

```
# Menambahkan modul News ke Admin
class NewsAdmin(admin.ModelAdmin):
    # Menampilkan kolom "title", "status", "user" dan "created_at" pada halaman News list di Admin
    list_display = ('title', 'status', 'user', 'created_at')
    # Menampilkan kolom "title", "content", "excerpt", "cover", "status", dan "categories"
    # pada halaman Add & Update News di Admin
    fields = ('title', 'content', 'excerpt', 'cover', 'status', 'categories')
    # Menambahkan fitur pencarian berdasarkan kolom "title"
    search_fields = ['title']
    # Menambahkan fitur filter data berdasarkan kolom "status"
    list_filter = ('status', )

    # override method save_model untuk menyimpan data user dan published_at secara otomatis.
```

```
def save_model(self, request, obj, form, change):
    # menyimpan kolom "user" berdasarkan user yang login
    if not obj.user_id:
        obj.user = request.user

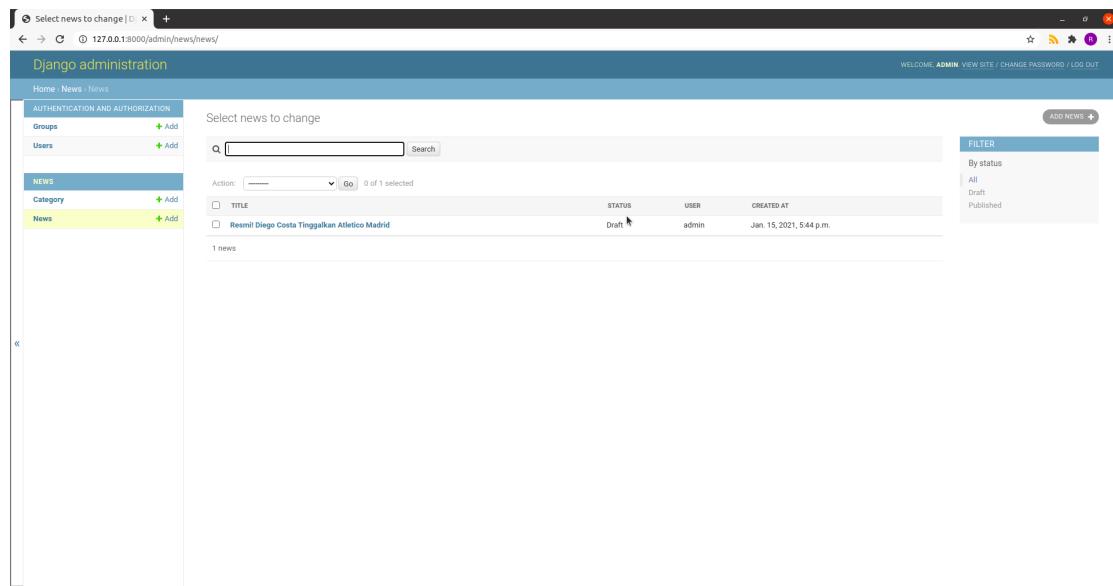
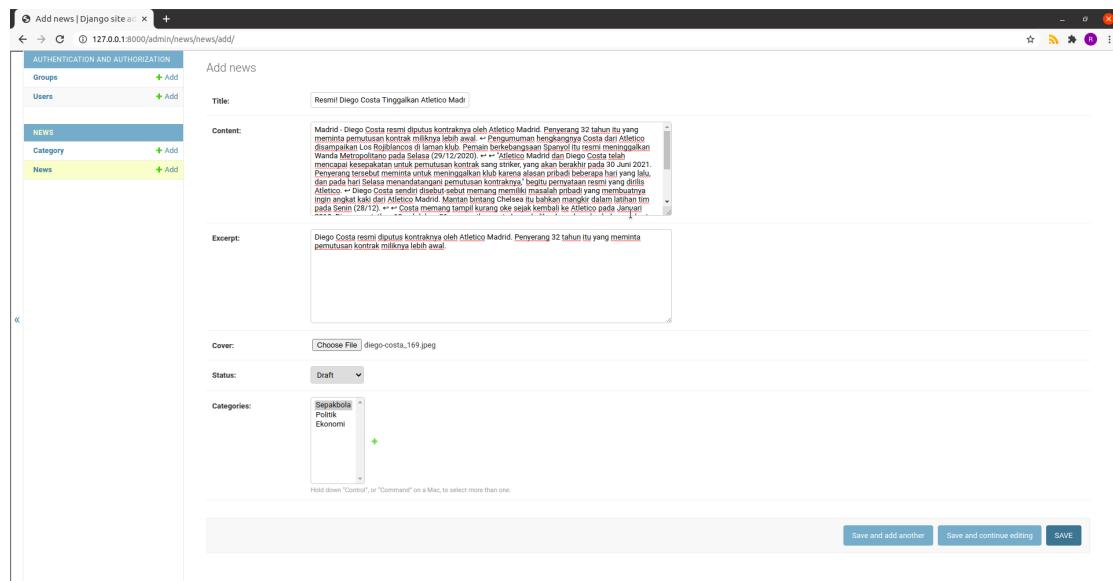
    if form.cleaned_data.get('status') == 1:
        # mengeset kolom "published_at" dengan nilai null jika status bernilai 1 (draft)
        obj.published_at = None
    else:
        # mengeset kolom "published_at" dengan tanggal&waktu sekarang jika status bernilai 2 (published)
        obj.published_at = timezone.now()

    obj.save()
```

Bagian ini cukup penting karena kita melakukan beberapa pengaturan seperti kolom apa saja yang akan ditampilkan pada halaman *list* maupun *form* untuk *news/berita*. Kita juga menambahkan fitur pencarian dan *filtering* berdasarkan kolom tertentu. Dan terakhir kita menulis ulang (*override*) method **save_model**. Method *save_model* ini merupakan *method* yang dijalankan ketika kita mensubmit *form* pada halaman *add* atau *change News* di Django Admin. Pada *Method* tersebut kita menambahkan kode untuk menyimpan kolom *user* berdasarkan *user* yang sedang *login* dan mengeset kolom *published_at* secara otomatis(*null* jika status adalah *draft*, dan terisi tanggal&waktu sekarang (menggunakan *helper timezone*) jika status adalah *published*). Dan terakhir kita mendaftarkan *NewsAdmin* ke dalam Django Admin.

```
# Mendaftarkan NewsAdmin
admin.site.register(News, NewsAdmin)
```

Berikut tampilan dari halaman *list* dan *form* dari modul *News* di Django Admin.

**Gambar 13:** Halaman News List Admin News App**Gambar 14:** Halaman News Form Admin News App

Kami harap sedikit penjelasan di atas dapat membantu kamu untuk memahami baris-baris kode yang sudah kita tuliskan pada bagian materi kali ini.

Pada materi selanjutnya kita akan memulai mengerjakan bagian penting yang lain dari proyek

kita, yaitu membangun **Rest API** untuk proyek portal berita kita.



Rest API

Pada bab ini kita akan fokus untuk membangun Rest API untuk proyek portal *news*/berita menggunakan *Django* dan juga *djangorestframework*. Sebelumnya pada bab persiapan dan instalasi kita sudah menginstal dua *package* tersebut menggunakan PIP. Namun pastikan kembali kalau dua *package* tersebut sudah terinstal pada *virtual environment* proyek kita.

Kita perlu melakukan pengaturan setting proyek *Django* terlebih dahulu agar kita bisa menjalankan *djangorestframework* dalam proyek *Django* kita. Buka kembali file *newsproject/app/settings.py* dan tambahkan kode pada beberapa bagian seperti di bawah ini.

Tambahkan *rest_framework* dan *django_filters* pada variabel **INSTALLED_APPS**.

```
INSTALLED_APPS = [
    # package untuk rest api
    'rest_framework',
    # package untuk fitur filtering pada rest api
    'django_filters',
    # aplikasi news app
    'news.apps.NewsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Dan tambahkan kode di bawah ini pada bagian paling bawah dari file *settings.py*. Kita menambahkan pengaturan untuk *default class fitur pagination*, *default limit per page* pada *pagination* dan juga *default class fitur filtering*.

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend']
}
```

Perubahan kode nya akan menjadi seperti di bawah ini. **newsproject/app/settings.py**

```
"""
Django settings for newsproject project.

Generated by 'django-admin startproject' using Django 3.1.4.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'bo46w@9act##xuwbzy3_2bs915*r&7okf&gzfgqy-u&liu()3r'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    # package untuk rest api
    'rest_framework',
    # package untuk fitur filtering pada rest api
    'django_filters',
    # package aplikasi kita
    'news.apps.NewsConfig',
    'django.contrib.admin',
]
```

```
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'newsproject.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'newsproject.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases
```

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'newsapp',  
        'USER': 'postgres',  
        'PASSWORD': '123456',  
        'HOST': '127.0.0.1',  
        'PORT': '5432',  
    }  
}  
  
# Password validation  
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators  
  
AUTH_PASSWORD_VALIDATORS = [  
    {  
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',  
    },  
]  
  
# Internationalization  
# https://docs.djangoproject.com/en/3.1/topics/i18n/  
  
LANGUAGE_CODE = 'en-us'  
  
TIME_ZONE = 'Asia/Jakarta'  
  
USE_I18N = True
```

```
USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'

REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend']
}
```

Selain mengubah pengaturan *setting* pada proyek *Django*, kita juga perlu sedikit mengubah di bagian *routing*. Buka file *newsproject/app/urls.py* dan lakukan perubahan seperti kode di bawah ini.

newsproject/app/urls.py

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    # routing untuk Rest API
    path('api/', include('news.urls')),
    # routing untuk Admin
    path('admin/', admin.site.urls),
]
```

API Endpoint Category

Sebelum masuk ke penulisan kode untuk *Rest API*, kita perlu memiliki gambaran *endpoint (url resource)* apa saja yang akan kita buat untuk modul *Category*. Pada proyek ini kita akan membuat dua *endpoint* untuk modul *Category*.

Nama	URL	Me-thod	Keterangan
Get List of Categories	/api/category	GET	<i>endpoint</i> untuk menampilkan semua data kategori.
Get Detail of Category	/api/category/:id	GET	<i>endpoint</i> untuk menampilkan detail data kategori berdasarkan kolom <i>id</i> .

Category Serializers

Serializers pada *djangorestframework* merupakan sebuah *class* yang digunakan untuk memudahkan *me-render* data *native Python* ke dalam format *JSON, XML* atau format lainnya yang biasanya dipake pada *Rest API*. Kita perlu membuat sebuah *file* baru dengan nama **serializers.py** di dalam *folder newsproject/news*. Buka *file* dan tuliskan kode di bawah ini untuk membuat *class* Serializers modul Category.

newsproject/news/serializers.py

```
# import class serializers
from rest_framework.serializers import ModelSerializer

# import Category model
from .models import Category

# Membuat class Serializers untuk API endpoint "Get List of Categories"
class CategoryListSerializer(ModelSerializer):
    class Meta:
        model = Category
        fields = ('id', 'name', )

# Membuat class Serializers untuk API endpoint "Get Detail of Category"
class CategoryDetailSerializer(ModelSerializer):
    class Meta:
        model = Category
        fields = ('id', 'name', 'created_at', )
```

Kita membuat *class* Serializers baru dari turunan *class* ModelSerializer. Kenapa kita memilih

class tersebut? Karena ModelSerializer memungkinkan kita secara otomatis membuat *class* Serializers dengan *fields/kolom* sesuai dengan *class Model* pada Django.

Referensi:

- <https://www.djangoproject.org/api-guide/serializers/>

Category Views

Views pada *djangorestframework* merupakan sebuah *class* yang bertugas memroses setiap *request client* melalui API endpoint. Selain membuat *serializers* tentunya kita juga perlu membuat Views untuk setiap API endpoint yang kita rancang. Kita akan membuat Views pada file **newsproject/news/views.py**. Buka *file* tersebut dan ubah menjadi seperti di bawah ini.

newsproject/news/views.py

```
# import class yang dibutuhkan untuk membuat view
from rest_framework.filters import SearchFilter, OrderingFilter
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.generics import ListAPIView, RetrieveAPIView

# import Category model dan class serializers untuk modul Category
from .models import Category
from .serializers import CategoryListSerializer, CategoryDetailSerializer

# Membuat View untuk API endpoint "Get All Categories"
# /api/category
class CategoryListView(ListAPIView):
    # mengeset class serializers
    serializer_class = CategoryListSerializer
    queryset = Category.objects.all()
    # Menambahkan fitur filtering, searching dan ordering
    filter_backends = (
        DjangoFilterBackend,
        SearchFilter,
        OrderingFilter,
    )
    # mengeset fields/kolom untuk fitur filtering
    filter_fields = ['name']
    # mengeset fields/kolom untuk fitur searching
```

```
search_fields = ['name']
# mengeset fields/kolom untuk fitur ordering
ordering_fields = ['name', 'created_at']
# mengeset fields/kolom default untuk fitur ordering
ordering = ['created_at']

# Membuat View untuk API endpoint "Get Detail of Category"
# /api/category/:id
class CategoryDetailView(RetrieveAPIView):
    # mengeset class serializers
    serializer_class = CategoryDetailSerializer
    queryset = Category.objects.all()
```

Jika kita perhatikan kode di atas, kita membuat dua **class Views** dari turunan **class Views** yang berbeda . Untuk **class CategoryListView** merupakan turunan dari **class ListAPIView** dan untuk **class CategoryDetailView** merupakan turunan dari **class RetrieveAPIView**. Berikut perbedaan dari dua **class** tersebut.

- **ListAPIView**: digunakan untuk *read-only endpoint* yang merepresentasikan banyak/*multiple* data dari **Model**. *class* ini mendukung penggunaan *method GET*.
- **RetrieveAPIView**: digunakan untuk *read-only endpoint* yang merepresentasikan satu/*single* data dari **Model**. *class* ini mendukung penggunaan *method GET*.

Referensi:

- <https://www.djangoproject.org/api-guide/generic-views/>

CategoryUrls

Terakhir kita perlu menambahkan **route** dengan membuat *file* baru di dalam *folder newsproject/news*. Tambahkan route utk **API endpoint Category** dengan menuliskan kode berikut.
newsproject/news/urls.py

```
from django.urls import path

# import semua class View untuk modul Category
from .views import CategoryListView, CategoryDetailView
```

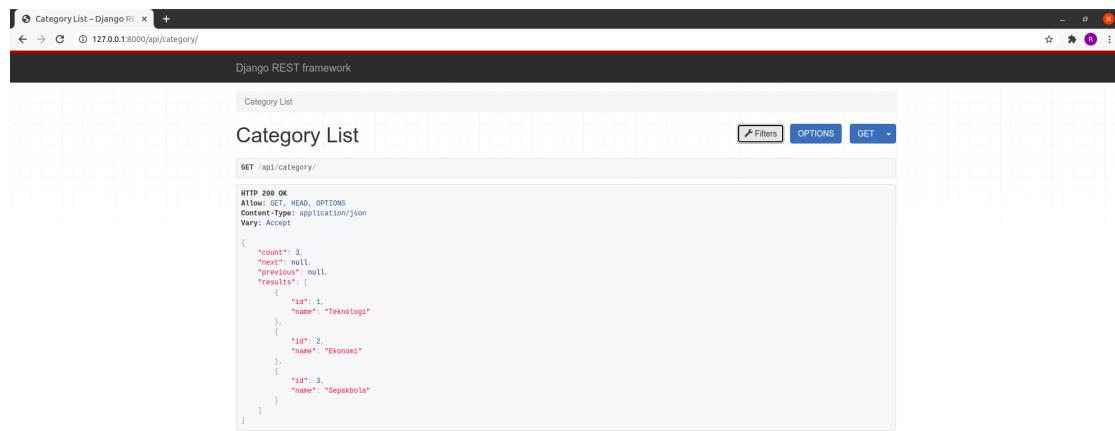
```
urlpatterns = [
    # membuat route untuk API endpoint Category
    path('category/', CategoryListView.as_view(), name='api-category-list'),
    path('category/<int:pk>', CategoryDetailView.as_view(), name='api-category-detail'),
]
```

Bisa dilihat pada kode di atas. kita menambahkan **route** dengan *method path* dengan mengirimkan *3 parameter*. *Parameter* pertama adalah format pathnya, yang kedua kita me-*render* *view* dari class **Views** dan terakhir nama route nya (bersifat *optional*). Untuk route **api-category-detail** kita menuliskan format pathnya seperti ini `'category/<int:pk>'` dimana **pk** merupakan parameter **primary key/id** dengan tipe data *integer* atau *numeric* dan itu bersifat **required** (harus diset parameternya).

Jalankan kembali proyek **Django** kita dengan mengetikkan perintah berikut di *terminal/command prompt* dari dalam folder **myprojects/newsproject**.

```
python3.9 manage.py runserver
```

Untuk mengetesnya kamu bisa menggunakan *tool* untuk mengetes **API** seperti **Postman**. Perlu kamu ketahui secara *default* **djangorestframework** juga menyediakan **playground** untuk mengetes **Rest API** di browser. Caranya buka url 127.0.0.1:8000/api/category dan akan muncul tampilan seperti ini.

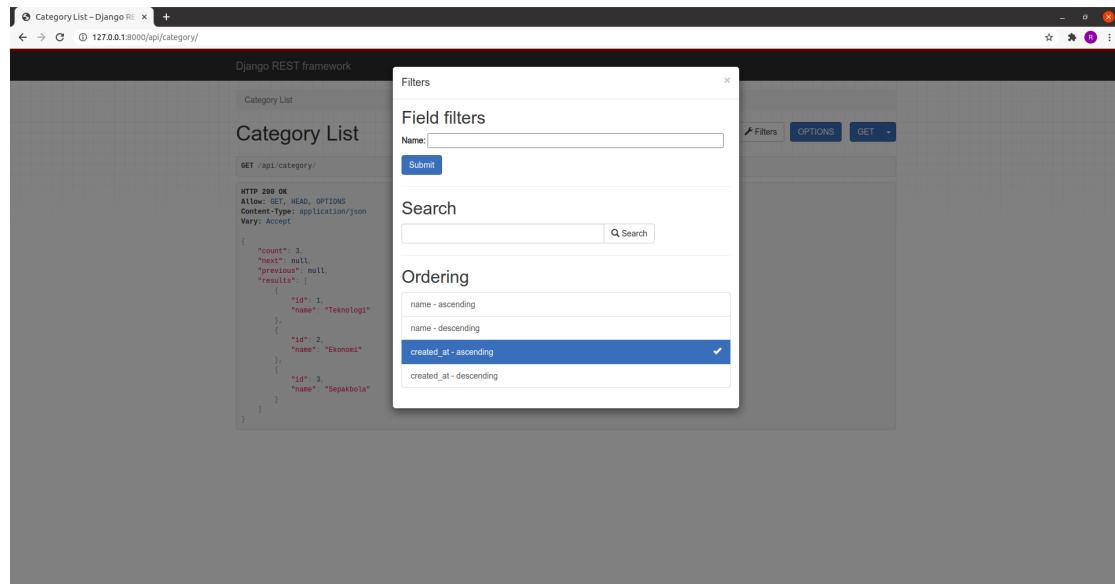


The screenshot shows a browser window titled "CategoryList-Django REST". The address bar indicates the URL is `127.0.0.1:8000/api/category/`. The page title is "Category List". Below the title, there are buttons for "Filters", "OPTIONS", and "GET". The main content area displays the JSON response for a GET request to `/api/category/`. The response is as follows:

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

{
    "count": 3,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 1,
            "name": "Teknologi"
        },
        {
            "id": 2,
            "name": "Ekonomi"
        },
        {
            "id": 3,
            "name": "Sepakbola"
        }
    ]
}
```

Gambar 15: Halaman Playground API Category List



Gambar 16: Halaman Playground API Category List

Klik pada tombol **Filters**, maka kamu bisa menggunakan fitur seperti filtering, searching dan juga ordering.

Dan untuk **API endpoint** Get Category detail bisa dengan mengakses url <http://127.0.0.1:8000/api/category/1>, maka akan tampil halaman seperti berikut.



Gambar 17: Halaman Playground API Category Detail

API Endpoint News

Pada **API Endpoint News/Berita**, kita akan membuat tiga API endpoint sebagai berikut.

Nama	URL	Me-thod	Keterangan
Get List of News	/api/news	GET	Endpoint untuk menampilkan semua data news/berita dengan status published .
Get Detail of News	/api/news/:id	GET	Endpoint untuk menampilkan detail data news/berita berdasarkan kolom id .
Create New Comment on News	/api/news/:id/comment	POST	Endpoint untuk menambahkan komentar pada news/berita berdasarkan kolom id .

Kita perlu memodifikasi **News Model** yang sudah kita buat terlebih dahulu supaya nantinya kita

bisa mengambil data *news/berita* yang hanya berstatus ***published***. Buka file **newsproject/news/models.py** dan tambahkan kode berikut sebelum *class News*.

```
# Membuat Custom Manager untuk NewsModel
class NewsManager(models.Manager):
    # Menambahkan method untuk memfilter news/berita yang hanya berstatus 'published'
    def is_published(self):
        return super().get_queryset().filter(status=News.NewsStatus.published)
```

Setelah itu tambahkan baris kode di bawah ini dalam *class News*.

```
# Menambahkan NewsManager
objects = NewsManager()
```

Sehingga keseluruhan kode pada file **models.py** akan menjadi seperti ini setelah kita menambahkan baris-baris kode di atas.

newsproject/news/models.py

```
from django.db import models
from django.contrib.auth.models import User

# Model untuk tabel category
class Category(models.Model):
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # set nama tabel
    class Meta:
        db_table = 'category'
        verbose_name_plural = "Category"

    def __str__(self):
        return self.name

# Membuat Custom Manager untuk NewsModel
```

```
class NewsManager(models.Manager):
    def is_published(self):
        return super().get_queryset().filter(status=News.NewsStatus.published)

# Model untuk tabel news
class News(models.Model):
    class NewsStatus(models.IntegerChoices):
        draft = 1
        published = 2

    title = models.CharField(max_length=255)
    cover = models.ImageField(upload_to='images')
    content = models.TextField()
    excerpt = models.TextField()
    status = models.IntegerField(choices=NewsStatus.choices)
    published_at = models.DateTimeField(null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # set relasi ke tabel user and category
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    categories = models.ManyToManyField(Category)

    # Menambahkan NewsManager
    objects = NewsManager()

    # set nama tabel
    class Meta:
        db_table = 'news'
        verbose_name_plural = "News"

    def __str__(self):
        return self.title

# Model untuk tabel comment
class Comment(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
```

```
content = models.TextField()  
created_at = models.DateTimeField(auto_now_add=True)  
updated_at = models.DateTimeField(auto_now=True)  
  
# set relasi ke tabel news  
news = models.ForeignKey(News, on_delete=models.CASCADE)  
  
# set nama tabel  
class Meta:  
    db_table = 'comment'
```

Pada kode di atas kita membuat *class* baru bernama **NewsManager** yang merupakan turunan dari *class* **models.Manager** dan menambahkannya ke dalam *class* **News**. *Class* **models.Manager** digunakan ketika kita ingin menambahkan beberapa *method* untuk menangani beberapa *query* yang cukup kompleks dan kita cukup memanggil *method* tersebut dengan nama yang lebih singkat. Seperti contoh di atas kita menambahkan *method* untuk memfilter data *news*/berita dengan status **published** sehingga nantinya jika kita ingin mengambil data *news*/berita dengan status **published** kita cukup menggunakan kode seperti ini.

```
# mengambil data news/berita dengan status published menggunakan models.Manager  
published_news = News.objects.is_published()  
# mengambil data news/berita dengan status published tanpa menggunakan models.Manager  
published_news = News.objects.filter(status=News.NewsStatus.published)
```

Referensi:

- <https://docs.djangoproject.com/en/3.1/topics/db/managers/>

News Serializers

Kita akan mempelajari penggunaan **Serializers** untuk modul atau tabel yang memiliki relasi ke tabel lain pada bagian **News Serializers** ini. Kalau kamu masih ingat untuk tabel **news** yang kita rancang itu berelasi ke beberapa tabel seperti tabel **auth_user**, **category** dan juga **comment**. Buka kembali *file* **newsproject/news/serializers.py**.

Tambahkan atau ubah kode berikut di bagian *import class* (bagian awal baris kode) untuk meng-import beberapa *class* yang dibutuhkan.

```
# import class serializers
from rest_framework.serializers import ModelSerializer

# import user model
from django.contrib.auth.models import User
# import class model lain
from .models import Category, News, Comment
```

Dan tambahkan kode berikut di bagian paling bawah dari file serializers.py.

```
# Membuat class UserSerializer yang akan berelasi ke NewsSerializer
class UserSerializer(ModelSerializer):
    class Meta:
        model = User
        fields = ('id', 'first_name', 'last_name', )

# Membuat class CommentListSerializer yang akan berelasi ke NewsSerializer
class CommentListSerializer(ModelSerializer):
    class Meta:
        model = Comment
        fields = ('id', 'name', 'content', 'created_at', )

# Membuat class CommentFormSerializer yang akan berelasi ke NewsSerializer
class CommentFormSerializer(ModelSerializer):
    class Meta:
        model = Comment
        fields = ('name', 'email', 'content', )

# Membuat class NewsListSerializer untuk API endpoint "Get List of News"
class NewsListSerializer(ModelSerializer):
    user = UserSerializer(read_only=True) # relasi ke modul/tabel user
    categories = CategoryListSerializer(many=True, read_only=True) # relasi ke modul/tabel category

    class Meta:
        model = News
        fields = ('id', 'title', 'excerpt', 'user', 'categories', 'published_at')
```

```
# Membuat class NewsDetailSerializer untuk API endpoint "Get Detail of News"
class NewsDetailSerializer(ModelSerializer):
    user = UserSerializer(read_only=True) # relasi ke modul/tabel user
    categories = CategoryListSerializer(many=True, read_only=True) # relasi ke modul/tabel category
    comments = CommentListSerializer(many=True, read_only=True, source='comment_set') # relasi ke
    ↵ modul/tabel comment

    class Meta:
        model = News
        fields = ('id', 'title', 'excerpt', 'content', 'cover', 'published_at', 'user', 'categories', 'comments', )
```

Kode di atas cukup panjang karena selain kita membuat *class Serializers* untuk **API Endpoint News** yaitu class **NewsListSerializer** dan **NewsDetailSerializer**, kita juga perlu membuat *class Serializers* lain untuk merelasikan *class Serializers* untuk modul *news*/berita dengan modul/tabel lainnya.

Secara keseluruhan baris kode pada *file serializers.py* akan menjadi seperti di bawah ini.

newsproject/news/serializers.py

```
# import class serializers
from rest_framework.serializers import ModelSerializer

# import user model
from django.contrib.auth.models import User
# import class model lain
from .models import Category, News, Comment

# Membuat class Serializers untuk API endpoint "Get List of Categories"
class CategoryListSerializer(ModelSerializer):
    class Meta:
        model = Category
        fields = ('id', 'name', )

# Membuat class Serializers untuk API endpoint "Get Detail of Category"
class CategoryDetailSerializer(ModelSerializer):
    class Meta:
```

```
model = Category
fields = ('id', 'name', 'created_at', )

# Membuat class UserSerializer yang akan berelasi ke NewsSerializer
class UserSerializer(ModelSerializer):
    class Meta:
        model = User
        fields = ('id', 'first_name', 'last_name', )

# Membuat class CommentListSerializer yang akan berelasi ke NewsSerializer
class CommentListSerializer(ModelSerializer):
    class Meta:
        model = Comment
        fields = ('id', 'name', 'content', 'created_at', )

# Membuat class CommentFormSerializer yang akan berelasi ke NewsSerializer
class CommentFormSerializer(ModelSerializer):
    class Meta:
        model = Comment
        fields = ('name', 'email', 'content',)

# Membuat class NewsListSerializer untuk API endpoint "Get List of News"
class NewsListSerializer(ModelSerializer):
    user = UserSerializer(read_only=True) # relasi ke modul/tabel user
    categories = CategoryListSerializer(many=True, read_only=True) # relasi ke modul/tabel category

    class Meta:
        model = News
        fields = ('id', 'title', 'excerpt', 'user', 'categories', 'published_at')

# Membuat class NewsDetailSerializer untuk API endpoint "Get Detail of News"
class NewsDetailSerializer(ModelSerializer):
    user = UserSerializer(read_only=True) # relasi ke modul/tabel user
    categories = CategoryListSerializer(many=True, read_only=True) # relasi ke modul/tabel category
    comments = CommentListSerializer(many=True, read_only=True, source='comment_set') # relasi ke
    ↵ modul/tabel comment
```

```
class Meta:  
    model = News  
    fields = ('id', 'title', 'excerpt', 'content', 'cover', 'published_at', 'user', 'categories', 'comments', )
```

News Views

Kita akan melanjutkan materi **API Endpoint News** dengan membuat **Views** untuk modul *news*/berita. Buka file **newsproject/news/views.py** dan pada bagian *import class* (bagian awal baris kode), ubah menjadi seperti baris-baris kode di bawah ini.

```
# import class yang dibutuhkan untuk membuat view  
from django.http import Http404  
from rest_framework.filters import SearchFilter, OrderingFilter  
from django_filters.rest_framework import DjangoFilterBackend  
from rest_framework.generics import ListAPIView, RetrieveAPIView, CreateAPIView  
# import class models dari aplikasi newsapp  
from .models import Category, News, Comment  
# import class serializers dari aplikasi newsapp  
from .serializers import CategoryListSerializer, CategoryDetailSerializer, NewsListSerializer,  
    NewsDetailSerializer, CommentFormSerializer  
# import class Filter untuk custom fitur filtering  
from .filters import NewsFilter
```

Dan tambahkan kode berikut di bagian paling bawah dari *file views.py*.

```
# Membuat View untuk API endpoint "Get List of News"  
# /api/news  
class NewsListView(ListAPIView):  
    serializer_class = NewsListSerializer  
    # mengambil data news/berita yang hanya berstatus published  
    queryset = News.objects.is_published()  
  
    filter_backends = (  
        DjangoFilterBackend,  
        SearchFilter,  
        OrderingFilter,  
    )
```

```
# mengeset fields/kolom untuk fitur filtering dengan class NewsFilter (custom filter)
filterset_class = NewsFilter
# mengeset fields/kolom untuk fitur searching
search_fields = ['title']
# mengeset fields/kolom untuk fitur ordering
ordering_fields = ['title', 'created_at']
# mengeset fields/kolom default untuk fitur ordering
ordering = ['created_at']

# Membuat View untuk API endpoint "Get Detail of News"
# /api/news/:id
class NewsDetailView(RetrieveAPIView):
    serializer_class = NewsDetailSerializer
    # mengambil data news/berita yang hanya berstatus published
    queryset = News.objects.is_published()

# Membuat View untuk API endpoint "Create New Comment on News"
# /api/news/:id/comment
class NewsCreateCommentView(CreateAPIView):
    serializer_class = CommentFormSerializer
    queryset = Comment.objects.all()
    # meng-override method perform_create untuk mengambil news_id dari parameter url API endpoint
    # Kemudian disimpan pada kolom news_id di tabel Comment pada saat kita melakukan requests endpoint
    # ini.
    def perform_create(self, serializer):
        news_id = self.kwargs['pk']
        try:
            news = News.objects.get(pk=news_id)
            serializer.save(news_id=news.id)
        except News.DoesNotExist:
            raise Http404
```

Dari baris-baris kode di atas, yang kita tambahkan adalah untuk mengimport beberapa *class* atau modul untuk membuat **Views** pada modul *news/berita*. Kemudian kita membuat tiga Views masing-masing untuk API endpoint **Get List of News (NewsListView)**, **Get Detail of News (NewsDetailView)** dan terakhir untuk API endpoint **Create New Comment on News (NewsCreateCommentView)**.

Secara keseluruhan baris kode pada *file views.py* akan menjadi seperti di bawah ini.

newsproject/news/views.py

```
# import class yang dibutuhkan untuk membuat view
from django.http import Http404
from rest_framework.filters import SearchFilter, OrderingFilter
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.generics import ListAPIView, RetrieveAPIView, CreateAPIView
# import class models dari aplikasi newsapp
from .models import Category, News, Comment
# import class serializers dari aplikasi newsapp
from .serializers import CategoryListSerializer, CategoryDetailSerializer, NewsListSerializer,
    NewsDetailSerializer, CommentFormSerializer
# import class Filter untuk custom fitur filtering
from .filters import NewsFilter

# Membuat View untuk API endpoint "Get All Categories"
# /api/category
class CategoryListView(ListAPIView):
    # mengeset class serializers
    serializer_class = CategoryListSerializer
    queryset = Category.objects.all()
    # Menambahkan fitur filtering, searching dan ordering
    filter_backends = (
        DjangoFilterBackend,
        SearchFilter,
        OrderingFilter,
    )
    # mengeset fields/kolom untuk fitur filtering
    filter_fields = ['name']
    # mengeset fields/kolom untuk fitur searching
    search_fields = ['name']
    # mengeset fields/kolom untuk fitur ordering
    ordering_fields = ['name', 'created_at']
    # mengeset fields/kolom default untuk fitur ordering
    ordering = ['created_at']

# Membuat View untuk API endpoint "Get Detail of Category"
# /api/category/:id
```

```
class CategoryDetailView(RetrieveAPIView):
    # mengeset class serializers
    serializer_class = CategoryDetailSerializer
    queryset = Category.objects.all()

    # Membuat View untuk API endpoint "Get List of News"
    # /api/news
    class NewsListView(ListAPIView):
        serializer_class = NewsListSerializer
        # mengambil data news/berita yang hanya berstatus published
        queryset = News.objects.is_published()

        filter_backends = (
            DjangoFilterBackend,
            SearchFilter,
            OrderingFilter,
        )
        # mengeset fields/kolom untuk fitur filtering dengan class NewsFilter (custom filter)
        filterset_class = NewsFilter
        # mengeset fields/kolom untuk fitur searching
        search_fields = ['title']
        # mengeset fields/kolom untuk fitur ordering
        ordering_fields = ['title', 'created_at']
        # mengeset fields/kolom default untuk fitur ordering
        ordering = ['created_at']

    # Membuat View untuk API endpoint "Get Detail of News"
    # /api/news/:id
    class NewsDetailView(RetrieveAPIView):
        serializer_class = NewsDetailSerializer
        # mengambil data news/berita yang hanya berstatus published
        queryset = News.objects.is_published()

    # Membuat View untuk API endpoint "Create New Comment on News"
    # /api/news/:id/comment
    class NewsCreateCommentView(CreateAPIView):
        serializer_class = CommentFormSerializer
```

```
queryset = Comment.objects.all()
# meng-override method perform_create untuk mengambil news_id dari parameter url API endpoint
# Kemudian disimpan pada kolom news_id di tabel Comment pada saat kita melakukan requests endpoint
# ini.
def perform_create(self, serializer):
    news_id = self.kwargs['pk']
    try:
        news = News.objects.get(pk=news_id)
        serializer.save(news_id=news.id)
    except News.DoesNotExist:
        raise Http404
```

Dari baris kode **views.py** di atas, perhatikan pada bagian ini.

```
# import class Filter untuk custom fitur filtering
from .filters import NewsFilter
```

Class NewsFilter ini digunakan untuk mengcustomisasi fitur *filtering* yang ada pada *package django-filter*. Kita mengimport *class* NewsFilter dan kalau diingat kita belum membuatnya. Untuk itu kita perlu membuatnya sekarang. Buat sebuah *file* baru dengan nama **filters.py** di dalam *folder* **newsproject/news**. Kemudian buka *file* tersebut dan tuliskan kode di bawah ini.

newsproject/news/filters.py

```
from django_filters import rest_framework as filters

from .models import News


class NewsFilter(filters.FilterSet):
    # custom filter news berdasarkan nama kategori
    categories = filters.CharFilter(field_name="categories__name", lookup_expr='exact')
    # custom filter tanggal publish news/berita
    # berdasarkan range tanggal dari parameter published_after & published_before
    published = filters.DateFromToRangeFilter(field_name="published_at")

    class Meta:
        model = News
        fields = ['title', 'categories', 'published']
```

Referensi:

- <https://django-filter.readthedocs.io/en/latest/ref/filterset.html>

News Urls

Terakhir untuk menyelesaikan pembuatan **API Endpoint News**, kita akan menambahkan *routing* untuk *endpoint-endpoint* tersebut pada file **urls.py**.

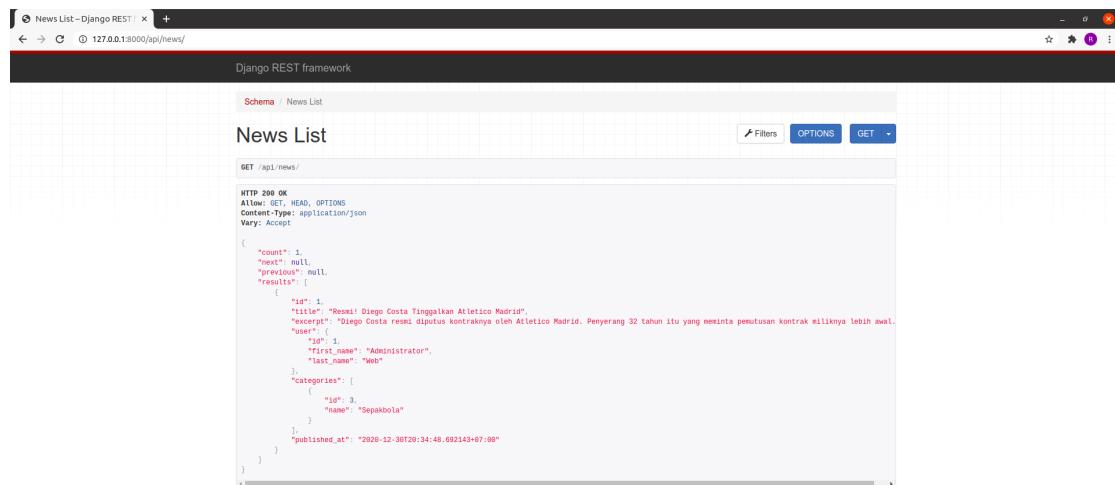
newsproject/news/urls.py

```
from django.urls import path

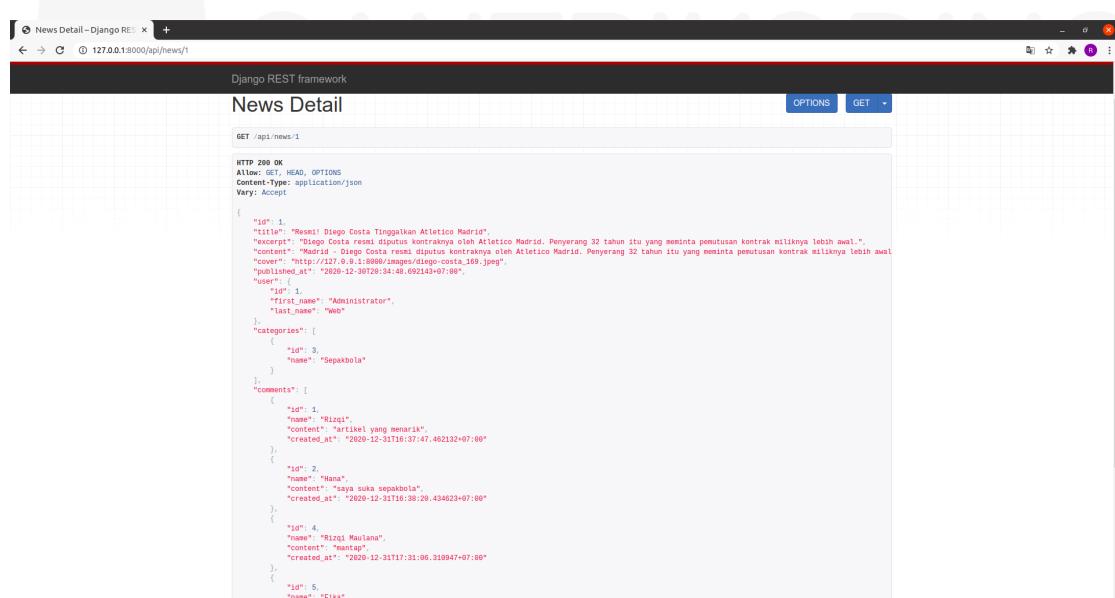
# import semua class View untuk modul Category dan News
from .views import CategoryListView, CategoryDetailView, NewsListView, NewsDetailView,
    NewsCreateCommentView

urlpatterns = [
    # membuat route untuk API endpoint Category
    path('category/', CategoryListView.as_view(), name='api-category-list'),
    path('category/<int:pk>', CategoryDetailView.as_view(), name='api-category-detail'),
    # membuat route untuk API endpoint News
    path('news/', NewsListView.as_view(), name='api-news-list'),
    path('news/<int:pk>/comment', NewsCreateCommentView.as_view(), name='api-news-create-comment'),
    path('news/<int:pk>', NewsDetailView.as_view(), name='api-news-detail'),
]
```

Silahkan buka url <http://127.0.0.1:8000/api/news> pada browser untuk melihat API endpoint **Get List of News**.

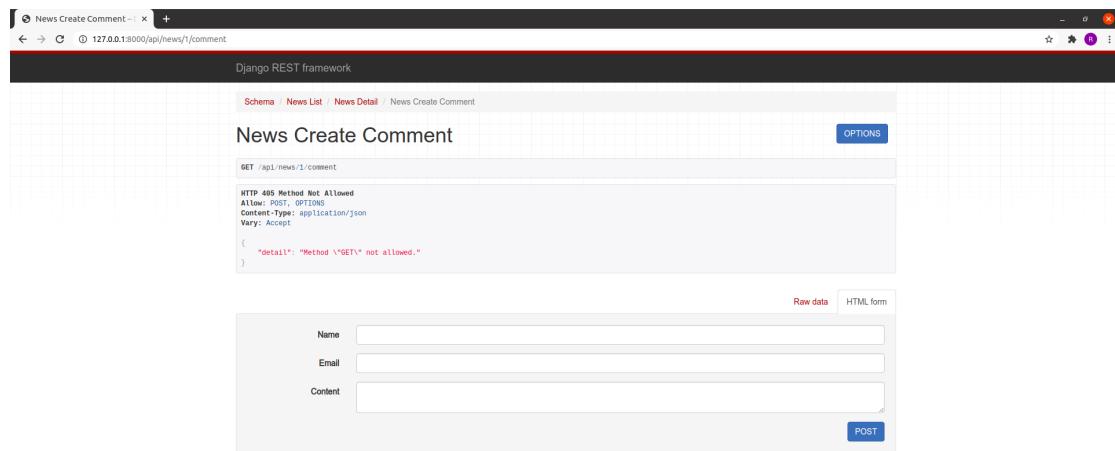
**Gambar 18:** Halaman Playground API News List

Selanjutnya untuk mengecek API endpoint **Get Detail of News** buka url <http://127.0.0.1:8000/api/news/1>

**Gambar 19:** Halaman Playground API News Detail

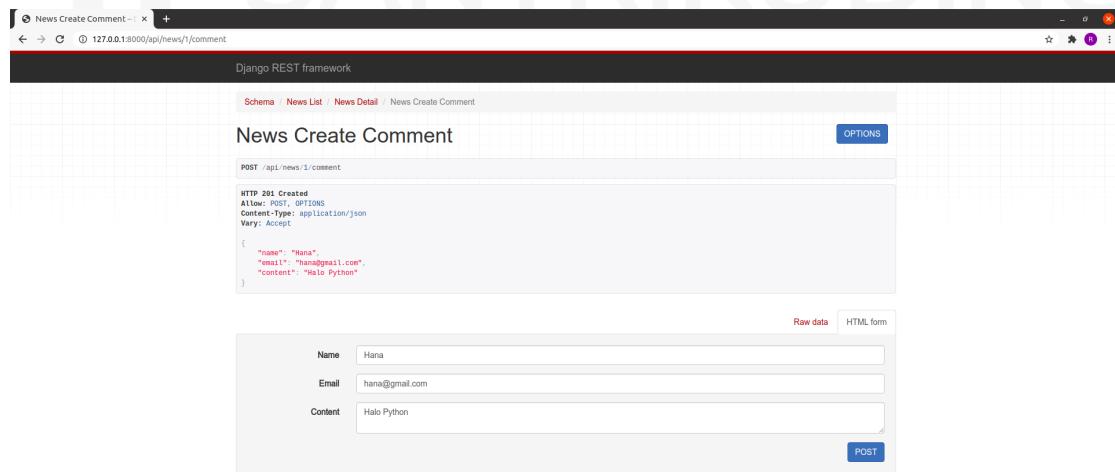
Terakhir untuk mengecek API endpoint **Create New Comment on News** buka url <http://127.0.0.1:8000/api/news/1/comments/>

8000/api/news/1/comment



Gambar 20: Halaman Playground API News Create Comment

Akan muncul Halaman dengan *form* untuk menambahkan komentar, coba isi semuanya kemudian klik tombol **POST**. jika sukses akan muncul halaman seperti ini.



Gambar 21: Halaman Playground API News Create Comment

Authentication

Sejauh ini kita sudah menyelesaikan materi pada bab **Rest API** untuk modul **Category** dan **News**. Tetapi Rest API yang kita bangun masih bersifat publik artinya semua orang nantinya bisa mengakses Rest API kita tanpa terkecuali jika kita sudah mendeploy proyek kita ke *server*. Pada materi ini kita akan menambahkan otentikasi (**Authentication**) untuk Rest API dari proyek kita sehingga hanya *user* yang mempunyai akses yang bisa mengakses Rest API tersebut.

Ada beberapa jenis otentikasi yang bisa digunakan pada **djangorestframework** dan yang akan kita pakai adalah otentikasi **TokenAuthentication**. Skema otentikasi ini menggunakan skema otentikasi **HTTP** berbasis *token* sederhana yang nanti implementasinya akan kita pelajari bersama. Untuk menggunakan TokenAuthentication kita perlu menambahkan atau mengubah konfigurasi *setting* proyek kita. Buka file **newsproject/app/settings.py**.

Tambahkan kode berikut pada bagian **INSTALLED_APPS**:

```
INSTALLED_APPS = [
    ...
    # Autentikasi Rest API dengan Token
    'rest_framework.authtoken'
]
```

Kemudian pada bagian **REST_FRAMEWORK** tambahkan kode berikut.

```
REST_FRAMEWORK = {
    ...
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
}
```

Sehingga nantinya kode pada file **settings.py** akan menjadi seperti ini.

newsproject/app/settings.py

```
"""
Django settings for newsproject project.

Generated by 'django-admin startproject' using Django 3.1.4.

```

```
For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'bo46w@9act##xuwbzy3_2bs915*r&7okf&gzfgqy-u&liu()3r'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    # package untuk rest api
    'rest_framework',
    # package untuk otentikasi rest api
    'rest_framework.authtoken',
    # package untuk fitur filtering pada rest api
    'django_filters',
    # package aplikasi kita
    'news.apps.NewsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
```

```
'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'newsproject.urls'

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]

WSGI_APPLICATION = 'newsproject.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'newsapp',
```

```
'USER': 'postgres',
'PASSWORD': '123456',
'HOST': '127.0.0.1',
'PORT': '5432',
}

}

# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-passwordValidators

AUTH_PASSWORD_VALIDATORS = [
{
    'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
},
]

# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Asia/Jakarta'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/
```

```
STATIC_URL = '/static/'

REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
}
```

Kita perlu menjalankan *migration* setelah melakukan penambahan pada *setting* konfigurasi untuk **TokenAuthentication**. Buka kembali *terminal/command prompt*. Jalankan perintah berikut.

```
python3.9 manage.py migrate
```

Selanjutnya kita perlu menambahkan *class TokenAuthentication* dan **IsAuthenticated** pada semua *class Views Category* dan **News** untuk memproteksi setiap **API endpoint** yang ada. Buka file **newsproject/news/views.py** dan tambahkan kode berikut di bagian *import class* (bagian atas).

```
# import class untuk otentikasi rest api
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated
```

Dan tambahkan dua *class* tersebut pada setiap *class Views* yang ada. seperti ini contohnya.

```
class CategoryListView(ListAPIView):
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]
    ...
```

Lakukan seperti itu juga untuk semua *class Views* yang ada, sehingga perubahan kodennya akan seperti ini.

newsproject/news/views.py

```
# import class yang dibutuhkan untuk membuat view
from django.http import Http404
from rest_framework.filters import SearchFilter, OrderingFilter
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.generics import ListAPIView, RetrieveAPIView, CreateAPIView
# import class untuk otentikasi rest api
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated

# import class models dari aplikasi newsapp
from .models import Category, News, Comment
# import class serializers dari aplikasi newsapp
from .serializers import CategoryListSerializer, CategoryDetailSerializer, NewsListSerializer,
    NewsDetailSerializer, CommentFormSerializer
# import class Filter untuk custom fitur filtering
from .filters import NewsFilter

# Membuat View untuk API endpoint "Get All Categories"
# /api/category
class CategoryListView(ListAPIView):
    # mengeset class serializers
    serializer_class = CategoryListSerializer
    queryset = Category.objects.all()
    # Menambahkan fitur filtering, searching dan ordering
    filter_backends = (
        DjangoFilterBackend,
        SearchFilter,
        OrderingFilter,
    )
    # mengeset fields/kolom untuk fitur filtering
    filter_fields = ['name']
    # mengeset fields/kolom untuk fitur searching
    search_fields = ['name']
    # mengeset fields/kolom untuk fitur ordering
    ordering_fields = ['name', 'created_at']
    # mengeset fields/kolom default untuk fitur ordering
    ordering = ['created_at']
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]
```

```
# Membuat View untuk API endpoint "Get Detail of Category"
# /api/category/:id
class CategoryDetailView(RetrieveAPIView):
    # mengeset class serializers
    serializer_class = CategoryDetailSerializer
    queryset = Category.objects.all()
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]

# Membuat View untuk API endpoint "Get List of News"
# /api/news
class NewsListView(ListAPIView):
    serializer_class = NewsListSerializer
    # mengambil data news/berita yang hanya berstatus published
    queryset = News.objects.is_published()

    filter_backends = (
        DjangoFilterBackend,
        SearchFilter,
        OrderingFilter,
    )
    # mengeset fields/kolom untuk fitur filtering dengan class NewsFilter (custom filter)
    filterset_class = NewsFilter
    # mengeset fields/kolom untuk fitur searching
    search_fields = ['title']
    # mengeset fields/kolom untuk fitur ordering
    ordering_fields = ['title', 'created_at']
    # mengeset fields/kolom default untuk fitur ordering
    ordering = ['created_at']
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]

# Membuat View untuk API endpoint "Get Detail of News"
# /api/news/:id
```

```
class NewsDetailView(RetrieveAPIView):
    serializer_class = NewsDetailSerializer
    # mengambil data news/berita yang hanya berstatus published
    queryset = News.objects.is_published()
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]

    # Membuat View untuk API endpoint "Create New Comment on News"
    # /api/news/:id/comment
class NewsCreateCommentView(CreateAPIView):
    serializer_class = CommentFormSerializer
    queryset = Comment.objects.all()
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]
    # meng-override method perform_create untuk mengambil news_id dari parameter url API endpoint
    # Kemudian disimpan pada kolom news_id di tabel Comment pada saat kita melakukan requests endpoint
    # ini.
    def perform_create(self, serializer):
        news_id = self.kwargs['pk']
        try:
            news = News.objects.get(pk=news_id)
            serializer.save(news_id=news.id)
        except News.DoesNotExist:
            raise Http404
```

Simpan kembali perubahan pada *file views.py* kemudian coba buka url <http://127.0.0.1:8000/api/category/> maka akan muncul halaman dengan pesan peringatan yang intinya memberi tahu bahwa Otentikasi tidak diberikan dan itu artinya kita sudah berhasil menambahkan otentikasi pada **Rest API** proyek kita.

```
{
    "detail": "Authentication credentials were not provided."
}
```

Untuk mendapatkan *token* kita bisa mengetikkan perintah berikut dari *terminal/command prompt*.

```
python3.9 manage.py drf_create_token <username>
```

Jika berhasil, maka akan muncul *token* pada *terminal/command prompt* seperti berikut.

```
env) sakukode@sakukode:~/newsproject$ python3.9 manage.py drf_create_token admin
Generated token da576204d0cc340a12a245055c305eb406235257 for user admin
```

Tentunya cara di atas bukan cara terbaik untuk menggenerate *token*, karena itu harus dilakukan manual dengan mengetikkan perintah di atas satu per satu untuk setiap user. Ada cara lain agar kita bisa menggenerate *token* secara otomatis ketika kita membuat *user* baru dari halaman **Admin**. Caranya dengan menambahkan baris kode berikut pada **file newsproject/news/models.py**. Tambahkan kode berikut pada bagian atas.

```
# import class untuk menggenerate token user
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
```

Selanjutnya tambahkan juga kode berikut pada bagian bawah *file*.

```
# Method untuk menggenerate token secara otomatis untuk Otentikasi Rest API ketika user dibuat
@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)
```

Keseluruhan kode akan berubah seperti di bawah ini.

newsproject/news/models.py

```
from django.db import models
from django.contrib.auth.models import User
# import class untuk menggenerate token user
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

# Model untuk tabel category
class Category(models.Model):
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # set nama tabel
    class Meta:
        db_table = 'category'
        verbose_name_plural = "Category"

    def __str__(self):
        return self.name

# Membuat Custom Manager untuk NewsModel
class NewsManager(models.Manager):
    def is_published(self):
        return super().get_queryset().filter(status=News.NewsStatus.published)

# Model untuk tabel news
class News(models.Model):
    class NewsStatus(models.IntegerChoices):
        draft = 1
        published = 2

        title = models.CharField(max_length=255)
        cover = models.ImageField(upload_to='images')
        content = models.TextField()
        excerpt = models.TextField()
        status = models.IntegerField(choices=NewsStatus.choices)
        published_at = models.DateTimeField(null=True)
        created_at = models.DateTimeField(auto_now_add=True)
        updated_at = models.DateTimeField(auto_now=True)

    # set relasi ke tabel user and category
    user = models.ForeignKey(User, on_delete=models.CASCADE)
```

```
categories = models.ManyToManyField(Category)

# Menambahkan NewsManager
objects = NewsManager()

# set nama tabel
class Meta:
    db_table = 'news'
    verbose_name_plural = "News"

def __str__(self):
    return self.title

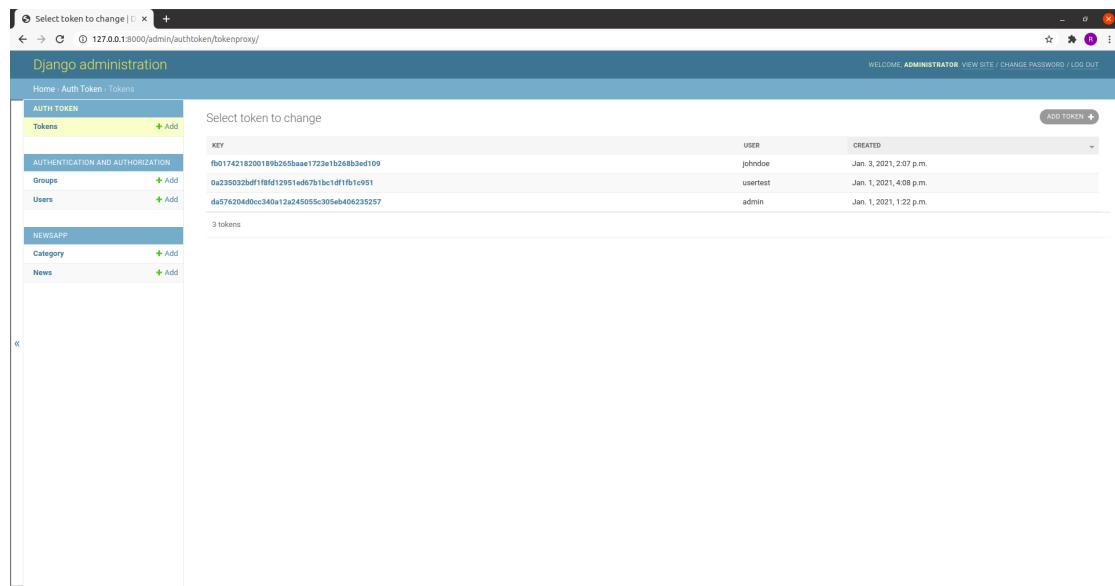
# Model untuk tabel comment
class Comment(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # set relasi ke tabel news
    news = models.ForeignKey(News, on_delete=models.CASCADE)

    # set nama tabel
    class Meta:
        db_table = 'comment'

# Method untuk menggenerate token secara otomatis untuk Otentikasi Rest API ketika user dibuat
@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)
```

Sekarang kita bisa mencoba untuk membuat *user* baru dari halaman **Admin**. Setelah itu masuk ke menu **Token** di bawah grup menu **AUTH TOKEN**. maka kamu akan melihat daftar *token* untuk *user* yang terdaftar.



Gambar 22: Halaman Token Menu Dashboard Admin

Kita bisa melakukan tes sederhana untuk mengakses **Rest API** menggunakan *token* dengan menjalankan perintah berikut dari *terminal/command prompt*.

```
curl -X GET http://127.0.0.1:8000/api/category/ -H 'Authorization: Token
↪ fb0174218200189b265baae1723e1b268b3ed109'
```

Dan jika sukses akan seperti ini *outputnya*.

```
sakukode@sakukode:~$ curl -X GET http://127.0.0.1:8000/api/category/ -H 'Authorization: Token
↪ fb0174218200189b265baae1723e1b268b3ed109'
>{"count":3,"next":null,"previous":null,"results":[{"id":1,"name":"Teknologi"}, {"id":2,"name":"Ekonomi"}, {"id":3,"name":"Sepakbola"}]}
```

Untuk *testing* yang lebih lengkap akan kita bahas secara mendalam pada materi selanjutnya yaitu **Testing API**.

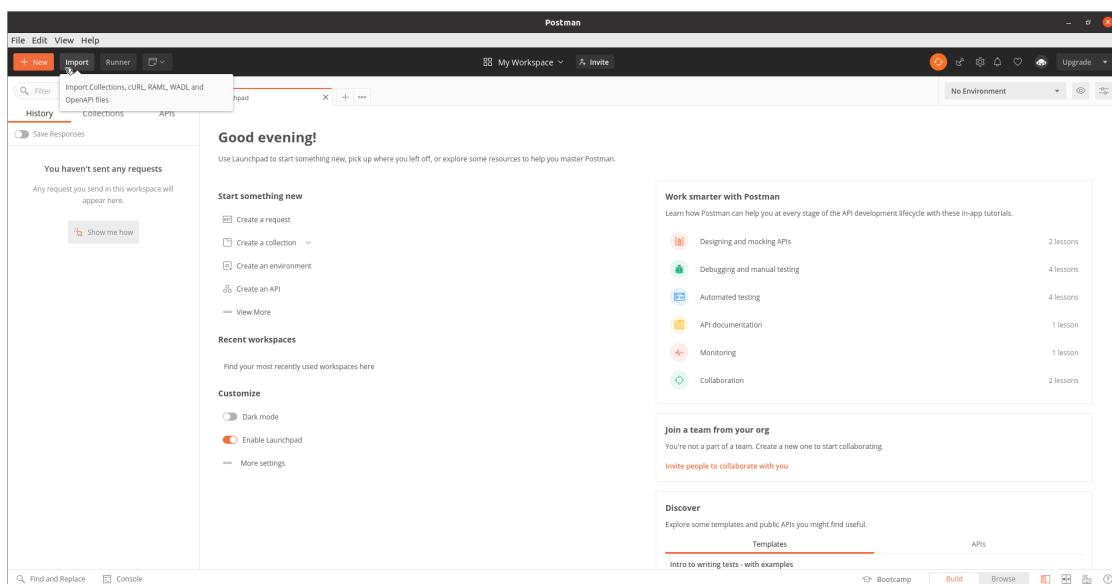
Testing API

Ada 2 cara yang akan kita pelajari pada materi **Testing API**, yang pertama melakukan pengujian atau *testing* secara *manual* menggunakan aplikasi **Postman** dan yang kedua membuat **unit testing** pada proyek **Django** kita dan bisa menjalankannya secara otomatis menggunakan *scripts*.

Postman

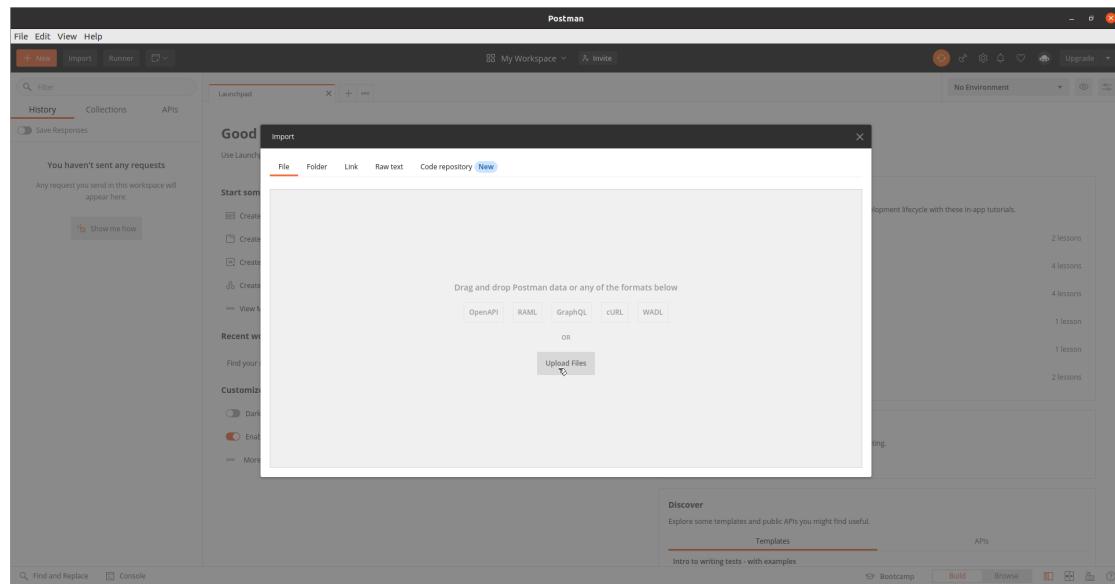
Untuk melakukan *testing* secara *manual* kita akan menggunakan aplikasi **Postman** yang sudah kita instal. Tapi sebelum itu kita perlu mendownload dua file berikut: **santrikoding_news_app_api_django.postman_collection.json** dan **santrikoding_news_app_api_django.postman_environment.json**. Dua file tersebut sudah penulis siapkan untuk mempermudah kita melakukan *testing* dengan aplikasi Postman. Tahap pertama kita perlu mengimport file **santrikoding_news_app_api_django.postman_collection.json**. File tersebut berisi kumpulan **API endpoint** dari proyek kita yang disebut **Collection**.

Buka aplikasi **Postman**, dan klik tombol **Import** yang ada pada bagian kiri atas.



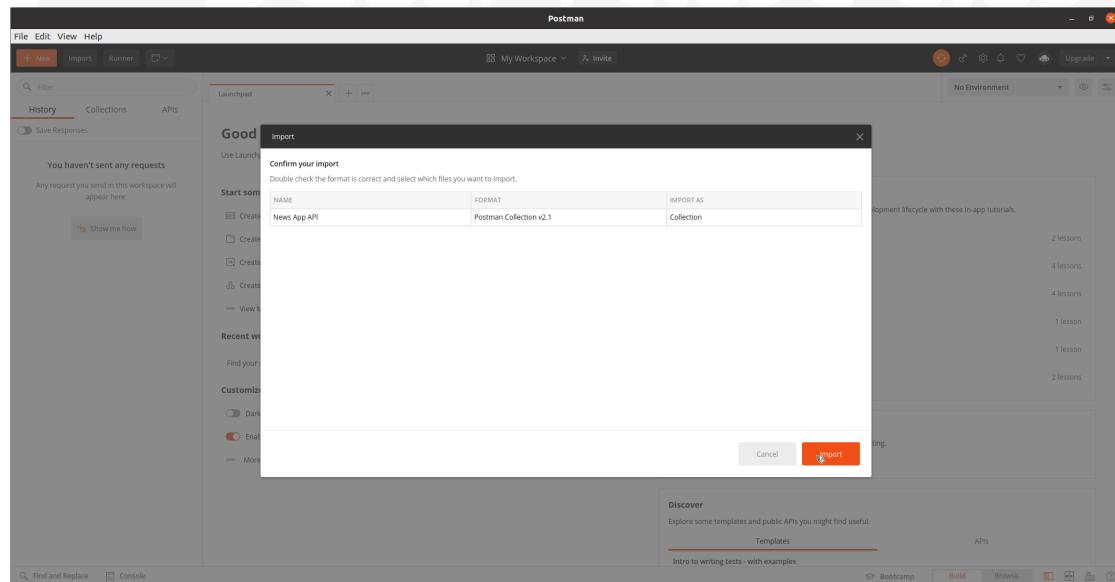
Gambar 23: Import Collection Pada Postman

Klik tombol **Upload Files** dan pilih file **santrikoding_news_app_api_django.collection.json** dari folder tempat kamu mendownloadnya.



Gambar 24: Import Collection Pada Postman

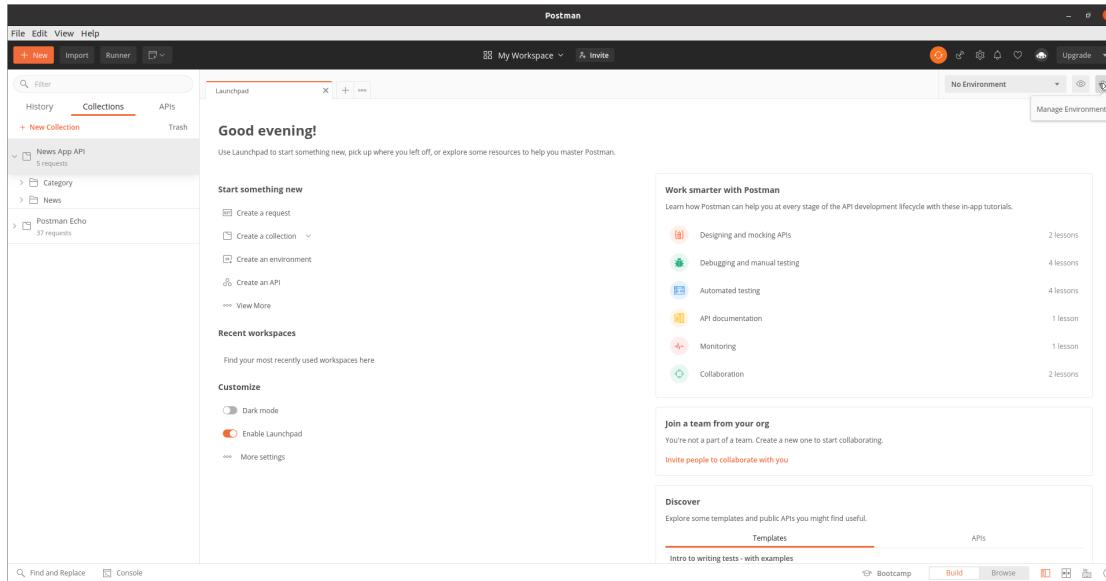
Setelah memilih **file**, akan muncul nama **Collection** dari **file** tersebut. Selesaikan tahap ini dengan mengklik tombol **Import**.



Gambar 25: Import Collection Pada Postman

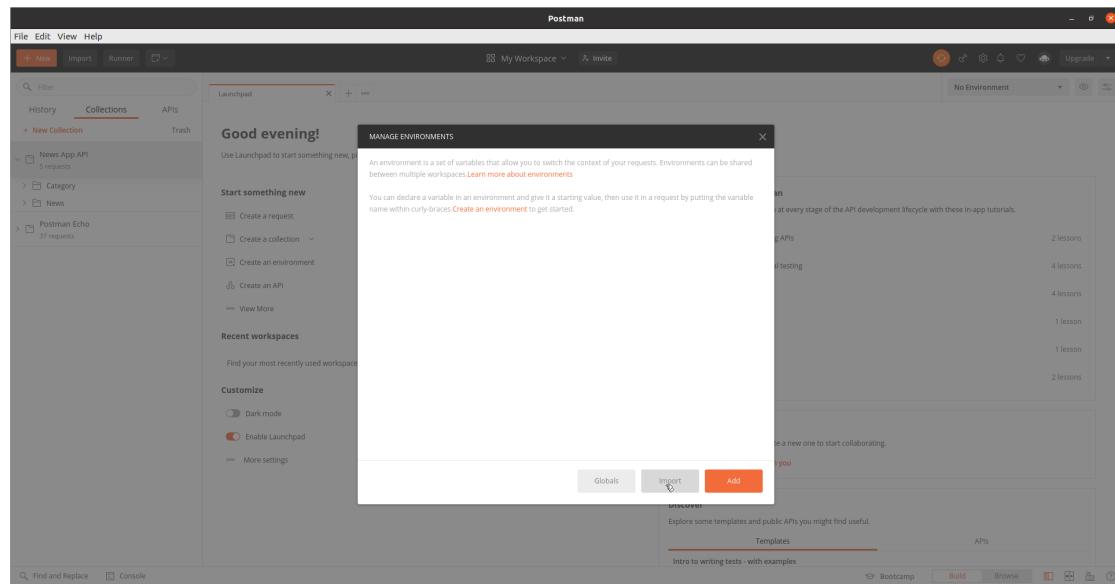
Selanjutnya kita perlu mengimport **Environment** yang ada pada **file** **santrikoding_ne-**

ws_app_api_django.postman_environment.json. File Environment ini berisi variabel **base url** Rest API dan juga **token** untuk otentikasi. Langsung saja silahkan klik tombol di bagian kanan atas seperti gambar di bawah ini.



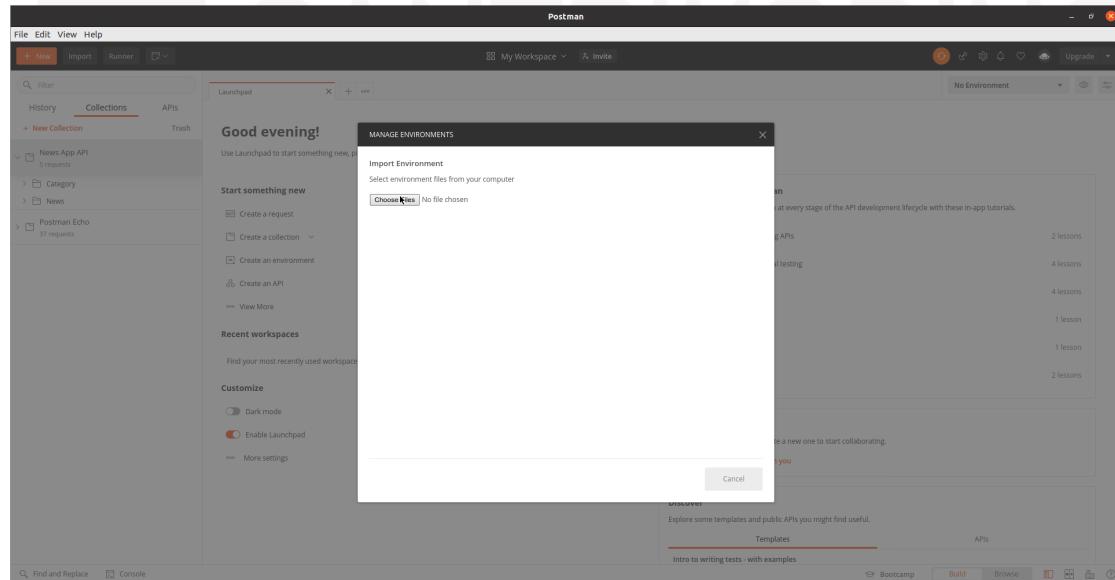
Gambar 26: Import Environment Pada Postman

Klik tombol **Import**.



Gambar 27: Import Environment Pada Postman

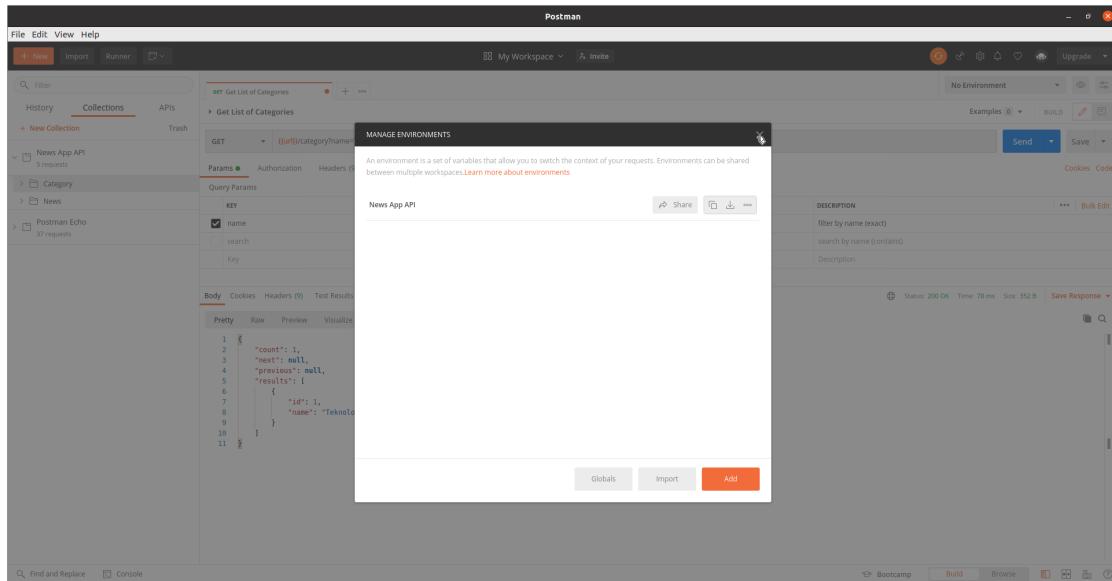
Dan silahkan pilih file **santrikoding_news_app_api_django.postman_environment.json** dari folder tempat kamu mendownloadnya.



Gambar 28: Import Environment Pada Postman

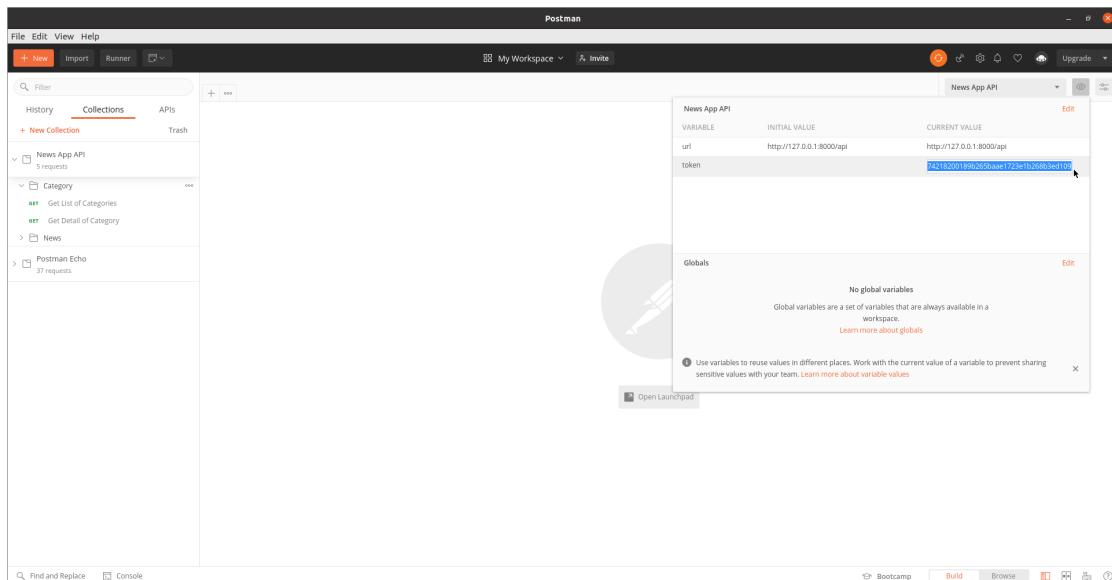
Akan muncul tampilan seperti di bawah ini jika kita berhasil mengimport **Environment**. Setelah

itu klik tombol **close (x)**.



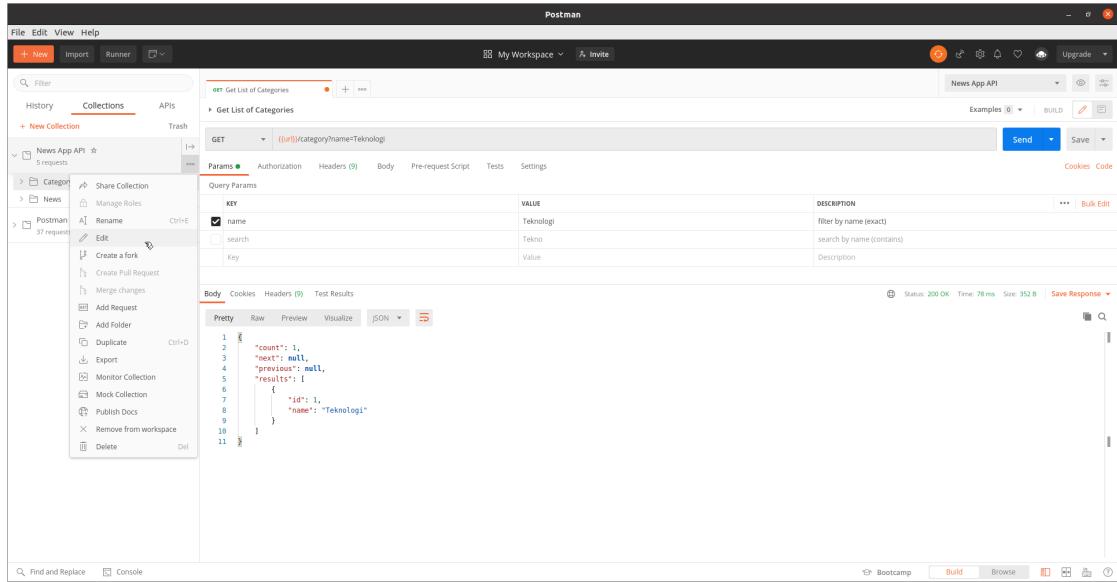
Gambar 29: Import Environment Pada Postman

Selanjutnya dari **select box** di bagian kanan atas, pilih opsi **News App API** dan klik tombol icon mata yang ada di sampingnya. Edit bagian **value token** dan isikan dengan token dari **user** yang sudah kita buat.



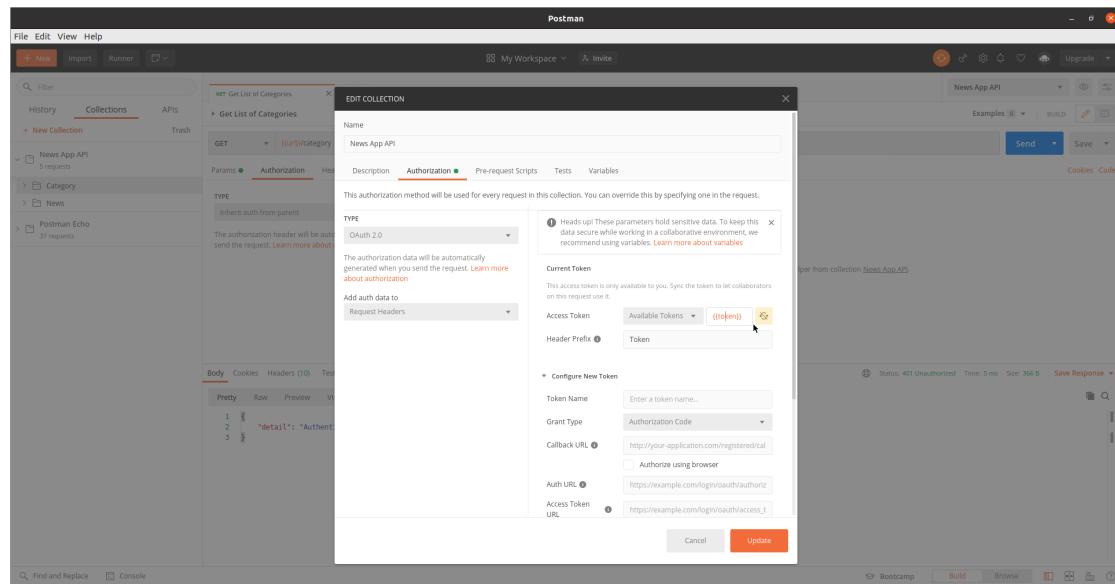
Gambar 30: Import Environment Pada Postman

Kita juga perlu mengedit **Collection** yang sudah kita *import*. Klik *icon titik tiga* yang ada di bagian kanan nama Collection dan pilih **Edit**.



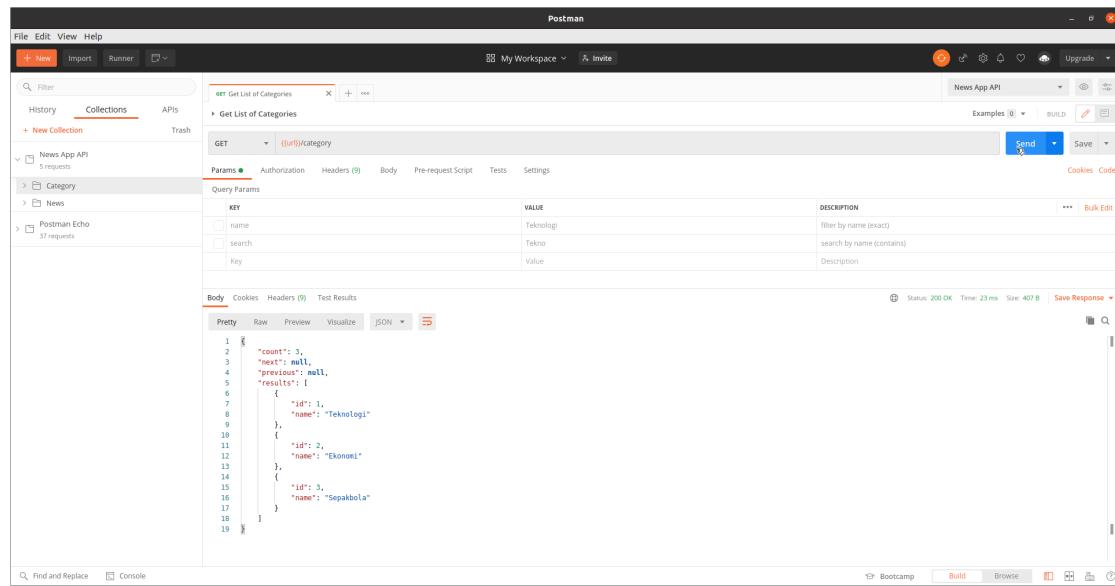
Gambar 31: Edit Collection Pada Postman

Pada bagian *input Access Token* isikan *valuanya* dengan nilai `{{token}}` kemudian klik **Update** untuk menyimpannya.

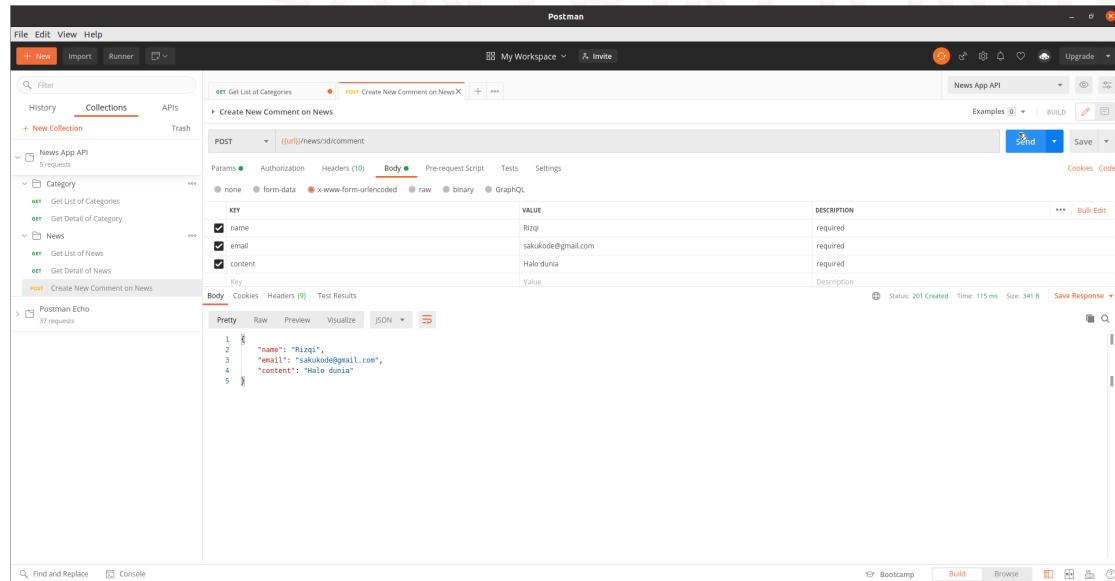


Gambar 32: Edit Collection Pada Postman

Sekarang saatnya untuk mencoba melakukan *testing* pada **Rest API**. Kalau dilihat akan ada dua *folder* di dalam **Collection News App API** yaitu **Category** dan **News**. Buka *folder* Category dan pilih **Get List of Categories** untuk melakukan *testing* API endpoint Get List of Categories. Silakan klik tombol **Send** dan jika *request* Rest API sukses hasilnya akan seperti gambar di bawah ini.

**Gambar 33:** Testing API Category dengan Postman

Kita akan mencoba **API endpoint** yang lain, buka pada *folder News* dan pilih **Create New Comment on News**. Kemudian klik pada *tab Body*, dan isikan semua *value* pada *form* yang ada. Setelah itu coba klik tombol **Send**. Jika sukses hasilnya akan seperti gambar di bawah ini.

**Gambar 34:** Testing API News dengan Postman

Unit Testing

Selain melakukan *testing* secara *manual*, kita akan melakukan *testing* secara otomatis yaitu dengan membuat **Unit test** pada **Django**. Kita akan membuat Unit test untuk menguji semua **API endpoint** yang sudah kita buat. Dan nantinya Unit test ini bisa kita jalankan melalui perintah di *terminal/command prompt*. Silahkan buka file **newsproject/news/tests.py** dan tuliskan kode di bawah ini.

newsproject/news/tests.py

```
# import class atau module yang digunakan untuk Testing
from django.contrib.auth.models import User
from django.core.files.uploadedfile import SimpleUploadedFile
from django.test import TestCase
from django.urls import reverse
from rest_framework import status
from rest_framework.authtoken.models import Token
from rest_framework.test import APIClient

# import semua models untuk membuat data dummy
from .models import Category, News
from .serializers import CategoryDetailSerializer, NewsDetailSerializer


class NewsAppApiTest(TestCase):
    """Membuat data dummy untuk Testing"""

    def setUp(self):
        self.client = APIClient()
        self.user = User.objects.create_user(username='usertest', password='12345')
        token = Token.objects.get(user__username=self.user.username)
        self.client.credentials(HTTP_AUTHORIZATION='Token ' + token.key)
        self.category = Category.objects.create(name='Category Test')

        dummy_image = (
            b'\x47\x49\x46\x38\x39\x61\x01\x00\x01\x00\x00\x00\x00\x21\xf9\x04'
            b'\x01\x0a\x00\x01\x00\x2c\x00\x00\x00\x01\x00\x01\x00\x00\x02'
            b'\x02\x4c\x01\x00\x3b'
        )
        cover = SimpleUploadedFile('small.gif', dummy_image, content_type='image/gif')
```

```
self.news = News.objects.create(  
    title='News Test',  
    excerpt='Short Content',  
    content='Long content with sample text',  
    cover=cover,  
    status=News.NewsStatus.published,  
    user=self.user  
)  
self.news.categories.add(self.category)  
self.news.comment_set.create(name='Commenter', email='commenter@test.com', content='Comment  
→ Test')  
  
"""Testing untuk API endpoint Get Category List"""  
  
def test_can_get_category_list(self):  
    response = self.client.get(reverse('api-category-list'))  
    self.assertEqual(response.status_code, status.HTTP_200_OK)  
  
"""Testing untuk API endpoint Get Category Detail"""  
  
def test_can_get_category_detail(self):  
    response = self.client.get(reverse('api-category-detail', args=[self.category.id]))  
    category = Category.objects.get(pk=self.category.id)  
    serializer = CategoryDetailSerializer(category)  
    self.assertEqual(response.status_code, status.HTTP_200_OK)  
    self.assertEqual(response.data, serializer.data)  
  
"""Testing untuk API endpoint Get Category Detail"""  
"""Status Gagal: ID tidak valid"""  
  
def test_get_cannot_category_detail(self):  
    res = self.client.get(reverse('api-category-detail', args=[999]))  
    self.assertEqual(res.status_code, status.HTTP_404_NOT_FOUND)  
  
"""Testing untuk API endpoint Get News List"""  
  
def test_can_get_news_list(self):  
    """Can get News list"""  
    response = self.client.get(reverse('api-news-list'))  
    self.assertEqual(response.status_code, status.HTTP_200_OK)
```

```
"""Testing untuk API endpoint Get News Detail"""

def test_can_get_news_detail(self):
    response = self.client.get(reverse('api-news-detail', args=[self.news.id]))
    news = News.objects.is_published().get(pk=self.news.id)
    serializer = NewsDetailSerializer(news)
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(response.data.keys(), serializer.data.keys())

"""Testing untuk API endpoint Get News Detail"""

"""Status Gagal: ID tidak valid"""

def test_cannot_get_news_detail(self):
    response = self.client.get(reverse('api-news-detail', args=[999]))
    self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)

"""Testing untuk API endpoint Post News Create Comment"""

def test_can_post_news_create_comment(self):
    payload = {
        'name': 'Commenter',
        'email': 'commenter@test.com',
        'content': 'Another Comment Test'
    }
    response = self.client.post(
        reverse('api-news-create-comment', args=[self.news.id]),
        payload
    )
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)

"""Testing untuk API endpoint Post News Create Comment"""

"""Status Gagal: Payload tidak valid"""

def test_cannot_post_news_create_comment(self):
    payload = {
        'name': 'Commenter',
        'email': '',
        'content': ''
    }
```

```
response = self.client.post(  
    reverse('api-news-create-comment', args=[self.news.id]),  
    payload  
)  
self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)  
  
""" Menghapus data image dummy untuk Testing """  
def tearDown(self):  
    self.news.cover.delete(False)
```

Pada kode di atas kita membuat sebuah *class* dengan nama **NewsAppApiTest** yang merupakan turunan dari class **TestCase**. Di dalam *class* tersebut kita menambahkan *method* untuk menguji setiap **API endpoint** yang sudah kita buat untuk modul **Category** dan juga modul **News**. Simpan perubahan *file* tersebut kemudian buka *terminal/command prompt*. Untuk menjalankan **Unit test** yang sudah kita buat, jalankan perintah berikut.

```
python3.9 manage.py test
```

Perintah di atas akan menampilkan output seperti di bawah ini.

```
(env) sakukode@sakukode:~/myprojects/newsproject$ python3.9 manage.py test  
Creating test database for alias 'default'...  
System check identified no issues (0 silenced).  
.....  
-----  
Ran 8 tests in 1.046s  
  
OK  
Destroying test database for alias 'default'...
```

Penjelasan dari *output* di atas kurang lebih seperti ini. Pertama **Django** akan membuat *database* sementara untuk melakukan *testing*, setelah itu akan dijalankan setiap **Unit test** yang kita tuliskan pada *file newsproject/news/tests.py*. Terdapat keterangan berapa banyak Unit test yang dijalankan (**8 Unit test**) dan waktu yang dibutuhkan untuk melakukan *testing* tersebut (**1.046 detik**). Setelah semua Unit test selesai dijalankan, Django akan menghapus *database* sementara yang digunakan untuk melakukan *testing*.

API Documentation

Dokumentasi untuk sebuah API bisa dibilang merupakan sesuatu hal yang wajib kita buat. Karena tujuan dari kita membuat sebuah API sendiri adalah agar API tersebut bisa digunakan orang lain untuk mengintegrasikan beberapa aplikasi dalam *platform* yang berbeda-beda. Oleh karena itu dengan adanya sebuah Dokumentasi tentunya bisa mempermudah orang lain dalam memahami cara kerja API yang kita buat.

Pertanyaannya bagaimana kita mendokumentasikan API yang sudah kita buat? Kita bisa saja membuat dokumentasi tersebut secara *manual*, semisal dengan menuliskan dokumentasi dengan aplikasi *text editor* atau kita membuat sebuah halaman *web* yang menampilkan cara kerja dan informasi dari API tersebut. Tapi tenang, di sini kita tidak akan membuatnya secara *manual* melainkan menggunakan sebuah *tool* yang nantinya akan secara otomatis meng-generate dokumentasi API yang kita buat. Ada dua *tool* yang akan kita gunakan untuk meng-generate dokumentasi API. Yang pertama ada [Swagger](#) dan yang kedua menggunakan [Redoc](#). Kita memerlukan *package* *drf-yasg* untuk mengintegrasikan *tool-tool* tersebut dengan *djangorestframework* dan *package* tersebut sudah kita instal sebelumnya pada bab **Persiapan dan Instalasi** jadi kita tidak perlu menginstalnya lagi.

Selanjutnya kita perlu melakukan perubahan kembali pada *setting* konfigurasi proyek *Django* kita, untuk itu silahkan buka kembali *file newsproject/app/settings.py*. dan tambahkan baris kode berikut pada bagian **INSTALLED_APPS**.

```
INSTALLED_APPS = [
    ...
    # package untuk integrasi tool dokumentasi api
    'drf_yasg',
]
```

dan tambahkan baris kode berikut pada akhir baris kode *file settings.py* tersebut.

```
SWAGGER_SETTINGS = {
    'SECURITY_DEFINITIONS': {
        'api_key': {
            'type': 'apiKey',
            'in': 'header',
            'name': 'Authorization'
        }
    }
}
```

```
    },
}
```

baris kode di atas digunakan untuk konfigurasi jenis otentikasi pada Rest API yang kita gunakan dan disini kita menggunakan otentikasi dengan api *key/token*. Selanjutnya perubahan *file settings.py* akan menjadi seperti ini.

newsproject/app/settings.py

```
"""
Django settings for newsproject project.

Generated by 'django-admin startproject' using Django 3.1.4.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'bo46w@9act##xuwbzy3_2bs915*r&7okf&gzfgqy-u&liu()3r'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
```

```
# package untuk rest api
'rest_framework',
# package untuk otentikasi rest api
'rest_framework.authtoken',
# package untuk fitur filtering pada rest api
'django_filters',
# package untuk integrasi tool dokumentasi api
'drf_yasg',
# package aplikasi kita
'news.apps.NewsConfig',
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'newsproject.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
            ],
        },
    },
]
```

```
        'django.contrib.messages.context_processors.messages',
    ],
},
},
]
]

WSGI_APPLICATION = 'newsproject.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'newsapp',
        'USER': 'postgres',
        'PASSWORD': '123456',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Asia/Jakarta'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'

REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
}

SWAGGER_SETTINGS = {
    'SECURITY_DEFINITIONS': {
        'api_key': {
            'type': 'apiKey',
            'in': 'header',
            'name': 'Authorization'
        }
    },
}
```

Kita perlu menambahkan *route* baru untuk dokumentasi API dan di sini kita akan menambahkan dua *route* untuk dokumentasi API dengan Swagger dan yang satunya menggunakan Redoc. Silahkan buka file **newsproject/app/urls.py**. Dan ubah menjadi seperti baris kode di bawah

ini.

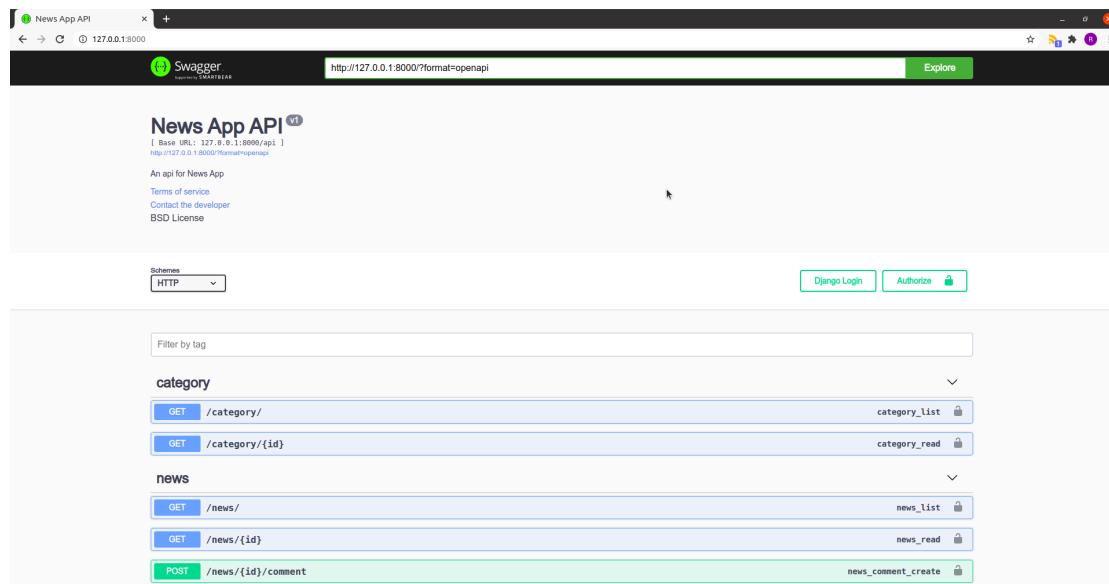
newsproject/app/urls.py

```
from django.contrib import admin
from django.urls import path, include
# import package yang dibutuhkan untuk dokumentasi API
from drf_yasg import openapi
from drf_yasg.views import get_schema_view
from rest_framework import permissions

# pengaturan untuk dokumentasi API (judul, deskripsi dll)
schema_view = get_schema_view(
    openapi.Info(
        title="News App API",
        default_version='v1',
        description="An api for News App",
        terms_of_service="/terms/",
        contact=openapi.Contact(email="sakukode@gmail.com"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)

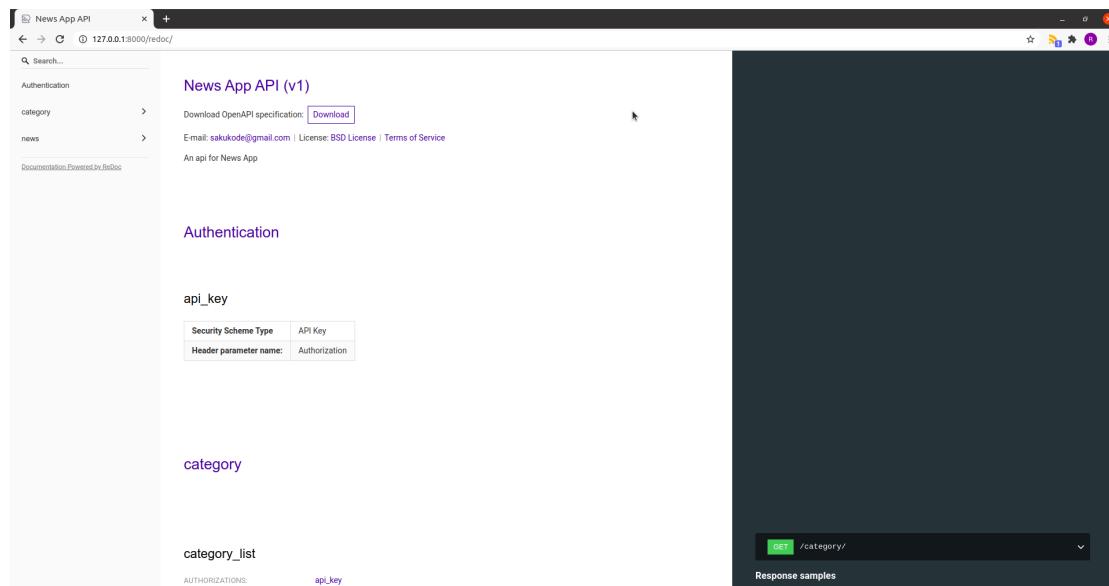
urlpatterns = [
    # routing untuk Rest API
    path('api/', include('news.urls')),
    # routing untuk Admin
    path('admin/', admin.site.urls),
    # dokumentasi API dengan Swagger
    path('', schema_view.with_ui('swagger',
                                 cache_timeout=0), name='schema-swagger-ui'),
    # dokumentasi API dengan Redoc
    path('redoc/', schema_view.with_ui('redoc',
                                 cache_timeout=0), name='schema-redoc'),
]
```

Simpan perubahan *file* di atas kemudian buka *url* <http://127.0.0.1:8000> dan jika kita sukses mengintegrasikan *tool* dokumentasi tersebut maka tampilannya akan seperti gambar di bawah ini.



Gambar 35: Halaman Dokumentasi API dengan Swagger

Gambar di atas merupakan tampilan dokumentasi yang secara otomatis di generate oleh *tool Swagger*. Kita bisa melihat ada daftar API endpoint yang sudah kita buat dan ini semua tergenerate secara otomatis. Kita tidak perlu menuliskannya satu per satu. Ditambah lagi dengan tampilan yang sudah interaktif dan rapi sehingga kita tidak perlu banyak bekerja untuk mengeditnya. Kita punya satu versi dokumentasi API lagi dengan *tool Redoc*. Silahkan buka url <http://127.0.0.1:8000/redoc> pada *browser* dan seharusnya akan tampil seperti gambar di bawah ini.



Gambar 36: Halaman Dokumentasi API dengan Redoc

Terakhir kita akan menambahkan nama dan deskripsi untuk setiap API endpoint pada dokumentasi API sehingga nantinya dokumentasi API yang kita buat lebih informatif bagi pengguna. Untuk melakukan itu silahkan buka *file newsproject/news/views.py*. Tambahkan baris kode berikut di bagian atas untuk mengimport *package* yang dibutuhkan.

```
...
# import package untuk menambahkan judul & deskripsi pada dokumentasi API
from drf_yasg.utils import swagger_auto_schema
from django.utils.decorators import method_decorator
```

Dan tambahkan kode seperti contoh berikut sebelum class Views untuk setiap API endpoint yang ada.

```
# Membuat View untuk API endpoint "Get All Categories"
# /api/category
@method_decorator(name='get', decorator=swagger_auto_schema(operation_id='Get List of Categories',
                                                               operation_description='Returns a paginated array of Categories.'))
class CategoryListView(ListAPIView):
    ...
    
```

Pada baris di atas kita menambahkan nama dan deskripsi untuk API endpoint **Get List of Categories**. Yang perlu diperhatikan adalah *parameter operation_id* digunakan untuk mengatur nama API endpoint dan parameter **operation_description** digunakan untuk mengatur deskripsi API endpoint. Silahkan tambahkan juga untuk API endpoint yang lain. Penggunaan bahasa bisa disesuaikan dan kode di atas hanya contoh dengan menggunakan bahasa inggris. Berikut contoh dari file **views.py** yang sudah diubah.

newsproject/news/views.py

```
# import class yang dibutuhkan untuk membuat view
from django.http import Http404
from rest_framework.filters import SearchFilter, OrderingFilter
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.generics import ListAPIView, RetrieveAPIView, CreateAPIView
# import class untuk otentikasi rest api
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated
# import package untuk menambahkan judul & deskripsi pada dokumentasi API
from drf_yasg.utils import swagger_auto_schema
from django.utils.decorators import method_decorator

# import class models dari aplikasi newsapp
from .models import Category, News, Comment
# import class serializers dari aplikasi newsapp
from .serializers import CategoryListSerializer, CategoryDetailSerializer, NewsListSerializer,
    NewsDetailSerializer, CommentFormSerializer
# import class Filter untuk custom fitur filtering
from .filters import NewsFilter

# Membuat View untuk API endpoint "Get All Categories"
# /api/category
@method_decorator(name='get', decorator=swagger_auto_schema(operation_id='Get List of Categories',
    operation_description='Returns a paginated array of Categories.'))
class CategoryListView(ListAPIView):
    # mengeset class serializers
    serializer_class = CategoryListSerializer
    queryset = Category.objects.all()
    # Menambahkan fitur filtering, searching dan ordering
    filter_backends = (
        DjangoFilterBackend,
```

```
filters.SearchFilter,
filters.OrderingFilter,
)
# mengeset fields/kolom untuk fitur filtering
filter_fields = ['name']
# mengeset fields/kolom untuk fitur searching
search_fields = ['name']
# mengeset fields/kolom untuk fitur ordering
ordering_fields = ['name', 'created_at']
# mengeset fields/kolom default untuk fitur ordering
ordering = ['created_at']
# menambahkan otentikasi
permission_classes = (IsAuthenticated,)
authentication_classes = [TokenAuthentication]

# Membuat View untuk API endpoint "Get Detail of Category"
# /api/category/:id
@method_decorator(name='get', decorator=swagger_auto_schema(operation_id='Get Detail of Category',
                                                               operation_description='Return a single Category by id.'))
class CategoryDetailView(RetrieveAPIView):
    # mengeset class serializers
    serializer_class = CategoryDetailSerializer
    queryset = Category.objects.all()
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]

# Membuat View untuk API endpoint "Get List of News"
# /api/news
@method_decorator(name='get', decorator=swagger_auto_schema(operation_id='Get List of News',
                                                               operation_description='Returns a paginated array of published News.'))
class NewsListView(ListAPIView):
    serializer_class = NewsListSerializer
    # mengambil data news/berita yang hanya berstatus published
    queryset = News.objects.is_published()

    filter_backends = (
        DjangoFilterBackend,
```

```
    SearchFilter,
    OrderingFilter,
)
# mengeset fields/kolom untuk fitur filtering dengan class NewsFilter (custom filter)
filterset_class = NewsFilter
# mengeset fields/kolom untuk fitur searching
search_fields = ['title']
# mengeset fields/kolom untuk fitur ordering
ordering_fields = ['title', 'created_at']
# mengeset fields/kolom default untuk fitur ordering
ordering = ['created_at']
# menambahkan otentikasi
permission_classes = (IsAuthenticated,)
authentication_classes = [TokenAuthentication]

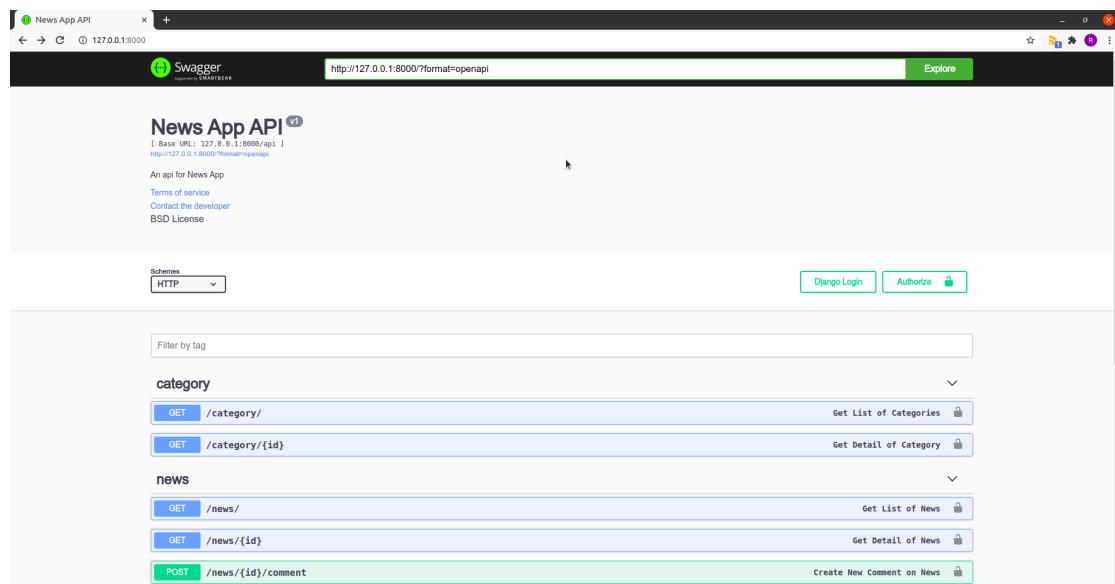
# Membuat View untuk API endpoint "Get Detail of News"
# /api/news/:id
@method_decorator(name='get', decorator=swagger_auto_schema(operation_id='Get Detail of News',
                                                               operation_description='Return a single News by id.'))
class NewsDetailView(RetrieveAPIView):
    serializer_class = NewsDetailSerializer
    # mengambil data news/berita yang hanya berstatus published
    queryset = News.objects.is_published()
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]

# Membuat View untuk API endpoint "Create New Comment on News"
# /api/news/:id/comment
@method_decorator(name='post', decorator=swagger_auto_schema(operation_id='Create New Comment on
                  News',
                                                               operation_description='Create a new comment on news.'))
class NewsCreateCommentView(CreateAPIView):
    serializer_class = CommentFormSerializer
    queryset = Comment.objects.all()
    # menambahkan otentikasi
    permission_classes = (IsAuthenticated,)
    authentication_classes = [TokenAuthentication]
```

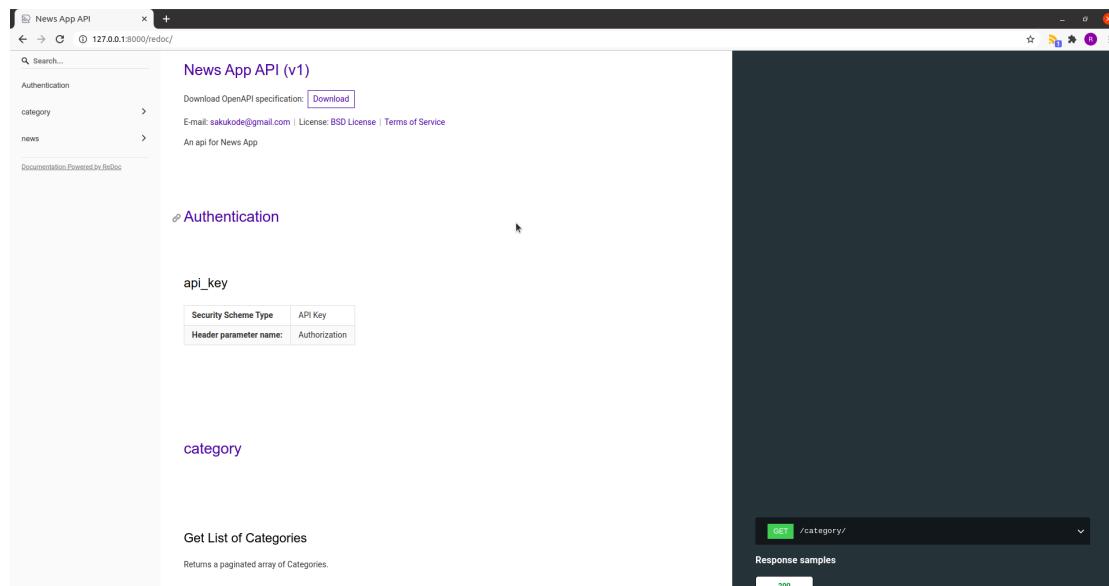
```
# meng-override method perform_create untuk mengambil news_id dari parameter url API endpoint
# Kemudian disimpan pada kolom news_id di tabel Comment pada saat kita melakukan requests endpoint
# ini.

def perform_create(self, serializer):
    news_id = self.kwargs['pk']
    try:
        news = News.objects.get(pk=news_id)
        serializer.save(news_id=news.id)
    except News.DoesNotExist:
        raise Http404
```

Perubahan tampilan Dokumentasi API nya akan menjadi seperti gambar di bawah ini.



Gambar 37: Halaman Dokumentasi API dengan Redoc



Gambar 38: Halaman Dokumentasi API dengan Redoc

Terlihat lebih informatif dan rapi dengan adanya nama dan deskripsi untuk setiap API endpoint yang ada. Pada contoh di atas kita menggunakan dua *tool* untuk membuat dokumentasi. Silahkan pilih salah satu yang kamu suka atau tetap ingin menggunakan keduanya dan kamu bisa mengeksplor *tool* dokumentasi API tersebut. Di akhir bab ini akan kami sertakan *link* referensi untuk *tool* dokumentasi ini.

Referensi:

- <https://drf-yasg.readthedocs.io/en/stable/index.html>
- <https://swagger.io/>
- <https://redoc.ly/>

Mendeploy Aplikasi

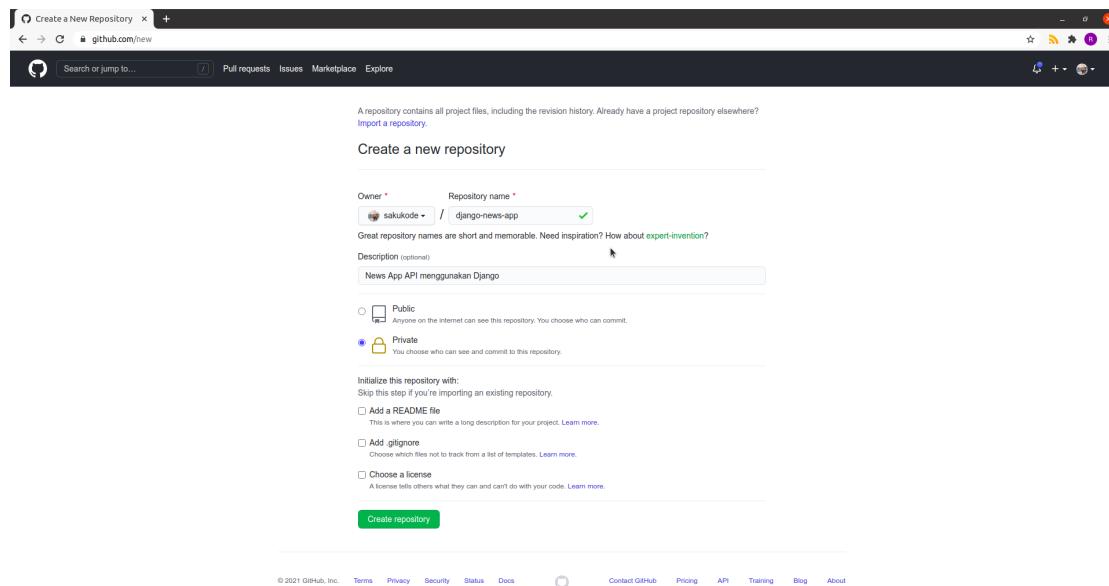
Jika kita ingin proyek aplikasi yang sudah kita bangun dapat diakses dari jaringan mana saja maka kita perlu mendeploy proyek kita ke sebuah server. Proses deploying aplikasi web secara sederhana bisa diartikan proses pemindahan folder dan file dari sebuah aplikasi web dari penyimpanan lokal ke sebuah server online dan kita melakukan konfigurasi pada server tersebut sehingga nantinya aplikasi web kita dapat diakses melalui sebuah alamat ip address public atau diakses melalui url sebuah domain.

Membuat Repository pada Github

Sebelum melakukan deploying proyek kita, alangkah baiknya untuk menyimpan source code proyek ke dalam sebuah repository. Ini bertujuan untuk memudahkan kita dalam memanage proyek kita. Contohnya ketika kita melakukan perubahan atau menambahkan sebuah modul baru pada proyek kita, kita tidak perlu mengupload file atau folder satu persatu secara manual. Kita cukup menggunakan tool yang bernama Git untuk melakukan sync proyek kita dari local komputer ke shared hosting maupun VPS. Banyak layanan penyedia VCS (Version Control System) yang bisa kita pilih di antaranya ada [Github](#) , [Gitlab](#) , [Bitbucket](#) ataupun yang lainnya. Dan pada buku ini kita akan menggunakan Github untuk membuat Repository untuk menyimpan source code kita.

Jika kamu belum mempunyai akun github, silahkan buat akun dulu dengan melakukan registrasi pada link berikut: <https://github.com/join>

Buat sebuah repository baru melalui link berikut <https://github.com/new> . Isikan nama repository, deskripsi, public/private dan yang lainnya seperti gambar di bawah ini:



Gambar 39: Membuat Repository Baru pada Github

Klik tombol Create repository untuk menyelesaikan langkah pembuatan repository baru. Kembali ke proyek kita, buka terminal/command prompt dan masuk ke dalam folder proyek kita. Ketikkan perintah berikut untuk menginisiasi git pada folder proyek kita.

```
git init
```

Buatlah dua file baru README.md dan .gitignore di dalam folder proyek kita.

README.md

```
# News App API menggunakan Django
News App API with Django Rest Framework
```

.gitignore

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
```

```
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
pip-wheel-metadata/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
# Usually these files are written by a python script from a template
# before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
```

```
nosetests.xml
coverage.xml
*.cover
*.py.cover
.hypothesis/
.pytest_cache/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
.python-version
```

```
# pipenv
# According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version control.
# However, in case of collaboration, if having platform-specific dependencies or dependencies
# having no cross-platform support, pipenv may install dependencies that don't work, or not
# install all needed dependencies.
#Pipfile.lock

# PEP 582; used by e.g. github.com/David-OConnor/pyflow
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json
```

```
# Pyre type checker  
.pyre/
```

```
# pycharm IDE config  
.idea
```

Jalankan perintah di bawah ini secara berurutan untuk melakukan push (mengupload file dan folder proyek ke repository) ke repository yang sudah kita buat.

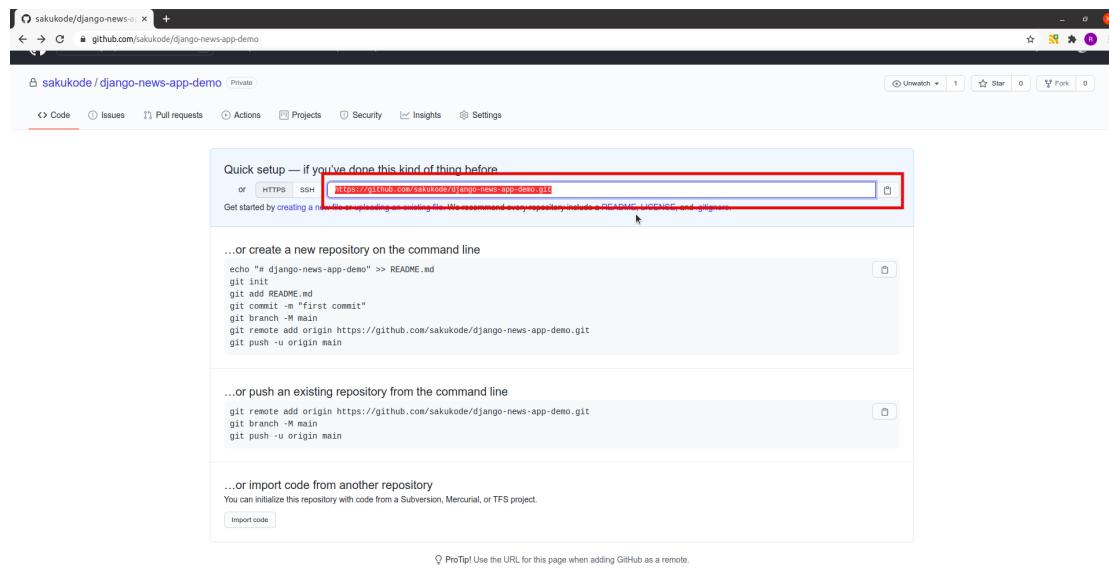
```
git add .
```

```
git commit -m 'first commit'
```

```
git branch -M main
```

```
git remote add origin https://github.com/sakukode/django-news-app-demo.git
```

Untuk tahap di bawah ini, ubah url nya dengan url repository kamu yang sudah kamu buat pada tahap sebelumnya. Kamu bisa mengcopy url nya dari halaman di github seperti gambar di bawah ini.



Gambar 40: Copy URL Repository pada Github

Jika tidak muncul pesan error pada tahap-tahap sebelumnya, terakhir ketikkan perintah di bawah ini.

```
git push -u origin main
```

Kamu akan diminta memasukkan username dan password akun Github kamu, jika benar maka proses push ke repository akan dijalankan.

```
sakukode@sakukode:~/newsproject$ git push -u origin main
Username for 'https://github.com': sakukode
Password for 'https://sakukode@github.com':
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 4 threads
Compressing objects: 100% (23/23), done.
Writing objects: 100% (25/25), 74.61 KiB | 8.29 MiB/s, done.
Total 25 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/sakukode/django-news-app-demo.git
 * [new branch]  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

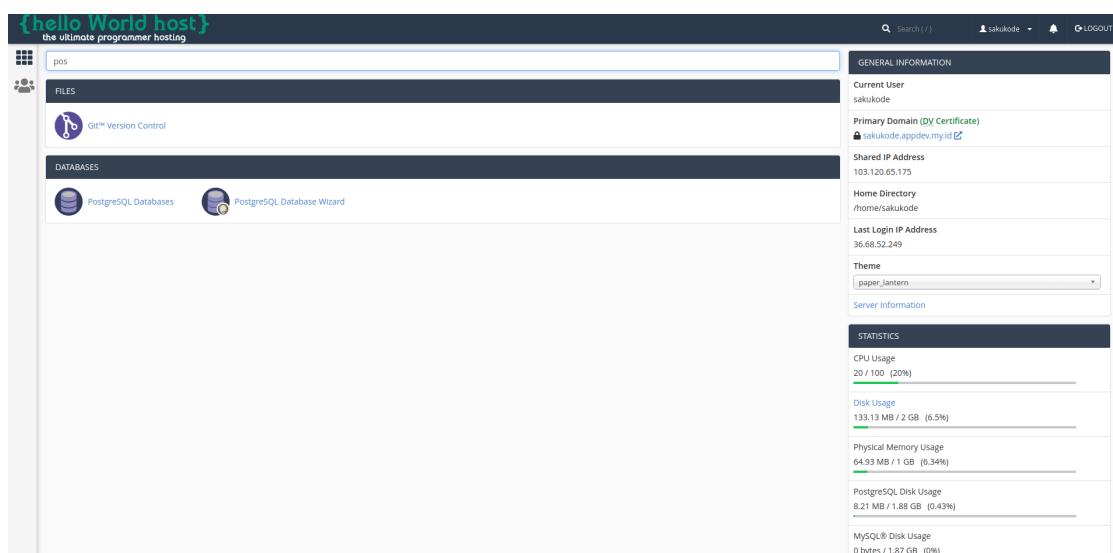
Deploy ke Shared Hosting

Shared hosting adalah layanan hosting di mana sebuah account hosting ditaruh bersama-sama beberapa account hosting lain dalam satu server yang sama, dan memakai services bersama-sama. Keuntungan shared hosting adalah harganya yang murah jadi cocok untuk kamu yang sedang tahap proses belajar dan ingin mencoba mendeploy aplikasi secara online.

Sebelum mengikuti materi pada tahap ini pastikan kamu sudah membeli atau berlangganan shared hosting yang di dalamnya terdapat service untuk men-setup aplikasi python dan mendukung database PostgreSQL. Jika kamu bingung untuk mengeceknya kamu bisa langsung menanyakannya pada CS dari layanan hosting yang kamu beli.

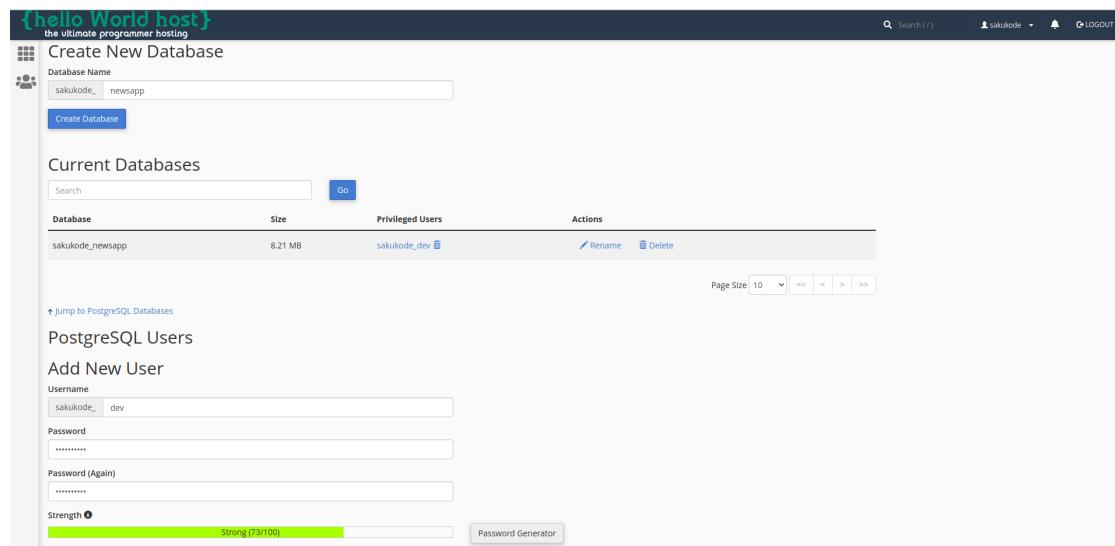
Membuat Username dan Database PostgreSQL

Langkah pertama yang bisa kita lakukan adalah membuat username dan database PostgreSQL. Silahkan login ke dalam cpanel shared hosting dan cari dan masuk ke menu atau service Postgresql Databases.

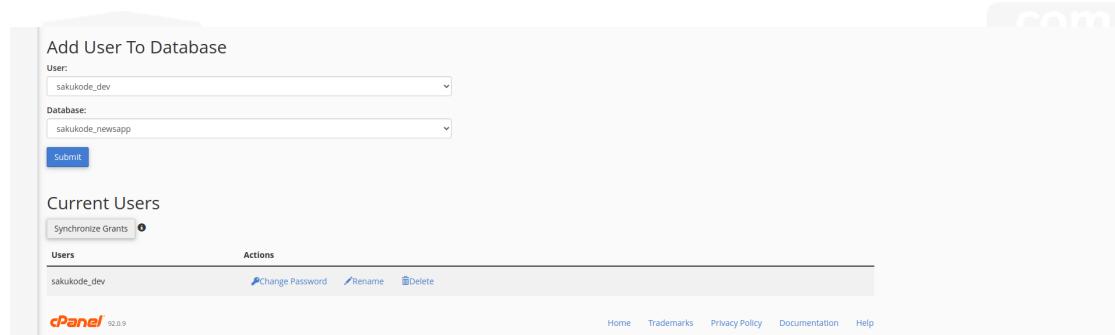


Gambar 41: Menu PostgreSQL Cpanel Shared Hosting

Buat sebuah Database baru dengan mengisikan nama database kemudian klik Create Database. Setelah itu Buat sebuah user baru dengan mengisikan username dan password dan klik Create User.

**Gambar 42:** Menu PostgreSQL Cpanel Shared Hosting

Dan Terakhir tambahkan user tersebut ke database yang sudah kita buat sebelumnya dengan memilih user dan database dari pilihan yang ada, klik Submit.

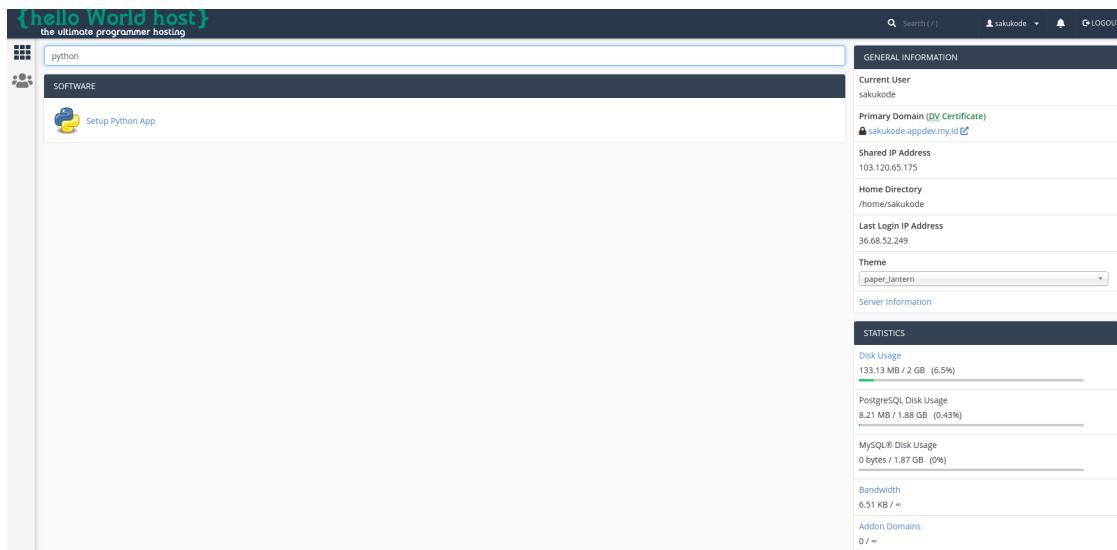
**Gambar 43:** Menu PostgreSQL Cpanel Shared Hosting

Hal yang perlu kamu ingat atau catat adalah nama database, username dan password yang sudah kita buat pada tahap di atas karena nantinya kita perlu mengatur konfigurasi proyek kita dengan database dan user tersebut.

Setup Aplikasi Python

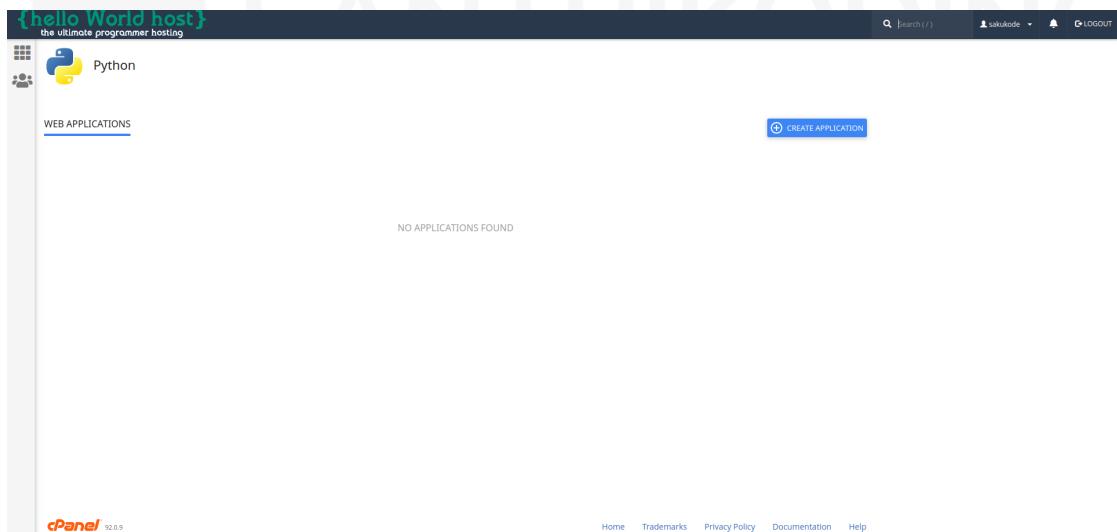
Kita membangun proyek kita dengan framework Django yang notabene merupakan framework berbasis bahasa python. Untuk itu kita perlu membuat atau mensetup aplikasi python pada shared

hosting agar nantinya aplikasi kita bisa berjalan. Dari menu beranda cpanel silahkan cari dan masuk ke dalam service atau menu Setup Python App.



Gambar 44: Menu Setup Python App Cpanel Shared Hosting

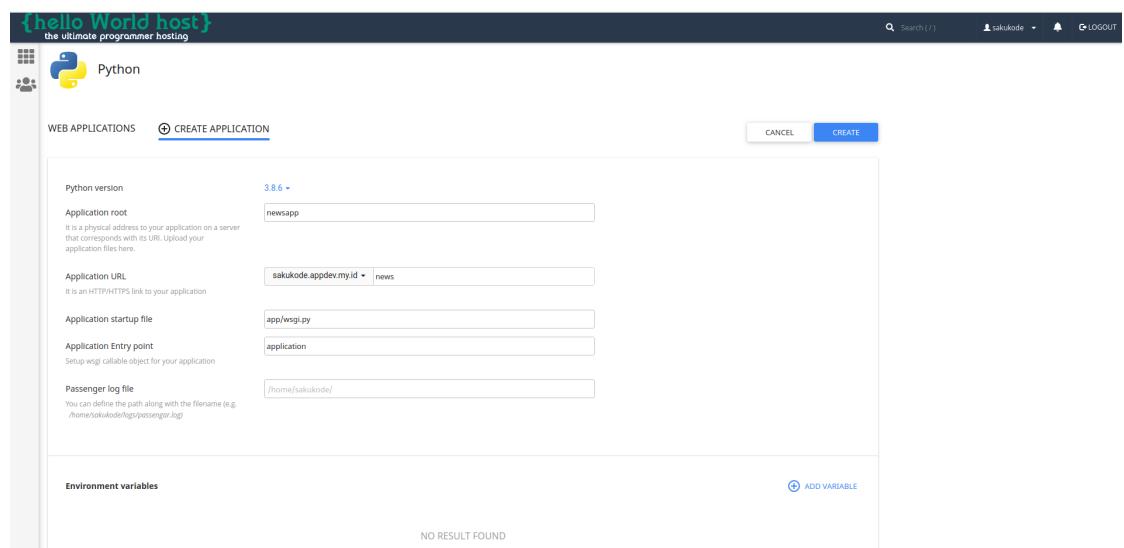
Klik tombol Create Application untuk membuat aplikasi python baru.



Gambar 45: Menu Setup Python App Cpanel Shared Hosting

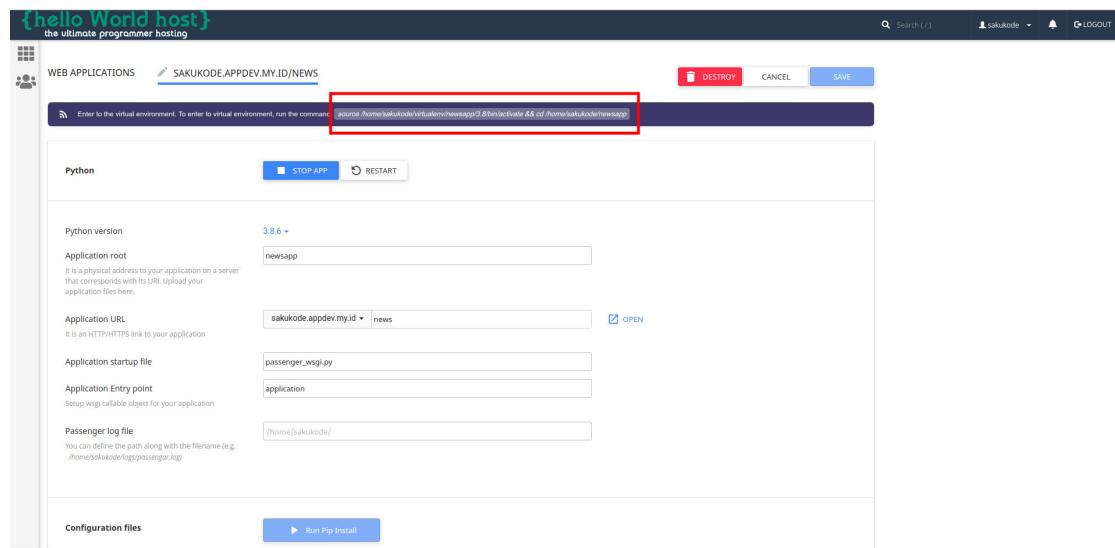
Akan muncul halaman seperti di bawah ini. Pilih versi Python yang akan kita gunakan (jpilih python versi 3.9 atau pilih versi yang paling tinggi atau mendekati versi 3.9). Silahkan isi

Application root dengan nama folder atau proyek kita. Kemudian Application URL isi sesuai yang kamu inginkan (apakah akan diarahkan ke domain utama, subdomain atau subfolder dari domain utama). Sedangkan isian lainnya silahkan isi seperti gambar di bawah ini. Klik Create untuk menyimpan.



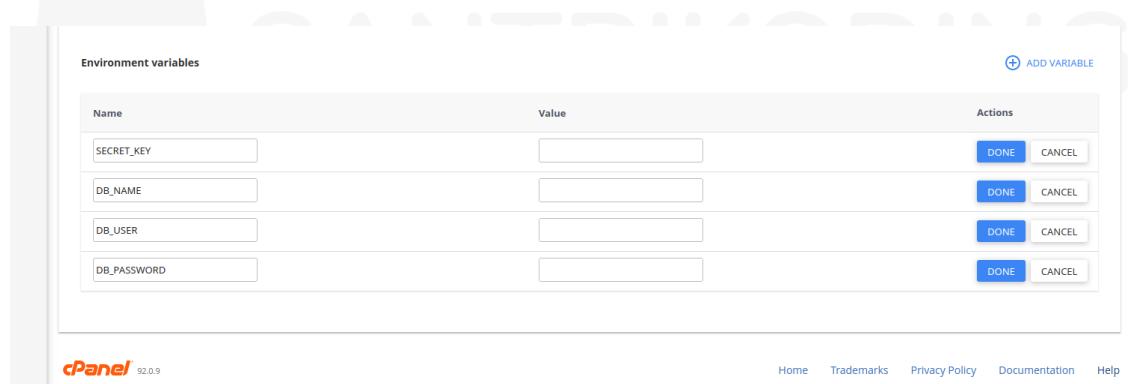
Gambar 46: Menu Setup Python App Cpanel Shared Hosting

Pada bagian atas (background warna gelap) akan ditampilkan script atau perintah untuk masuk atau mengaktifkan virtual environment (Enter the virtual environment...) dari aplikasi python yang sudah kita buat. Copy script tersebut dan simpan karena nantinya script ini akan kita gunakan pada tahap selanjutnya.



Gambar 47: Menu Setup Python App Cpanel Shared Hosting

Pada bagian bawah halaman kamu bisa melihat menu untuk menambahkan environment variabel yang akan kita gunakan pada proyek kita. Silahkan buat beberapa variabel seperti di bawah ini dan isikan valuenya. Jangan lupa klik DONE untuk menyimpan.



Gambar 48: Menu Setup Python App Cpanel Shared Hosting

Langkah selanjutnya kita perlu login ke shared hosting via ssh (jika kamu tidak tahu caranya silahkan hubungin CS nya). Setelah berhasil login, copy script yang sudah kita copy sebelumnya pada halaman setup python app. kemudian paste ke terminal seperti contoh di bawah ini.

```
source /home/sakukode/virtualenv/newsapp/3.8/bin/activate && cd /home/sakukode/newsapp
```

Hapus semua file dan folder yang ada di dalamnya.

```
rm -rf *
```

Kita akan mengclone proyek kita menggunakan Git tool. Silahkan ketikkan perintah di bawah ini untuk menginisiasi git.

```
git init
```

Tambahkan git remote url dengan mengetikkan perintah berikut (ubah url repositorynya dengan url repository kamu).

```
git remote add origin https://github.com/sakukode/django-news-app-demo.git
```

Cek kembali apakah url repository yang kita tambahkan sudah benar

```
git remote -v
```

Output:

```
(newsapp:3.8)[sakukode@public newsapp]$ git remote -v
origin  https://github.com/sakukode/django-news-app-demo.git (fetch)
origin  https://github.com/sakukode/django-news-app-demo.git (push)
```

Selanjutnya ketikkan perintah di bawah ini untuk melakukan pulling/mendownload proyek dari repository

```
git pull origin main
```

Kamu akan diminta memasukkan username dan password akun Github. Isikan dan tekan Enter.

```
Username for 'https://github.com': sakukode
Password for 'https://sakukode@github.com':
From https://github.com/sakukode/django-news-app-demo
 * branch      main    -> FETCH_HEAD
(newsapp:3.8)[sakukode@public newsapp]$
```

Pastikan semua file dan folder sudah tercloning dengan mengetikkan perintah berikut.

```
ls -la
```

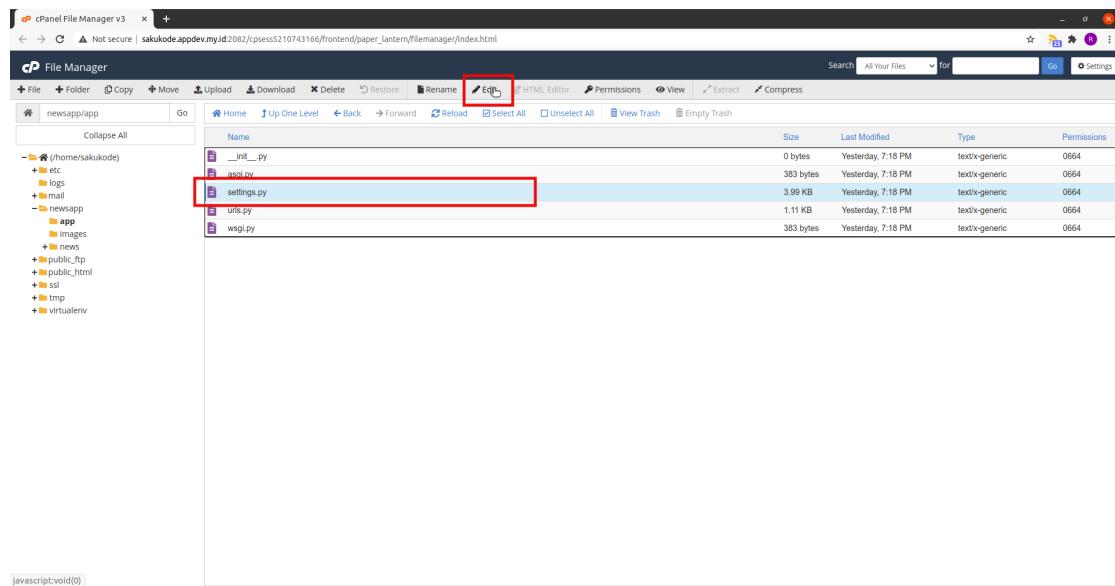
Output:

```
(newsapp:3.8)[sakukode@public newsapp]$ ls -la
total 40
drwxr-xr-x  6 sakukode sakukode 4096 Peb  5 19:18 .
drwx--x--x 22 sakukode sakukode 4096 Peb  1 08:48 ..
drwxrwxr-x  2 sakukode sakukode 4096 Peb  5 19:18 app
drwxrwxr-x  7 sakukode sakukode 4096 Peb  5 19:19 .git
-rw-rw-r--  1 sakukode sakukode 1812 Peb  5 19:18 .gitignore
drwxrwxr-x  2 sakukode sakukode 4096 Peb  5 19:18 images
-rwxrwxr-x  1 sakukode sakukode  659 Peb  5 19:18 manage.py
drwxrwxr-x  3 sakukode sakukode 4096 Peb  5 19:18 news
-rw-rw-r--  1 sakukode sakukode   74 Peb  5 19:18 README.md
-rw-rw-r--  1 sakukode sakukode  119 Peb  5 19:18 requirements.txt
(newsapp:3.8)[sakukode@public newsapp]$
```

Tahap berikutnya kita butuh menginstal package-package python yang ada pada file requirements.txt

```
pip install -r requirements.txt
```

Jika tidak ada pesan error berarti kita telah berhasil menginstal package-package tersebut. Oke kita kembali lagi ke halaman cpanel di browser dan masuk ke halaman beranda. Pada kolom pencarian silahkan cari menu atau service File Manager kemudian klik menu tersebut. Di dalam menu File Manager akan terdapat daftar folder dan file di dalam penyimpanan shared hosting kita. Cari file settings.py di dalam folder python app kemudian klik menu Edit.



Gambar 49: Menu File Manager Cpanel Shared Hosting

Lakukan perubahan pada file tersebut seperti kode di bawah ini kemudian klik tombol Save Changes.

```
.....
Django settings for app project.

Generated by 'django-admin startproject' using Django 3.1.4.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""

import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
```

```
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/
```

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get("SECRET_KEY")
```

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

```
ALLOWED_HOSTS = [
    '127.0.0.1',
    '*'
]
```

```
# Application definition
```

```
INSTALLED_APPS = [
    # package untuk rest api
    'rest_framework',
    # package untuk otentikasi rest api
    'rest_framework.authtoken',
    # package untuk fitur filtering pada rest api
    'django_filters',
    # package untuk integrasi tool dokumentasi api
    'drf_yasg',
    # package aplikasi kita
    'news.apps.NewsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
```

```
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'app.urls'

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
],
]

WSGI_APPLICATION = 'app.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get("DB_NAME"),
        'USER': os.environ.get("DB_USER"),
        'PASSWORD': os.environ.get("DB_PASSWORD"),
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

```
# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validation

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Asia/Jakarta'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, "static")
```

```
REST_FRAMEWORK = {  
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',  
    'PAGE_SIZE': 10,  
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],  
    'DEFAULT_AUTHENTICATION_CLASSES': [  
        'rest_framework.authentication.TokenAuthentication',  
    ],  
}  
  
SWAGGER_SETTINGS = {  
    'SECURITY_DEFINITIONS': {  
        'api_key': {  
            'type': 'apiKey',  
            'in': 'header',  
            'name': 'Authorization'  
        }  
    },  
}
```

Pada kode di atas kita melakukan perubahan untuk pengaturan variabel `ALLOWED_HOST` , `SECRET_KEY` , `DATABASE` , serta variabel `STATIC_URL` dan `STATIC_ROOT`. Selanjutnya kita perlu menjalankan migration dan membuat user baru pada python app kita.

Silahkan login kembali ke shared hosting via ssh dari terminal dan jangan lupa masuk ke folder python app dan aktifkan virtual environmentnya (jika lupa caranya, silahkan baca kembali pada tahap sebelumnya).

Jalankan migration dengan mengetikkan perintah berikut.

```
python manage.py migrate
```

Output:

```
(newsapp:3.8)[sakukode@public newsapp]$ python manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, auth, authtoken, contenttypes, news, 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 authtoken.0001_initial... OK
Applying authtoken.0002_auto_20160226_1747... OK
Applying authtoken.0003_tokenproxy... OK
Applying news.0001_initial... OK
Applying sessions.0001_initial... OK
(newsapp:3.8)[sakukode@public newsapp]$
```

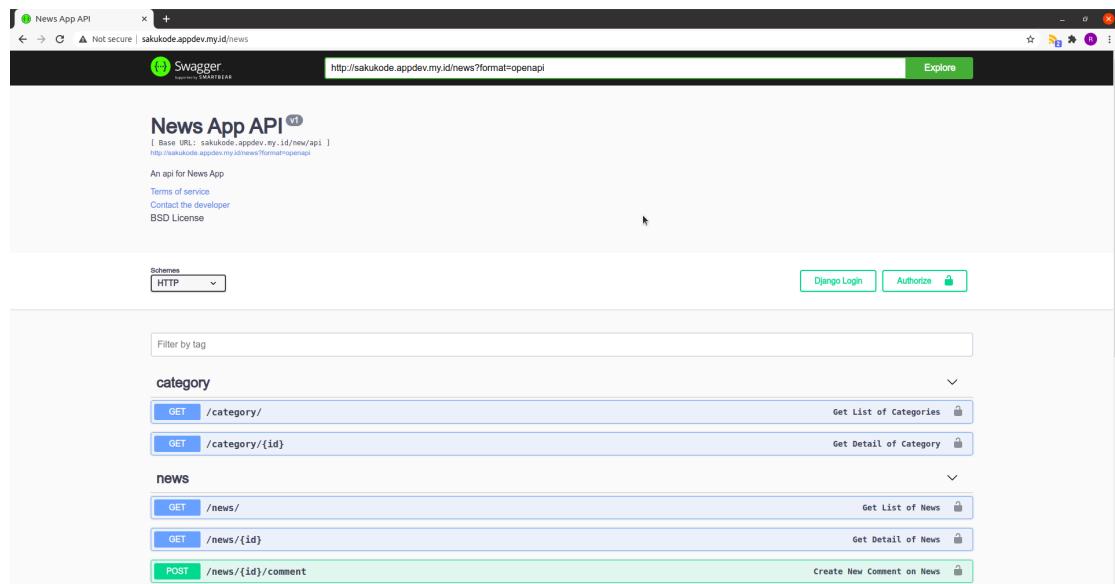
Selanjutnya kita perlu menggenerate asset file (css,js,image) pada python app dengan menjalankan perintah berikut.

```
python manage.py collectstatic
```

Terakhir buatlah symlink ke folder static dari folder python app kita ke folder public_html (Ini diperlukan agar file asset kita dapat diakses).

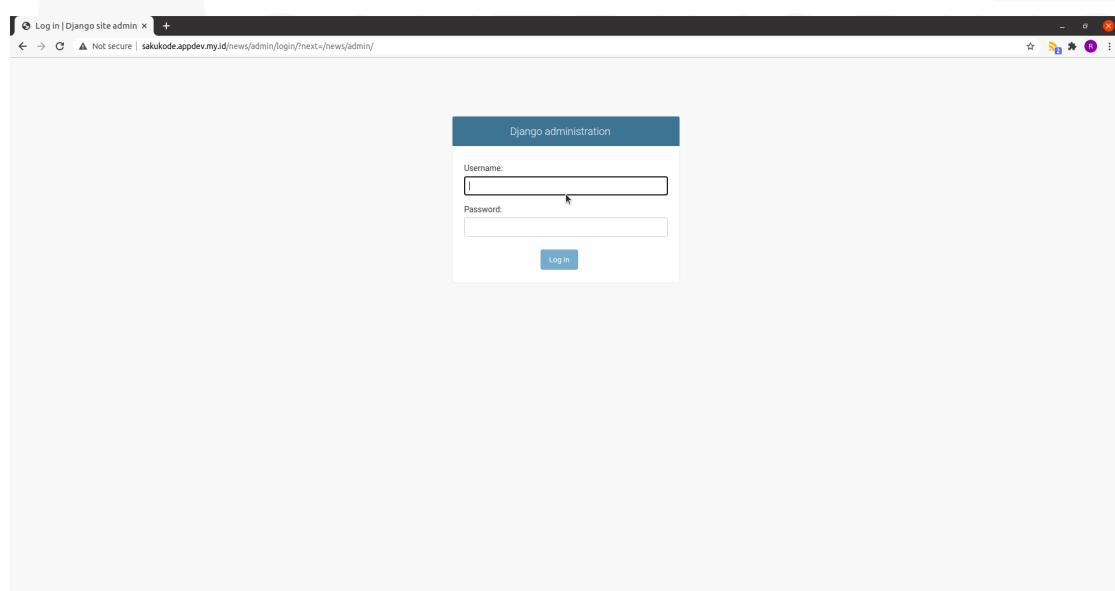
```
ln -sf /home/sakukode/newsapp/static /home/sakukode/public_html/static
```

Kita telah selesai mendeploy proyek kita ke shared hosting. Silahkan cek dengan mengetikkan alamat domain website kamu di browser. Jika tidak ada error maka akan tampil halaman home dari proyek kita dimana itu merupakan halaman dokumentasi dari API proyek kita.



Gambar 50: Halaman API Documentation live di Shared Hosting

Untuk login ke halaman django admin silahkan tambahkan `/admin` pada alamat domain website kita.



Gambar 51: Halaman Dashboard Admin Login live di Shared Hosting