Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

KETIKKAN TOPIK ANDA

1. REST API

a. Framework yang digunakan : Django Rest Framework

b. Model

1. Library di models

```
import sys
from datetime import datetime, timedelta
from django.db import models
from django.contrib.auth.models import AbstractUser
from PIL import Image
from io import BytesIO
from django.core.files.uploadedfile import InMemoryUploadedFile
```

modul sys diimpor untuk akses ke variabel dan fungsi sistem. datetime dan timedelta digunakan untuk menangani tanggal dan waktu. Library Django, models, digunakan untuk mendefinisikan model database, dan AbstractUser memungkinkan penyesuaian model pengguna.

Selain itu, library Pillow (PIL) dengan modul Image digunakan untuk membuka dan memanipulasi gambar. BytesIO dari modul io mengelola stream byte dalam memori, berguna untuk operasi input/output yang tidak membutuhkan penulisan data ke disk. InMemoryUploadedFile dari Django mengelola file yang diunggah dalam memori, memungkinkan manipulasi gambar sebelum penyimpanan.

2. User

```
class User(AbstractUser):
    is_admin = models.BooleanField(default=False)
    is_user = models.BooleanField(default=False)

    def __str__(self):
        return str(self.username) + ' ' + str(self.first_name) + ' ' + str(self.last_name)
```

model User yang diperluas dari AbstractUser dalam aplikasi Django. Model ini menambahkan dua atribut tambahan: is_admin dan is_user, keduanya berupa BooleanField dengan nilai default False. Atribut ini digunakan untuk menentukan peran pengguna dalam aplikasi. Fungsi _str_ di-overwrite untuk mengembalikan representasi string dari objek pengguna, yang menggabungkan username, first_name, dan last_name. Ini memungkinkan pengenalan pengguna yang lebih mudah saat objek pengguna ditampilkan dalam antarmuka admin atau log.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

3. Status Model

```python
class StatusModel(models.Model):
    status_choices = (
        ('Aktif', 'Aktif'),
        ('Tidak Aktif', 'Tidak Aktif')
    )
    name = models.CharField(max_length=50, unique=True)
    status = models.TextField(blank=True, null=True)
    user_create = models.ForeignKey(User, related_name='user_create_status_model', blank=True, null=True, on_delete=models.SET_NULL)
    user_update = models.ForeignKey(User, related_name='user_update_status_model', blank=True, null=True, on_delete=models.SET_NULL)
    created_on = models.DateTimeField(auto_now_add=True)
    last_modified = models.DateTimeField(auto_now=True)

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

Model ini memiliki beberapa atribut: name yang berupa CharField dengan panjang maksimum 50 karakter dan harus unik, serta status yang berupa TextField yang opsional. Ada dua ForeignKey yang menghubungkan ke model User untuk mencatat pengguna yang membuat dan memperbarui status, dengan opsi SET_NULL jika pengguna terkait dihapus. Selain itu, terdapat dua atribut waktu, created_on dan last_modified, yang otomatis diisi saat pembuatan dan pembaruan. Pilihan status status_choices berisi dua nilai: 'Aktif' dan 'Tidak Aktif'. Metode _str_ mengembalikan nama status sebagai representasi string dari objek.Serta status model juga terkoneksi dengan class lain seperti kategori pulau,profile,user yang menandakan model itu aktif atau tidak

4. Profile

```python
class Profile(models.Model):
    user = models.OneToOneField(User, related_name='user_profile', on_delete=models.PROTECT)
    avatar = models.ImageField(default=None, upload_to='profile_images/', blank=True, null=True)
    biografi = models.TextField()
    status = models.ForeignKey(StatusModel, related_name='status_profile', default=None, on_delete=models.PROTECT)
    user_create = models.ForeignKey(User, related_name='user_create_profile', blank=True, null=True, on_delete=models.SET_NULL)
    user_update = models.ForeignKey(User, related_name='user_update_profile', blank=True, null=True, on_delete=models.SET_NULL)
    created_on = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self.user.first_name} {self.user.last_name} {self.user.id}'

    def save(self, force_insert=False, force_update=False, using=None, update_fields=None, *args, **kwargs):
        if self.id:
            try:
                this = Profile.objects.get(id=self.id)
                if this.avatar and this.avatar != self.avatar:
                    this.avatar.delete(save=False)
            except Profile.DoesNotExist:
                pass

        if self.avatar:
            var_avatar = self.avatar
            self.avatar = compress_image(var_avatar, 'profile')

        super(Profile, self).save(*args, **kwargs)


def compress_image(image, category):
    img = Image.open(image)
    img = img.convert('RGB')
    output = BytesIO()
    img.save(output, format='JPEG', quality=85)
    output.seek(0)
    return InMemoryUploadedFile(output, 'ImageField', f"{image.name.split('.')[0]}_{category}.jpg", 'image/jpeg', sys.getsizeof(output), None)
```

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

Model ini terhubung satu-satu dengan model User melalui OneToOneField, yang mengamankan profil pengguna agar tidak dapat dihapus dengan on_delete=models.PROTECT. Model ini memiliki beberapa atribut tambahan seperti avatar untuk menyimpan gambar profil yang diunggah, biografi sebagai teks, dan status yang terhubung ke model StatusModel. Atribut user_create dan user_update mencatat pengguna yang membuat dan memperbarui profil, dengan on_delete=models.SET_NULL agar nilai menjadi null jika pengguna terkait dihapus. Tanggal pembuatan otomatis dicatat dengan created_on.

Metode _str_ mengembalikan representasi string dari profil berupa nama depan, nama belakang, dan ID pengguna. Metode save diperbarui untuk menghapus avatar lama saat diperbarui dan memanggil fungsi compress_image untuk mengompres gambar sebelum menyimpannya. Fungsi compress_image mengompres gambar ke format JPEG dengan kualitas 85%, mengurangi ukuran file sebelum menyimpannya ke dalam memori.

5. Kategori Pulau

```python
class KategoriPulau(models.Model):
    pulau = models.CharField(max_length = 100)
    status = models.ForeignKey(StatusModel, related_name= 'status_pulau', default=get_default_status, on_delete=models.PROTECT)

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

Model ini memiliki dua atribut: pulau, berupa CharField dengan panjang maksimum 100 karakter, dan status, berupa ForeignKey yang terhubung ke model StatusModel. Atribut status menggunakan fungsi get_default_status untuk menentukan status default, dan dilindungi dari penghapusan dengan on_delete=models.PROTECT.

6. Table List Hotel

```python
class TableListHotel(models.Model):
    status_choices = (
        ('Aktif', 'Aktif'),
        ('Tidak Aktif', 'Tidak Aktif')
    )
    # status_table_choices = (
    #     ('Buka', 'Buka'),
    #     ('Tutup', 'Tutup'),
    # )

    code = models.CharField(max_length=100)
    name = models.TextField(max_length=100)
    # table_status = models.CharField(max_length=15, choices=status_table_choices, default='Buka' )
    status = models.CharField(max_length=15, choices=status_choices, default='Aktif')
    user_create = models.ForeignKey(User, related_name='user_create_table_listhotel', blank=True, null=True, on_delete=models.SET_NULL)
    user_update = models.ForeignKey(User, related_name='user_update_table_listhotel', blank=True, null=True, on_delete=models.SET_NULL)
    created_on = models.DateTimeField(auto_now_add=True)
    last_modified = models.DateTimeField(auto_now=True)

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

Model ini mencakup beberapa atribut: code berupa CharField dengan panjang maksimum 100 karakter, dan name berupa TextField dengan panjang maksimum 100 karakter. Atribut status menggunakan pilihan status_choices yang bisa berupa 'Aktif' atau 'Tidak Aktif', dengan default 'Aktif'. Dua atribut user_create dan user_update mencatat pengguna yang membuat dan memperbarui entri, dengan on_delete=models.SET_NULL jika pengguna terkait dihapus.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

Tanggal pembuatan otomatis dicatat dengan created_on, dan tanggal modifikasi terakhir dicatat dengan last_modified.

7. Table RatingBintang

```python
class RatingBintang(models.Model):
    jumlah_bintang_hotel_choices = (
        ('1', '1'),
        ('2', '2'),
        ('3', '3'),
        ('4', '4'),
        ('5', '5'),
    )
    nama = models.CharField(max_length=3, null=False, blank= False ,default= None)
    jumlah_bintang_hotel = models.CharField(max_length=100, choices=jumlah_bintang_hotel_choices, default='1')

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

Model ini memiliki dua atribut utama: nama, berupa CharField dengan panjang maksimum 3 karakter dan tidak boleh kosong, serta jumlah_bintang_hotel, berupa CharField dengan pilihan jumlah_bintang_hotel_choices yang bisa bernilai dari '1' hingga '5', dengan default '1'.

8. Table Detail Hotel

```python
class DetailHotel(models.Model):
    hotel = models.ForeignKey(TableListHotel, on_delete=models.CASCADE, related_name='detail')
    pulau = models.ForeignKey(KategoriPulau, on_delete=models.CASCADE, related_name='detail', default= KategoriPulau.objects.first().pk)
    alamat = models.TextField(max_length=100)
    deskripsi = models.TextField()
    fasilitas = models.TextField()
    informasi_kamar = models.TextField()
    google_maps_link = models.URLField(max_length=100, blank=True, null=True)
    rating_bintang = models.ForeignKey(RatingBintang, related_name='rating_detail_hotel', blank=True, null=True, on_delete=models.SET_NULL)

    def __str__(self):
        return f"DetailHotel {self.hotel.name}"
```

model DetailHotel ini digunakan untuk menyimpan detail dari suatu hotel, seperti nama hotel, pulau tempat hotel tersebut berada, alamat hotel, deskripsi hotel, fasilitas yang disediakan, informasi tentang kamar-kamar yang tersedia, serta tautan Google Maps jika ada.

c. Controller

1. Library di controller

```python
from rest_framework import generics
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from rest_framework import permissions
from rest_framework.authtoken.models import Token
from hotel_app.models import TableListHotel, DetailHotel, RatingBintang, KategoriPulau, StatusModel
from api.serializers import TableListHotelSerializers, DetailHotelSerializers, RatingBintangSerializers, KategoriPulauSerializer, RegisterUserSerializer, LoginSerializer
from django.contrib.auth import login as django_login
from django.http import HttpResponse, JsonResponse
from rest_framework.generics import GenericAPIView
from rest_framework.authentication import SessionAuthentication, BasicAuthentication, TokenAuthentication
from rest_framework.permissions import IsAuthenticated, AllowAny
from django.contrib.auth import logout
from .paginators import  CustomPagination
# from django_filters.rest_framework import DjangoFilterBackend
```

Import Modul: Kode dimulai dengan mengimpor beberapa modul yang diperlukan dari Django REST Framework dan juga beberapa modul khusus dari aplikasi hotel_app.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

Deklarasi Class Views: Kita mendefinisikan beberapa class untuk views menggunakan Django REST Framework:

generics: Digunakan untuk membuat views generik seperti ListCreateAPIView.
APIView: Digunakan untuk membuat views berbasis fungsi khusus yang lebih fleksibel.
Import Model dan Serializer: Model dan serializer dari aplikasi hotel_app diimpor untuk digunakan dalam views.

Import dan Penggunaan Autentikasi dan Perizinan: Kita mengimpor dan menggunakan berbagai jenis autentikasi seperti SessionAuthentication, BasicAuthentication, dan TokenAuthentication serta perizinan seperti IsAuthenticated dan AllowAny.

Login dan Register User: Terdapat implementasi untuk fungsi login dan register user. Fungsi django_login digunakan untuk masuk (login) pengguna dan juga terdapat serialisasi untuk registrasi pengguna baru.

Paginasi Kustom: Digunakan untuk membuat paginasi kustom dengan mengimpor CustomPagination dari file paginators.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

2. Table List Hotel Views

```python
class TableListHotelViews(APIView):
    authentication_classes = [TokenAuthentication,SessionAuthentication, BasicAuthentication]
    permission_classes = [IsAuthenticated]

    def get(self, request, *args, **kwargs):
        table_listhotel = TableListHotel.objects.filter(status='Aktif')
        serializer = TableListHotelSerializers(table_listhotel, many=True)
        response = {
            'status' : status.HTTP_200_OK,
            'message' : 'Pembacaan seluruh data berhasil',
            'user' :str(request.user),
            'auth' :str(request.auth),
            'data' : serializer.data,
        }
        return Response(response, status= status.HTTP_200_OK)

    def post(self, request, *args, **kwargs):
        data = {
            'code': request.data.get('code'),
            'name': request.data.get('name'),
        }
        serializer = TableListHotelSerializers(data=data)
        if serializer.is_valid():
            serializer.save()
            response = {
                'status': status.HTTP_201_CREATED,
                'message': 'Data Berhasil Dibuat....',
                'data': serializer.data
            }
            return Response(response, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

model TableListHotel. Di sini, kita menggunakan APIView dari Django REST Framework untuk mengelola permintaan HTTP. Hanya pengguna yang sudah terautentikasi (authenticated) yang bisa mengakses views ini, dan autentikasi dilakukan dengan token, session, atau basic authentication.

Ketika permintaan GET diterima, views akan mengambil semua objek TableListHotel yang memiliki status 'Aktif'. Data ini kemudian diserialisasikan dan dikirim sebagai respons bersama dengan pesan sukses dan status HTTP 200 OK.

Ketika permintaan POST diterima, views akan mencoba untuk membuat objek TableListHotel baru berdasarkan data yang diberikan dalam permintaan. Jika data valid, objek akan disimpan dan respons yang sesuai dengan pesan sukses dan status HTTP 201 CREATED akan dikirimkan. Jika data tidak valid, respons akan berisi pesan kesalahan dan status HTTP 400 BAD REQUEST.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

3. Table Detail Hotel Views

```python
class DetailHotelViews(APIView):
    def get_object(self, id):
        try:
            return DetailHotel.objects.get(id=id)
        except DetailHotel.DoesNotExist:
            return None

    def get(self, request, id=None, *args, **kwargs):
        if id:
            detail_hotel = self.get_object(id)
            if not detail_hotel:
                return Response(
                    {
                        'status': status.HTTP_400_BAD_REQUEST,
                        'message': 'Data tidak ada',
                        'data': {}
                    }, status=status.HTTP_400_BAD_REQUEST
                )
            serializer = DetailHotelSerializers(detail_hotel)
            response = {
                'status': status.HTTP_200_OK,
                'message': 'Data berhasil diambil',
                'data': serializer.data
            }
            return Response(response, status=status.HTTP_200_OK)
        else:
            detail_hotels = DetailHotel.objects.all()
            serializer = DetailHotelSerializers(detail_hotels, many=True)
            return Response(serializer.data, status=status.HTTP_200_OK)
```

```python
def put(self, request, id=None, *args, **kwargs):
    detail_hotel = self.get_object(id)
    if not detail_hotel:
        return Response(
            {
                'status': status.HTTP_400_BAD_REQUEST,
                'message': 'Data tidak ada',
                'data': {}
            }, status=status.HTTP_400_BAD_REQUEST
        )

    data = {
        'hotel': request.data.get('hotel'),
        'alamat': request.data.get('alamat'),
        'deskripsi': request.data.get('deskripsi'),
        'fasilitas': request.data.get('fasilitas'),
        'gambar_hotel': request.data.get('gambar_hotel'),
        'informasi_kamar': request.data.get('informasi_kamar'),
        'google_maps_link': request.data.get('google_maps_link'),
        'rating_bintang': request.data.get('rating_bintang'),
    }
    serializer = DetailHotelSerializers(instance=detail_hotel, data=data, partial=True)
    if serializer.is_valid():
        serializer.save()
        response = {
            'status': status.HTTP_200_OK,
            'message': 'Data berhasil diupdate',
            'data': serializer.data
        }
        return Response(response, status=status.HTTP_200_OK)

    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

```python
def delete(self, request, id=None, *args, **kwargs):
    detail_hotel = self.get_object(id)
    if not detail_hotel:
        return Response(
            {
                'status': status.HTTP_400_BAD_REQUEST,
                'message': 'Data tidak ada',
                'data': {}
            }, status=status.HTTP_400_BAD_REQUEST
        )

    detail_hotel.delete()
    response = {
        'status': status.HTTP_200_OK,
        'message': 'Data berhasil dihapus'
    }
    return Response(response, status=status.HTTP_200_OK)
```

model DetailHotel. Views ini mengimplementasikan operasi dasar CRUD (Create, Read, Update, Delete) untuk objek DetailHotel dalam aplikasi.

Ketika permintaan GET diterima, views akan mencari objek DetailHotel berdasarkan ID yang diberikan. Jika ID ditemukan, data akan diserialisasikan dan dikirim sebagai respons dengan pesan sukses dan status HTTP 200 OK. Jika ID tidak ditemukan, respons akan berisi pesan kesalahan dan status HTTP 400 BAD REQUEST. Jika ID tidak disediakan, views akan mengambil semua objek DetailHotel, men-serialize-nya, dan mengirimkan sebagai respons.

Ketika permintaan PUT diterima, views akan mencoba untuk memperbarui objek DetailHotel yang sesuai dengan ID yang diberikan. Jika objek ditemukan, data akan diperbarui berdasarkan data yang diterima dalam permintaan. Jika data valid, objek akan disimpan dan respons yang sesuai dengan pesan sukses dan status HTTP 200 OK akan dikirimkan. Jika tidak valid, respons akan berisi pesan kesalahan dan status HTTP 400 BAD REQUEST.

Ketika permintaan DELETE diterima, views akan mencoba untuk menghapus objek DetailHotel yang sesuai dengan ID yang diberikan. Jika objek ditemukan, objek akan dihapus dan respons yang sesuai dengan pesan sukses dan status HTTP 200 OK akan dikirimkan. Jika ID tidak ditemukan, respons akan berisi pesan kesalahan dan status HTTP 400 BAD REQUEST.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

4. Rating Bintang Views

```python
class RatingBintangViews(APIView):
    def get(self, request, *args, **kwargs):
        rating_bintang = RatingBintang.objects.all()
        serializer = RatingBintangSerializers(rating_bintang, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

    def post(self, request, *args, **kwargs):
        data = {
            'hotel': request.data.get('hotel'),
            'jumlah_bintang_hotel': request.data.get('jumlah_bintang_hotel'),
            'ulasan': request.data.get('ulasan')
        }
        serializer = RatingBintangSerializers(data=data)
        if serializer.is_valid():
            serializer.save()
            response = {
                'status': status.HTTP_201_CREATED,
                'message': 'Rating Bintang Berhasil Dibuat....',
                'data': serializer.data
            }
            return Response(response, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

model RatingBintang. Views ini memungkinkan pengguna untuk melihat semua data rating bintang yang tersedia dan juga membuat rating baru untuk hotel.

Ketika permintaan GET diterima, views akan mengambil semua objek RatingBintang, men-serialize-nya, dan mengirimkannya sebagai respons dengan status HTTP 200 OK.

Ketika permintaan POST diterima, views akan mencoba membuat objek RatingBintang baru berdasarkan data yang diberikan dalam permintaan. Data tersebut meliputi hotel yang dirating, jumlah bintang, dan ulasan. Jika data valid, objek akan disimpan dan respons yang sesuai dengan pesan sukses dan status HTTP 201 CREATED akan dikirimkan. Jika tidak valid, respons akan berisi pesan kesalahan dan status HTTP 400 BAD REQUEST.

5. Kategori Pulau Views

```python
class KategoriPulauViews(APIView):
    def get(self, request, *args, **kwargs):
        kategori_pulau = KategoriPulau.objects.all()
        serializer = KategoriPulauSerializer(kategori_pulau, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

    def post(self, request, *args, **kwargs):
        data = {
            'pulau': request.data.get('pulau'),
        }
        serializer = KategoriPulauSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            response = {
                'status': status.HTTP_201_CREATED,
                'message': 'Data created successfully...',
                'data': serializer.data
            }
            return Response(response, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

6. Register User View

```python
class RegisterUserAPIView(APIView):
    serializer_class = RegisterUserSerializer

    def post(self, request, format = None):
        serializer = self.serializer_class(data = request.data)
        if serializer.is_valid():
            serializer.save()
            response_data = {
                'status' : status.HTTP_201_CREATED,
                'message' : 'Selamat anda telah terdaftar...',
                'data' : serializer.data,
            }
            return Response(response_data, status = status.HTTP_201_CREATED)
        return Response({
                'status' : status.HTTP_400_BAD_REQUEST,
                'data' : serializer.errors
            }, status = status.HTTP_400_BAD_REQUEST)
```

Kode ini adalah bagian dari sebuah API yang digunakan untuk mendaftarkan pengguna baru. Kelas RegisterUserAPIView memiliki metode post yang menangani permintaan POST untuk pendaftaran. Ketika ada permintaan pendaftaran, data yang dikirimkan oleh pengguna akan diambil dan diperiksa apakah valid atau tidak.

Jika data tersebut valid, data pengguna akan disimpan dan API akan mengembalikan respons dengan status HTTP 201 Created, bersama dengan pesan "Selamat anda telah terdaftar..." dan data pengguna tersebut. Namun, jika data yang dikirimkan tidak valid, API akan mengembalikan respons dengan status HTTP 400 Bad Request serta pesan kesalahan yang terjadi.

7. LoginView

```python
class LoginView(GenericAPIView):
    permission_classes = []
    serializer_class = LoginSerializer

    def post(self, request):
        serializer = LoginSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        user = serializer.validated_data['user']
        django_login(request, user)

        # Membuat atau mengambil token autentikasi
        token, created = Token.objects.get_or_create(user=user)

        return Response({
            'status': status.HTTP_200_OK,
            'message': 'Selamat anda berhasil masuk...',
            'data': {
                'token': token.key,  # Mengirim token autentikasi
                'id': user.id,
                'first_name': user.first_name,
                'last_name': user.last_name,
                'email': user.email,
                'is_admin': user.is_admin,
                'is_user': user.is_user,
            }
        })
```

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

Views ini menggunakan LoginSerializer untuk memvalidasi data yang diterima dari permintaan POST.

Ketika permintaan POST diterima, views akan mencoba untuk memvalidasi data yang diberikan. Jika data valid, pengguna akan diautentikasi dan token autentikasi akan dibuat atau diambil dari database. Respons yang dikirimkan akan berisi pesan sukses, status HTTP 200 OK, dan informasi pengguna yang diautentikasi, termasuk token autentikasi yang akan digunakan untuk otorisasi dalam permintaan berikutnya.

8. Logout View

```
class LogoutView(APIView):
    def get(self, request, *args, **kwargs):
        logout(request)
        response = {
            'status': status.HTTP_200_OK,
            'message': 'Logout berhasil'
        }
        return Response(response, status=status.HTTP_200_OK)
```

ini adalah views untuk proses logout pengguna. Ketika permintaan GET logout diterima, views akan melakukan proses logout pengguna yang terkait dengan permintaan tersebut.

Setelah logout berhasil dilakukan, respons yang dikirimkan akan berisi pesan sukses dan status HTTP 200 OK

9. Table List Hotel Filter

```
class TableListHotelFilterApi(generics.ListAPIView):
    queryset = TableListHotel.objects.all()
    serializer_class = TableListHotelSerializers
    pagination_class = CustomPagination
    permissions_classes = [permissions.IsAuthenticated]
    # filter_backends = [DjangoFilterBackend,]
    ordering_fields = ['created_on']
```

ini adalah views untuk menyediakan endpoint API yang memungkinkan pengguna untuk melakukan filter pada model TableListHotel.

Views ini menggunakan ListAPIView dari Django REST Framework yang memungkinkan pengguna untuk melakukan operasi baca saja (GET) pada data TableListHotel. Queryset default yang digunakan adalah semua objek TableListHotel. Serializer yang digunakan untuk menangani data adalah TableListHotelSerializers.

Selain itu, views juga menggunakan CustomPagination untuk membagi hasil data menjadi beberapa halaman.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

Hanya pengguna yang sudah terautentikasi yang bisa mengakses endpoint ini, yang diatur melalui permissions_classes = [permissions.IsAuthenticated].

Pengguna juga dapat melakukan pengurutan hasil berdasarkan kolom created_on, yang diatur menggunakan ordering_fields = ['created_on'].

d. Routing (url)
Screenshot dari *routing (url)* yang anda bangun dan uraikan narasi dari masing-masing *routing (url)* yang anda bangun

1. api/urls.py

```python
from django.urls import path,include
from api import views
from rest_framework.urlpatterns import format_suffix_patterns
from .views import(
    TableListHotelViews,DetailHotelViews,RegisterUserAPIView,LoginView,LogoutView, TableListHotelFilterApi, TableListHotelFilterApi
)

app_name = 'api'
urlpatterns = [

    path('api/register', RegisterUserAPIView.as_view()),
    path('api/login', LoginView.as_view()),
    path('api/logout', LogoutView.as_view()),
    path('api/table_list_hotel_filter', TableListHotelFilterApi.as_view()),
    path('api/table_listhotel',views.TableListHotelViews.as_view()),
    path('api/detailhotel', views.DetailHotelViews.as_view(), name='detailhotel'),
    path('api/detailhotel/<int:id>', views.DetailHotelViews.as_view(), name='detailhotel_detail'),
    path('api/rating_bintang', views.RatingBintangViews.as_view()),
    path('api/kategori_pulau', views.KategoriPulauViews.as_view()),
    path('api/table-list-hotel-filter/', views.TableListHotelFilterApi.as_view())

]
```

Ini adalah file urls.py yang berisi konfigurasi URL untuk aplikasi API.

Setiap path dihubungkan dengan views yang sesuai menggunakan as_view() untuk menghasilkan tampilan yang dapat diakses oleh Django.Pengguna dapat mendaftar (register), masuk (login), dan keluar (logout) dari akun mereka menggunakan endpoint yang disediakan.Selain itu, terdapat juga endpoint untuk mengakses data TableListHotel, DetailHotel, RatingBintang, dan KategoriPulau.Endpoint khusus juga disediakan untuk melakukan filter pada data TableListHotel.Semua URL ini terhubung dengan views yang relevan untuk mengakses atau memanipulasi data sesuai dengan permintaan pengguna.

Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

2. Pos/urls.py

```python
from django.contrib import admin
from django.urls import path, include
from hotel_app import views
from api import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('super-admin/', admin.site.urls),
    path('',include('api.urls',namespace = 'api')),

]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,document_root=settings.MEDIA_ROOT)
    urlpatterns += static(settings.STATIC_URL,document_root=settings.STATIC_ROOT)
```

Ini adalah file urls.py yang merupakan titik awal untuk routing URL dalam aplikasi.Pertama, terdapat path untuk mengakses halaman admin Django pada endpoint super-admin/.

Kemudian, terdapat path kosong ('') yang mengarahkan ke file urls.py dalam aplikasi api, yang telah didefinisikan sebelumnya.Jika pengaturan DEBUG diaktifkan, juga ada konfigurasi untuk menyajikan file media dan statis selama pengembangan aplikasi.

Penggunaan include('api.urls', namespace='api') menunjukkan bahwa semua URL yang dimulai dengan api/ akan ditangani oleh file urls.py dalam aplikasi api dengan namespace api.Ini adalah konfigurasi dasar untuk mengarahkan permintaan pengguna ke views yang relevan dalam aplikasi, baik itu untuk mengelola data dari API atau halaman admin Django

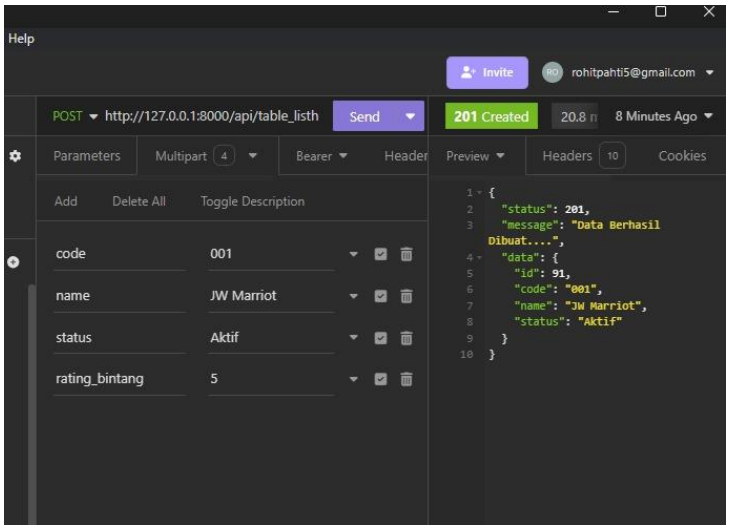
e. Middleware
Tidak ada Middleware

f. Input - Proses/Transaksi - Output REST API (service GET, POST, PUT dan DELETE)
Screenshot dari Input - Proses/Transaksi - Output REST API yang anda bangun dan uraikan narasi dari Input - Proses/Transaksi - Output REST API yang anda bangun. Lalu tampilkan hasil screenshot dari REST API yang dibangun menggunakan aplikasi Postman atau Insomnia *(pilih salah satu)*.

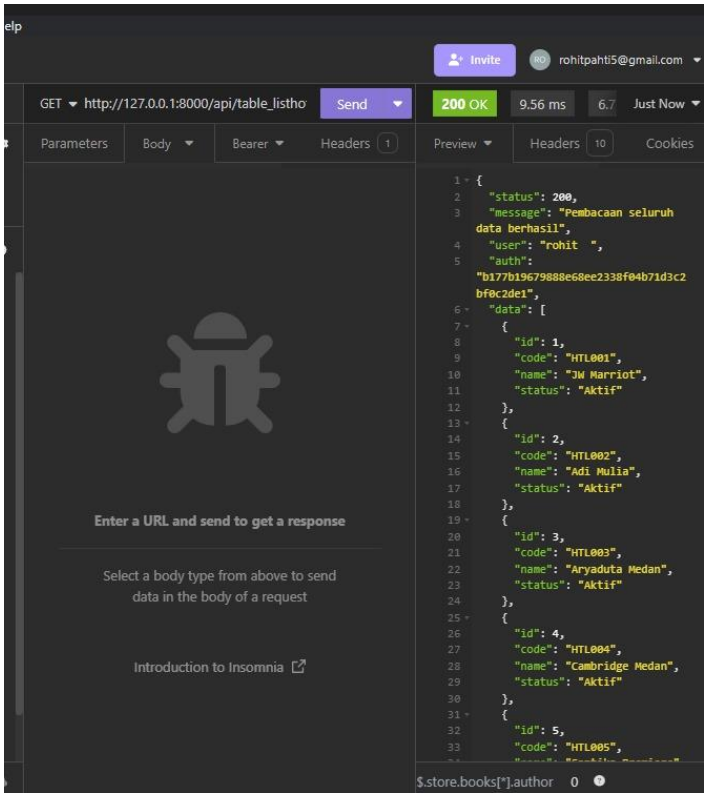
Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

1. POST/TableListHotel



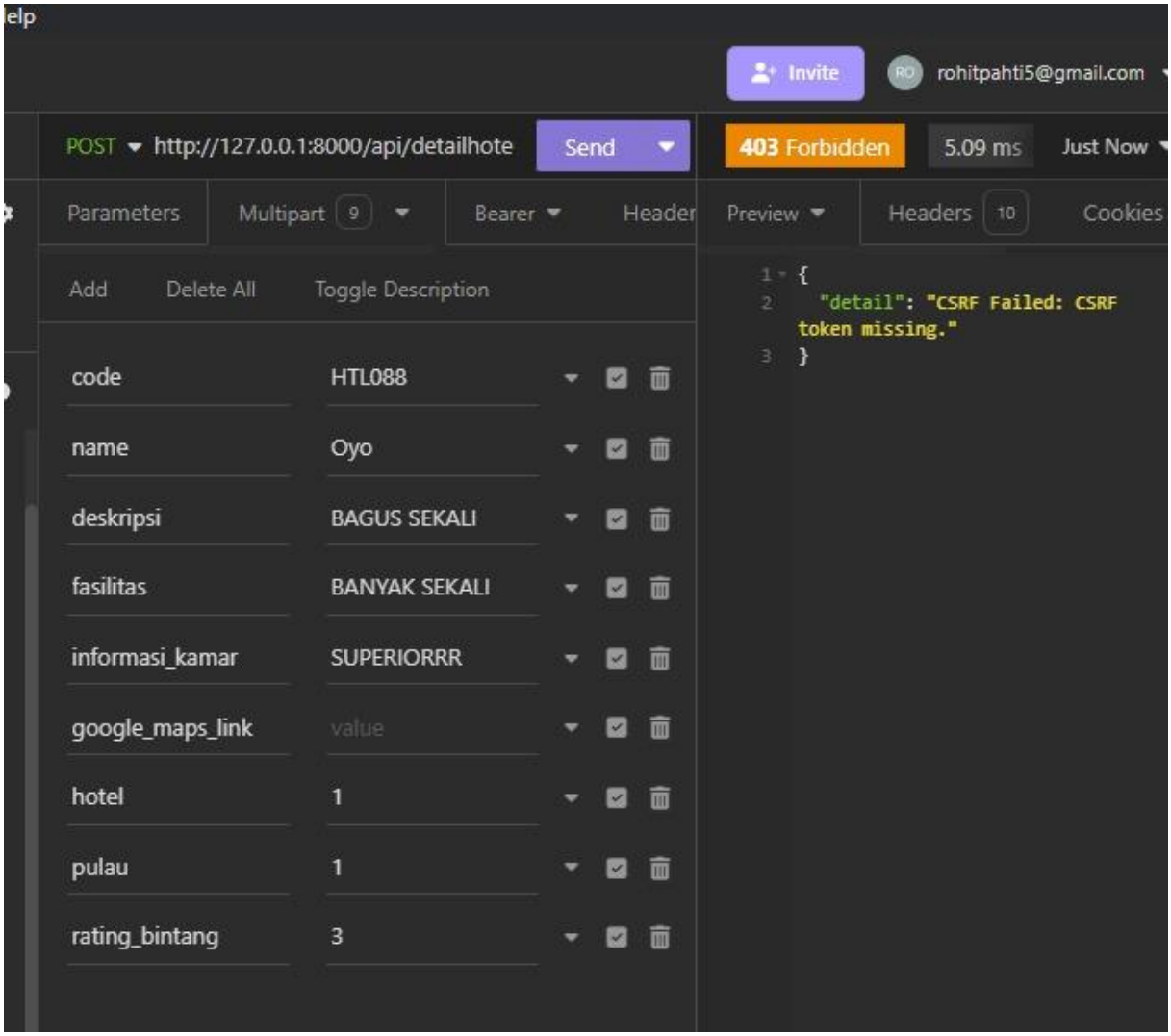
2. GET/TableListHotel



Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

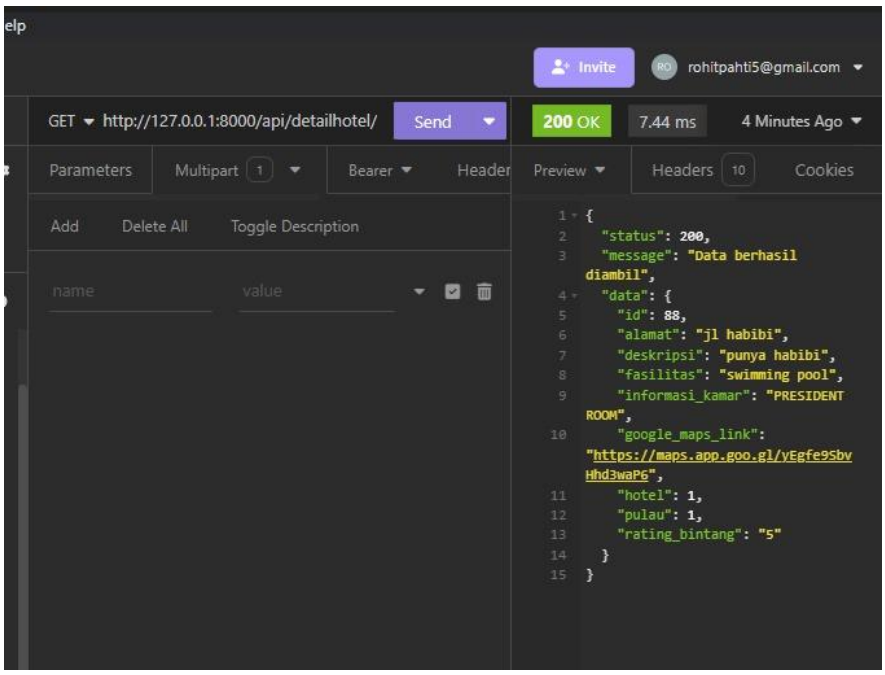
3. GET/Detail hotel



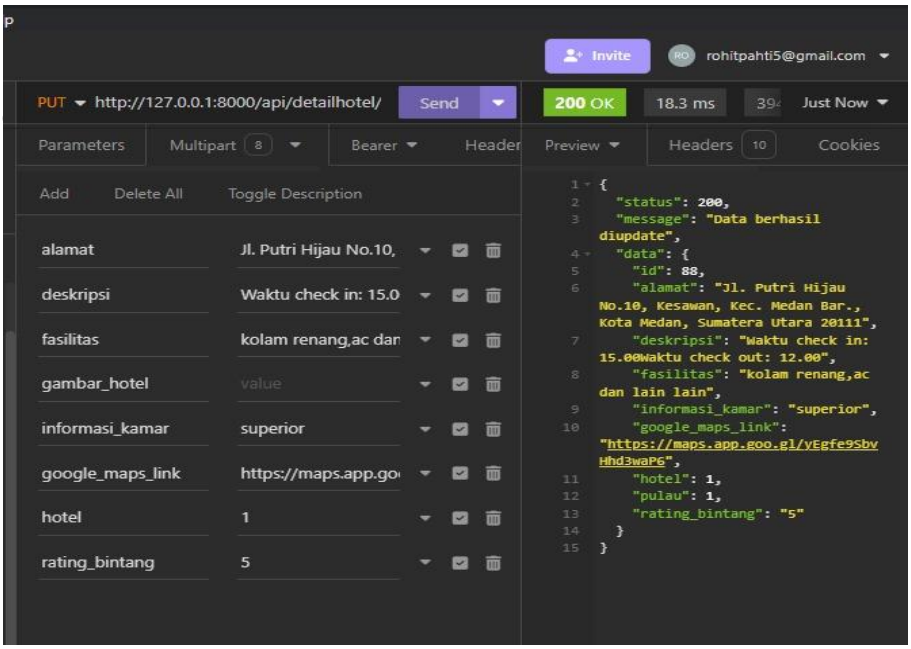
Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

4. GET/Detail Hotel



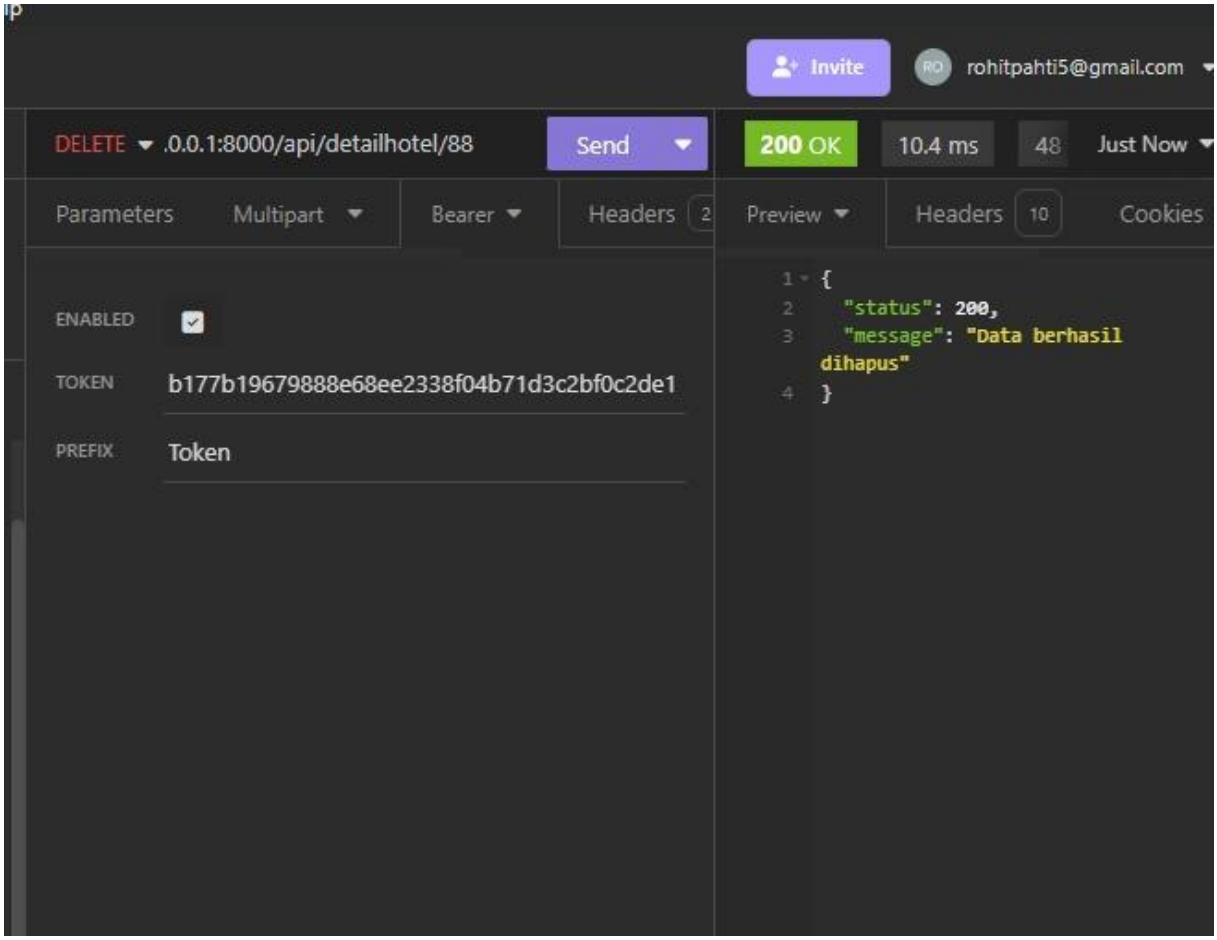
5. PUT/Detail Hotel



Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

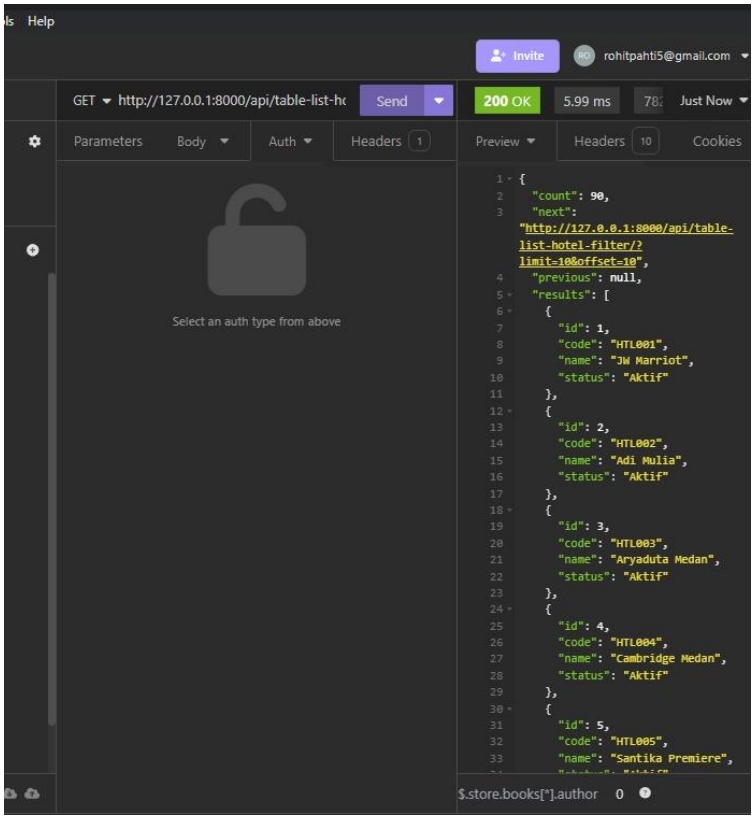
6. Delete/Detail Hotel



Nama TIM : TIM 1, TIM 2, dst

NIM : 000	NIM : 000
Nama : XXX	Nama : XXX
Peran :	Peran : Anggota Tim

7. Pagination



- *REST API tidak harus 1 (satu) halaman.*
- *Point nomor f pada REST API disesuaikan dengan topik yang telah disetujui oleh Dosen Pengampu.*
- *Format penamaan file ini : NIM_KETUA_TIM_NAMA_KETUA_TIM_ NAMA_MATA_KULIAH_KELAS, contoh : 2033030300007_Andi_Wijaya_PM_TI_4_PAGI_A.*
- *Wajib mengikuti format penaaman file diatas untuk memudahkan proses penilaian.*