

UniPath

Technical Manual

2025

Table of Contents

Social Wellness: Backend	6
Stack	6
Data Model (high-level).....	6
Project Structure & Modularization — Social Wellness Backend (Django)	6
Environment & .env	7
Where DB connection happens	7
URL Routing Overview	9
Auth & Custom User Model.....	10
Cultures App (Dropdown + Posts with Moderation)	12
Chatbox App (Messages, Report, Delete)	16
WeNeed App (Requests, Likes – No Moderation)	17
Filters & Ordering Examples	19
Seeding Cultures (management command pattern)	19
Development Workflow	20
Coding Conventions	20
Academic Wellness: Backend	21
Stack	21
Academic wellness	21
Data Model (high-level).....	21
user_management/models.py	21
planner/models.py	22
wtd/models.py.....	24
box/models.py	25
Project Structure & Modularization - Academic Wellness Backend (Django).....	26
Environment & .env	28
Where DB connection happens.....	29
ecom_project/settings.py (DATABASES).....	29

URL Routing Overview	30
ecom_project/urls.py	30
Auth & Custom User Model.....	31
api_management/urls.py.....	31
user_management/views.py (selected)	32
creates User + Profile(student) and returns JWT tokens.....	32
login by email or username; returns JWT tokens.....	33
WTD App (Posts, Helpful, Moderation)	34
wtd/urls.py.....	34
wtd/permissions.py (helpers)	34
wtd/serializers.py (selected).....	35
wtd/views.py (selected)	36
GET with role-based visibility; POST (seniors) creates pending post.....	37
first-year only; idempotent; increments helpful_count.....	37
Chatbox App (Messages, Report, Delete)	38
box/api_urls.py.....	38
box/permissions.py	38
box/views.py (selected).....	39
box/serializers.py (selected)	41
Planner App (Tasks, Month View, Reminders)	41
planner/api_urls.py	41
planner/permissions.py	42
planner/serializers.py	43
planner/views.py (selected)	44
parse and filter dates if provided.....	44
planner/management/commands/send_deadline_reminders.py (selected)	45
Filters & Ordering Examples.....	46
Development Workflow.....	46
Coding Conventions	46
Academic Wellness: Frontend.....	47
Project Overview	47
Purpose:.....	47

Technologies used:	47
Project Structure and Modularization	47
Project Folder Structure:	47
Login & Registration.....	55
Frontend Implementation:.....	55
User Flow:	55
“What’s The Difference”	57
Frontend Integration:.....	57
Key Components:	57
Two-Week Task Planner (Eisenhower Matrix)	59
Frontend Implementation:.....	59
Features:.....	60
Monthly Academic Planner with Automated Reminders	60
Frontend Implementation:.....	60
Academic Chatboxes.....	61
Frontend Implementation:.....	61
FAQ Page	62
Frontend Implementation:.....	62
Header & Footer Components	63
Frontend Implementation:.....	63
Styling and Theme	63
Color Scheme:	63
Styling Approach:	64
Installation guide	64
Social Wellness: Frontend	64
1. Project Overview	64
2. Folder structure: frontend	65
3. Code Modularization	65
3.1 API Functionality.....	65
3.3 Section Components.....	67
6. Routing Flow	68

7. Development Setup.....	69
9. Security Practices.....	70
10. Developer Notes.....	70
11. Version Control Workflow	71
12.5. Social Chatbox Page.....	77

Social Wellness: Backend

Stack

Django 5 + DRF, SimpleJWT; Apps: user_management, cultures, chatbox, weneed, api_management; DB: PostgreSQL; CORS: django-cors-headers; Filtering/Ordering: django-filter, DRF OrderingFilter

Data Model (high-level)

- user_management.User — user_id (auto string counter), email, name, surname, phone_number, role, is_staff, timestamps. USERNAME_FIELD='email'.
- cultures.Culture — name (unique), description.
- cultures.CulturePost — user FK, culture FK(PROTECT), text_message (≤500), is_approved, is_reviewed, date_created, date_approved, num_of_likes; indexes on status, created_at, culture.
- chatbox.ChatMessage — user FK, message_text (≤255), is_reported, date_created; indexes.
- weneed.NeedRequest — user FK, display_name (optional), category (choices), text (≤250), status (default approved), likes_count, created_at; indexes on status, created_at, user.
- weneed.RequestLike — (request, user) unique pair.

Project Structure & Modularization — Social Wellness Backend (Django)

We keep features in separate Django apps for clarity and maintainability.

```
/ecom_project
├─ ecom_project/
│   ├── settings.py # DB config via decouple (reads .env)
│   └─ urls.py     # mounts /api/ → api_management
├─ api_management/
│   └─ urls.py     # includes user_management, cultures, chatbox, weneed
├─ user_management/ # auth & user model
├─ cultures/       # Culture & CulturePost
├─ chatbox/        # ChatMessage
├─ weneed/         # NeedRequest & RequestLike
└─ requirements.txt
```


The image shows a VS Code editor interface. On the left is the Explorer sidebar showing a project structure. The main editor displays a Python file named 'server.py'. The code in 'server.py' is a Flask application with a single route and a main function. The terminal at the bottom shows the command 'python server.py' being executed. The status bar at the bottom indicates the file is 'server.py' and the editor is in 'Python' mode.

[illegible]

URL Routing Overview

ecom_project/urls.py:

```
path("api/", include("api_management.urls"))
```

[illegible][ecom_project/urls.py](#)

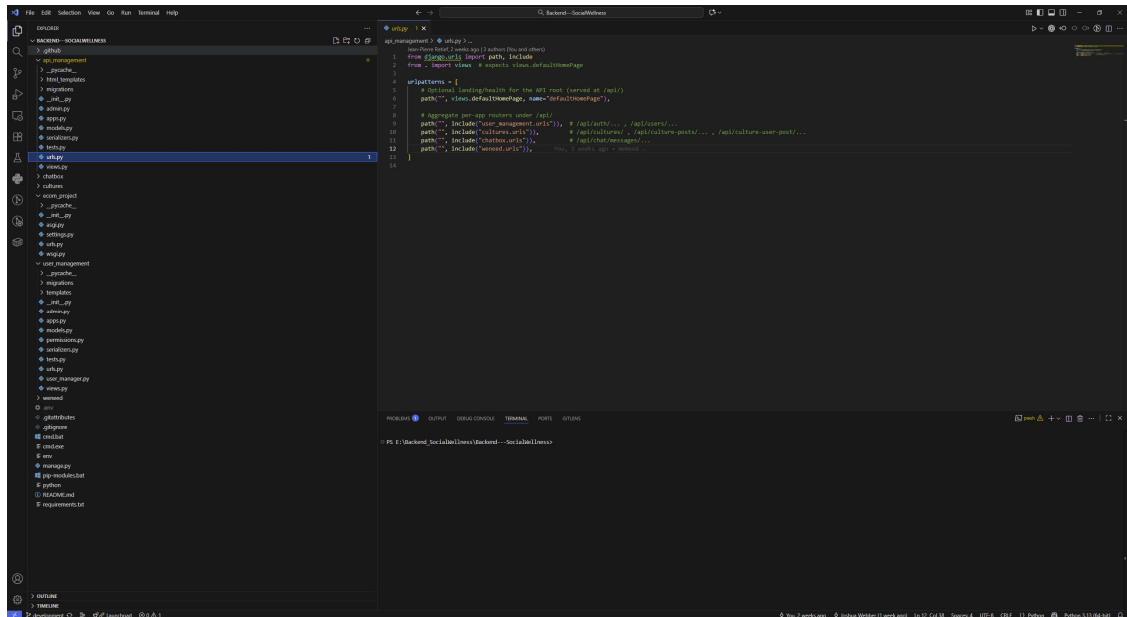
api_management/urls.py:

```
path("auth/", include("user_management.urls"))
```

```
path("", include("cultures.urls"))
```

```
path("", include("chatbox.urls"))
```

```
path("", include("weneed.urls"))
```



api_management/urls.py

Auth & Custom User Model

```
# user_management/models.py (excerpt)
from django.contrib.auth.models import AbstractBaseUser, PermissionsMixin
from django.utils import timezone
from .user_manager import UserManager
from django.db import models
```

```
class User(AbstractBaseUser, PermissionsMixin):
    user_id = models.CharField(max_length=20, unique=True, blank=True, editable=False)
    email = models.EmailField(unique=True)
    name = models.CharField(max_length=100)
    surname = models.CharField(max_length=100)
    phone_number = models.CharField(max_length=20, blank=True)
    role = models.CharField(max_length=20, default='first_year')
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    date_created = models.DateTimeField(default=timezone.now, editable=False)
    date_updated = models.DateTimeField(auto_now=True)
```

```
USERNAME_FIELD = 'email'
REQUIRED_FIELDS = ['name', 'surname']
```

```
objects = UserManager()
```

```
def save(self, *args, **kwargs):
    if not self.user_id:
        last = User.objects.order_by('-date_created').first()
        self.user_id = str(int(getattr(last, 'user_id', '0') or 0) + 1).zfill(6)
```

```
from django.conf import settings
from django.db import models
from django.contrib.auth.models import AbstractUser, PermissionsMixin
from django.contrib.auth import get_user_model

class User(AbstractUser, PermissionsMixin):
    """
    User model
    """
    name = models.CharField(max_length=100, unique=True, blank=True, editable=True)
    email = models.EmailField(unique=True)  # user enters their own email
    password = models.CharField(max_length=100)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    date_created = models.DateTimeField(default=timezone.now, editable=False)
    date_updated = models.DateTimeField(auto_now=True)
    groups = models.ManyToManyField(Group, related_name='users')

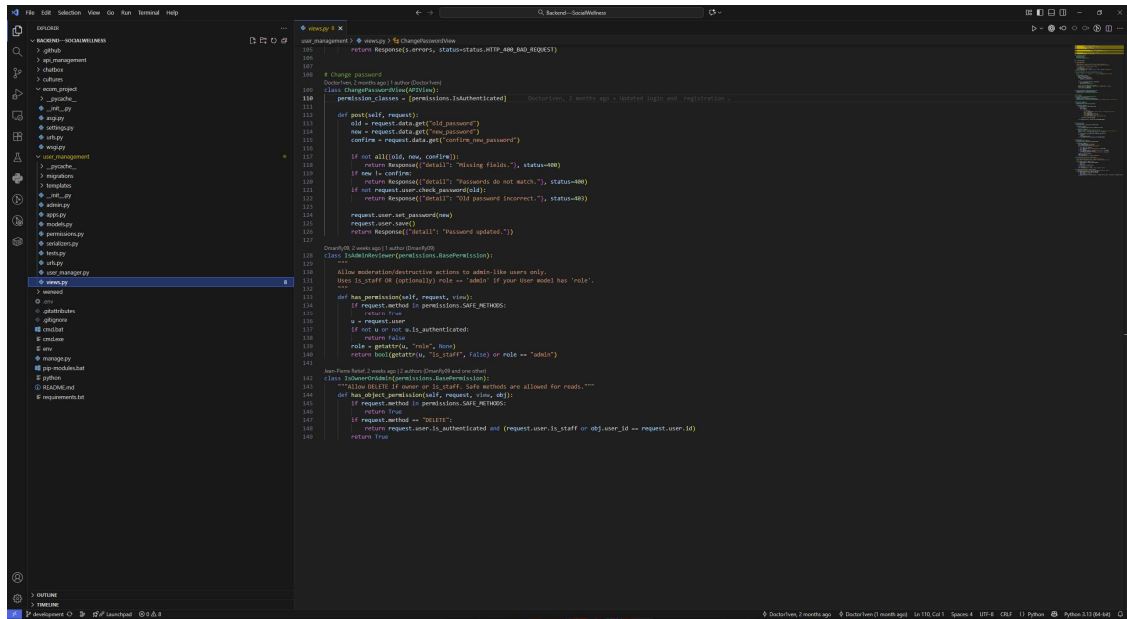
    def save(self, *args, **kwargs):
        """
        Save the user
        """
        if not self.name:
            self.name = 'User'
        if not self.email:
            self.email = 'user@example.com'
        if not self.password:
            self.password = 'password'
        super().save(*args, **kwargs)

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

```
# user_management/views.py (JWT email login serializer)
from rest_framework_simplejwt.serializers import TokenObtainPairSerializer
from django.contrib.auth import get_user_model
```

```
class EmailTokenObtainPairSerializer(TokenObtainPairSerializer):
    def validate(self, attrs):
        email = attrs.get('email'); password = attrs.get('password')
        user = User.objects.get(email__iexact=email)
        attrs['username'] = getattr(user, getattr(User, 'USERNAME_FIELD', 'email'))
        attrs['password'] = password
```

```
return super().validate(attrs)
```



[user_management/views.py](#)

Cultures App (Dropdown + Posts with Moderation)

```
# cultures/models.py
```

```
from django.conf import settings
```

```
from django.db import models
```

```
class Culture(models.Model):
```

```
    name = models.CharField(max_length=100, unique=True)
```

```
    description = models.TextField(blank=True)
```

```
class CulturePost(models.Model):
```

```
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE,
```

```
related_name='culture_posts')
```

```
    culture = models.ForeignKey(Culture, on_delete=models.PROTECT, related_name='posts')
```

```
    text_message = models.CharField(max_length=500)
```

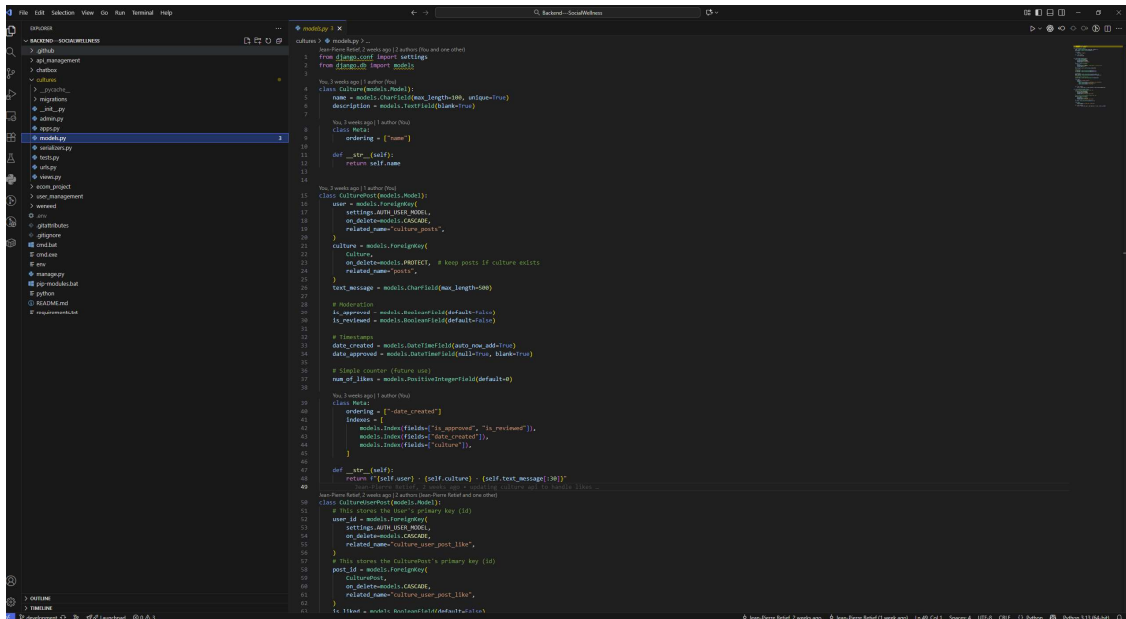
```
    is_approved = models.BooleanField(default=False)
```

```
    is_reviewed = models.BooleanField(default=False)
```

```
    date_created = models.DateTimeField(auto_now_add=True)
```

```
    date_approved = models.DateTimeField(null=True, blank=True)
```

```
num_of_likes = models.PositiveIntegerField(default=0)
```



cultures/models.py

```
# cultures/serializers.py (excerpt)
from rest_framework import serializers
from .models import Culture, CulturePost
from django.utils import timezone
```

```
class CultureSerializer(serializers.ModelSerializer):
    class Meta:
        model = Culture
        fields = ['id', 'name', 'description']
```

```
class CulturePostSerializer(serializers.ModelSerializer):
    user = serializers.HiddenField(default=serializers.CurrentUserDefault())
    class Meta:
        model = CulturePost
        fields =
['id','user','culture','text_message','is_approved','is_reviewed','date_created','date_approved','num_of_likes']
        read_only_fields = ['is_approved','is_reviewed','date_created','date_approved','num_of_likes']
```

```
def create(self, validated_data):
    validated_data['is_approved'] = False
    validated_data['is_reviewed'] = False
    return super().create(validated_data)
```

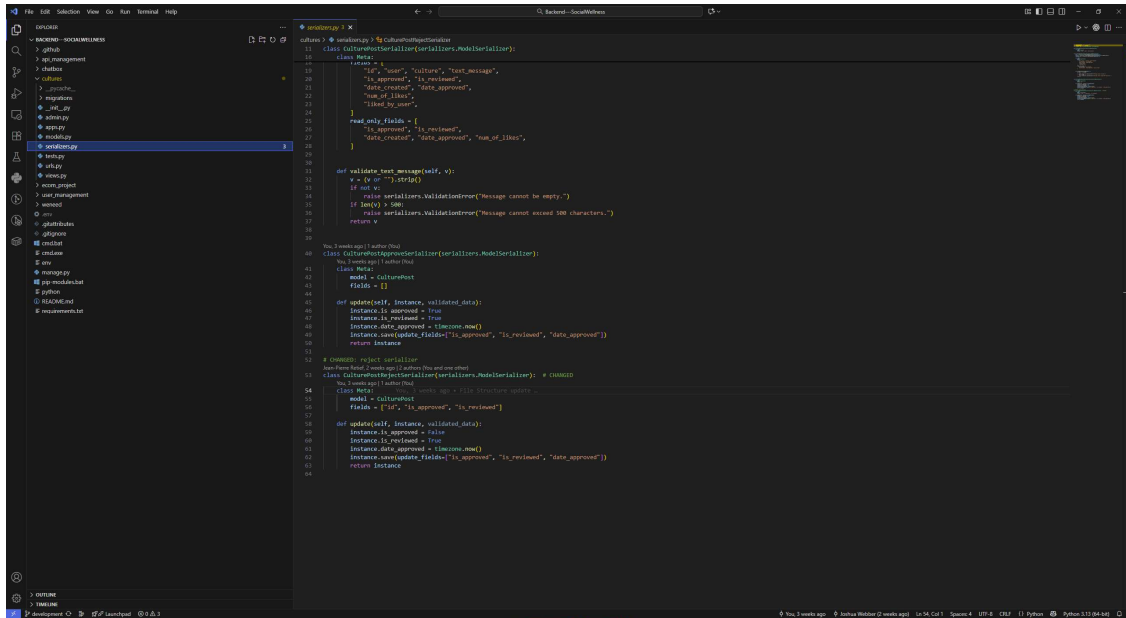
```
class CulturePostApproveSerializer(serializers.ModelSerializer):
    class Meta:
        model = CulturePost
        fields = []
```

```
def update(self, instance, validated_data):
    instance.is_approved = True
```

```

instance.is_reviewed = True
instance.date_approved = timezone.now()
instance.save(update_fields=['is_approved','is_reviewed','date_approved'])
return instance

```



cultures/serializers.py

```

# cultures/views.py (viewsets + reviewer logic)
from rest_framework import viewsets, permissions, mixins, status
from rest_framework.decorators import action
from rest_framework.response import Response
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.filters import OrderingFilter
from rest_framework.exceptions import PermissionDenied
from .models import Culture, CulturePost
from .serializers import CultureSerializer, CulturePostSerializer, CulturePostApproveSerializer

```

```

def is_reviewer(user):
    return bool(getattr(user, 'is_staff', False) or getattr(user, 'role', None) in {'admin','moderator'})

```

```

class CultureViewSet(mixins.ListModelMixin, mixins.RetrieveModelMixin, viewsets.GenericViewSet):
    queryset = Culture.objects.all()
    serializer_class = CultureSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    filter_backends = [OrderingFilter]
    ordering_fields = ['name']

```

```

class CulturePostViewSet(viewsets.ModelViewSet):
    queryset = CulturePost.objects.select_related('user','culture')
    serializer_class = CulturePostSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    filter_backends = [DjangoFilterBackend, OrderingFilter]
    filterset_fields = {'culture':['exact'],'is_approved':['exact']}
    ordering_fields = ['date_created']

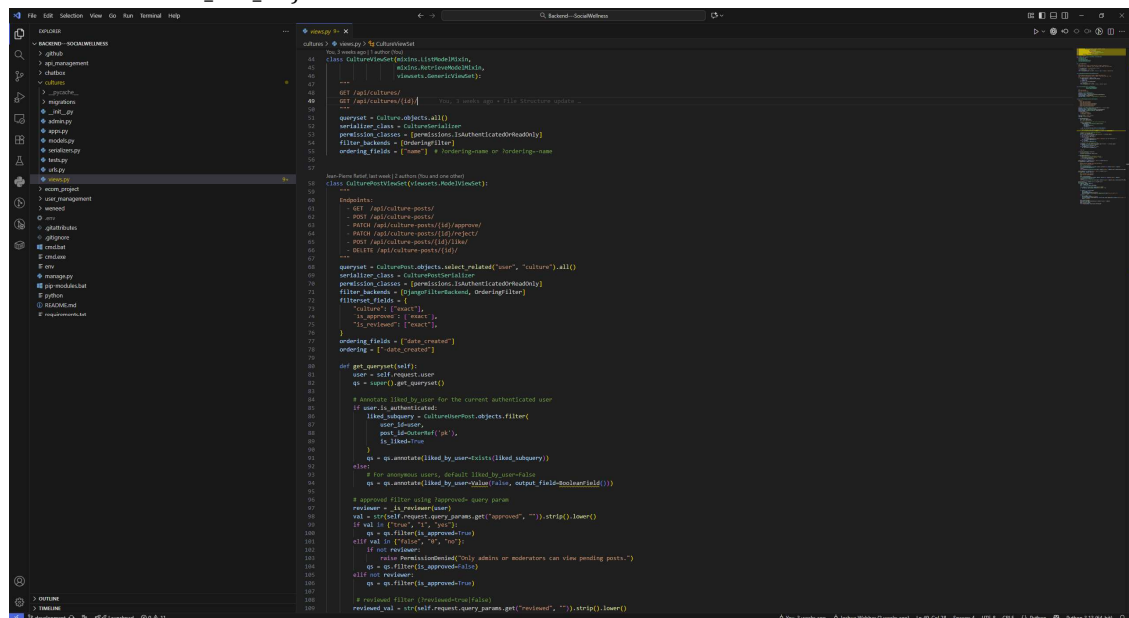
```

```
ordering = ['-date_created']
```

```
def get_queryset(self):
    qs = super().get_queryset()
    reviewer = is_reviewer(getattr(self.request, 'user', None))
    approved = self.request.query_params.get('approved')
    if approved is not None:
        val = approved.strip().lower()
        if val in ('true', '1', 'yes'):
            return qs.filter(is_approved=True)
        if val in ('false', '0', 'no'):
            if not reviewer:
                raise PermissionDenied('Only reviewers may view pending posts.')
            return qs.filter(is_approved=False)
    if not reviewer:
        qs = qs.filter(is_approved=True)
    return qs
```

```
@action(detail=True, methods=['patch', 'post'])
```

```
def approve(self, request, pk=None):
    if not is_reviewer(request.user):
        raise PermissionDenied('Only reviewers may approve.')
    post = self.get_object()
    s = CulturePostApproveSerializer(post, data={}, partial=True, context={'request': request})
    s.is_valid(raise_exception=True)
    s.save()
    return Response(CulturePostSerializer(post, context={'request': request}).data,
                    status=status.HTTP_200_OK)
```



```
class CulturePostViewSet(viewsets.ModelViewSet):
    queryset = CulturePost.objects.all()
    serializer_class = CulturePostSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    filter_backends = [django_filters.OrderingFilter]
    ordering_fields = ['name']

    def get_queryset(self):
        qs = CulturePost.objects.all()
        reviewer = is_reviewer(self.request.user)
        approved = self.request.query_params.get('approved')
        if approved is not None:
            val = approved.strip().lower()
            if val in ('true', '1', 'yes'):
                return qs.filter(is_approved=True)
            if val in ('false', '0', 'no'):
                if not reviewer:
                    raise PermissionDenied('Only reviewers may view pending posts.')
                return qs.filter(is_approved=False)
        if not reviewer:
            qs = qs.filter(is_approved=True)
        return qs

    @action(detail=True, methods=['patch', 'post'])
    def approve(self, request, pk=None):
        if not is_reviewer(request.user):
            raise PermissionDenied('Only reviewers may approve.')
        post = self.get_object()
        s = CulturePostApproveSerializer(post, data={}, partial=True, context={'request': request})
        s.is_valid(raise_exception=True)
        s.save()
        return Response(CulturePostSerializer(post, context={'request': request}).data,
                        status=status.HTTP_200_OK)
```

[cultures/views.py](#)

```
# chatbox/models.py
```

```
from django.db import models
```

```
user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE,
```

```
message_text = models.CharField(max_length=255)
```

```
date_created = models.DateTimeField(auto_now_a
```


The screenshot shows the VS Code Explorer sidebar on the left. The 'EXPLORER' tab is active, and a file named 'models.py' is selected, indicated by a blue highlight and a tab at the top of the sidebar. The main editor area on the right is currently empty.



chatham/vicuna.py

```
from rest_framework import viewsets, permissions, status
```

```
from rest_framework import viewsets, permissions, status
```

```
from rest_framework.decorators import action
```

```
from rest_framework.response import Response
```

```
from django_filters.rest_framework import DjangoFilterBackend
```

```
from rest_framework.filters import OrderingFilter
```

```
from .models import ChatMessage
```

```
from .serializers import ChatMessageSerializer
```

```
class ChatMessageViewSet(viewsets.ModelViewSet):
```

```
queryset = ChatMessage.objects.select_related('user')
```

```
serializer_class = ChatMessageSerializer
```

```
permission_classes = [permissions.IsAuthenticated]
```

```
filter_backends = [DjangoFilterBackend, OrderingFilter]
```

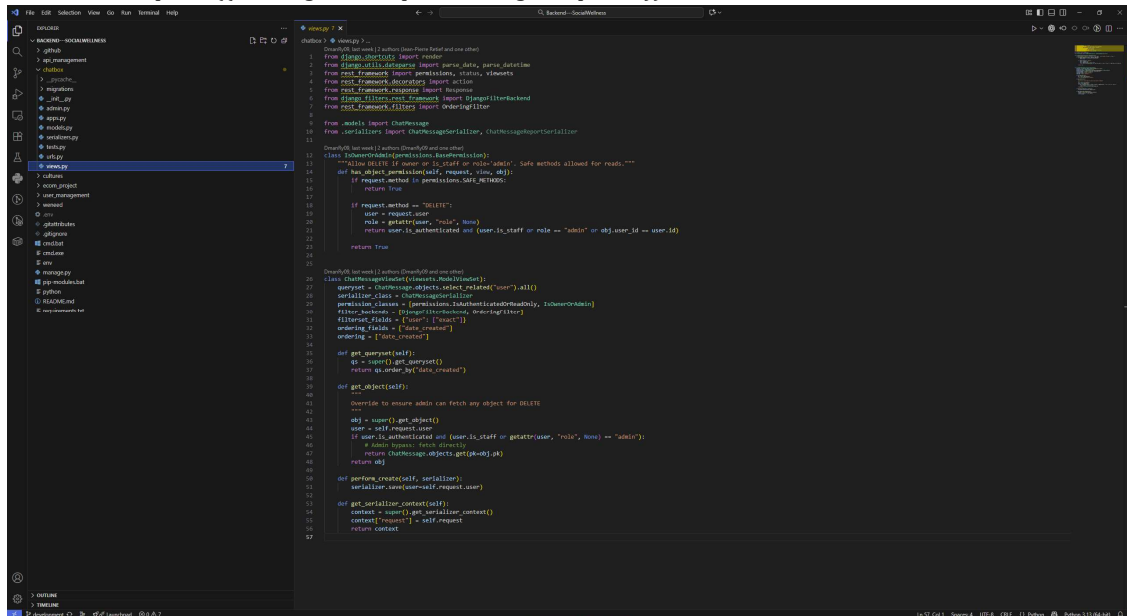
```
filterset_fields = {'user':['exact']}
```

```
ordering_fields = ['date_created']
```

```
ordering = ['-date_created']
```

```
@action(detail=True, methods=['patch'], url_path='report')
```

```
def report(self, request, pk=None):
    msg = self.get_object()
    flag = bool(request.data.get('report', True))
    msg.is_reported = flag
    msg.save(update_fields=['is_reported'])
    return Response({'id': msg.id, 'is_reported': msg.is_reported})
```



[chatbox/views.py](#)

WeNeed App (Requests, Likes – No Moderation)

weneed/models.py

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

class NeedRequest(models.Model):

```
    CATEGORY_CHOICES = [
        ('clubs', 'Clubs'), ('sports', 'Sports'), ('events', 'Events'),
        ('cultural', 'Cultural'), ('facilities', 'Facilities'),
        ('support', 'Support'), ('other', 'Other'),
    ]
```

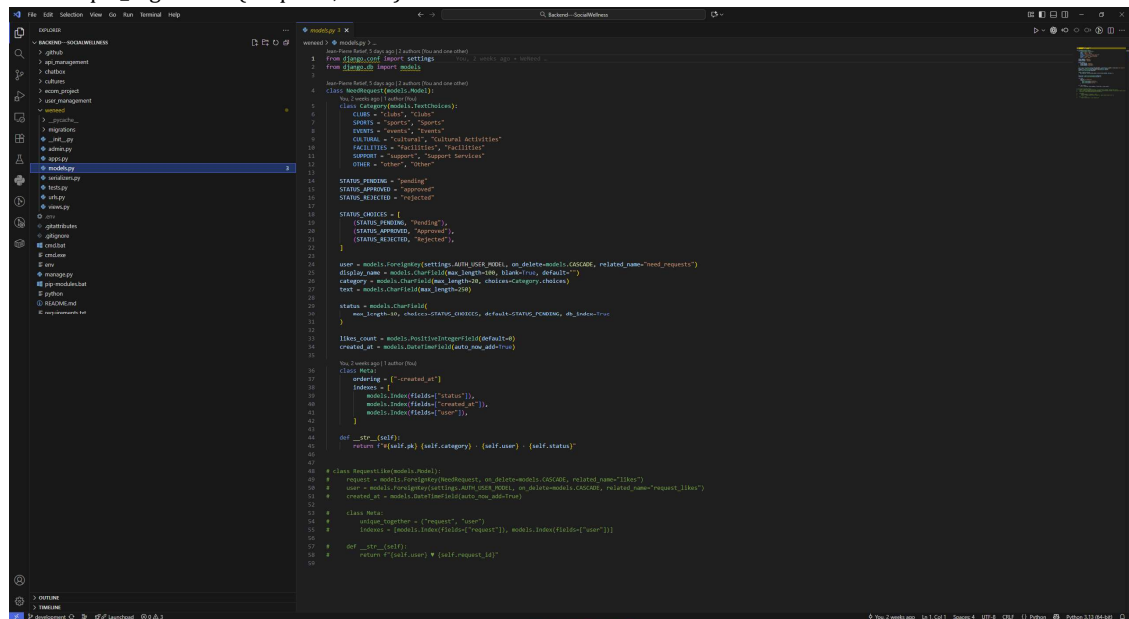
```
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE,
related_name='need_requests')
    display_name = models.CharField(max_length=100, blank=True, default='')
    category = models.CharField(max_length=20, choices=CATEGORY_CHOICES)
    text = models.CharField(max_length=250)
    status = models.CharField(max_length=20, default='approved')
    likes_count = models.PositiveIntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
```

class RequestLike(models.Model):

```
    request = models.ForeignKey(NeedRequest, on_delete=models.CASCADE, related_name='likes')
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE,
related_name='request_likes')
```

class Meta:

unique_together = ('request','user')



weneed/models.py

weneed/views.py (likes + mine)

from rest_framework import viewsets, permissions, status

from rest_framework.decorators import action

from rest_framework.response import Response

from django_filters.rest_framework import DjangoFilterBackend

from rest_framework.filters import OrderingFilter

from .models import NeedRequest, RequestLike

from .serializers import NeedRequestCreateSerializer, NeedRequestPublicSerializer

class NeedRequestViewSet(viewsets.ModelViewSet):

queryset = NeedRequest.objects.select_related('user')

permission_classes = [permissions.IsAuthenticatedOrReadOnly]

filter_backends = [DjangoFilterBackend, OrderingFilter]

filterset_fields = {'category': ['exact']}

ordering_fields = ['created_at', 'likes_count']

ordering = ['-created_at']

def get_serializer_class(self):

return NeedRequestCreateSerializer if self.action in ('create', 'update', 'partial_update') else

NeedRequestPublicSerializer

def perform_create(self, serializer):

serializer.save(user=self.request.user, status='approved')

@action(detail=False, methods=['get'], url_path='mine', permission_classes=[permissions.IsAuthenticated])

def mine(self, request):

qs = self.get_queryset().filter(user=request.user)

s = NeedRequestPublicSerializer(qs, many=True, context={'request': request})

return Response(s.data)

```

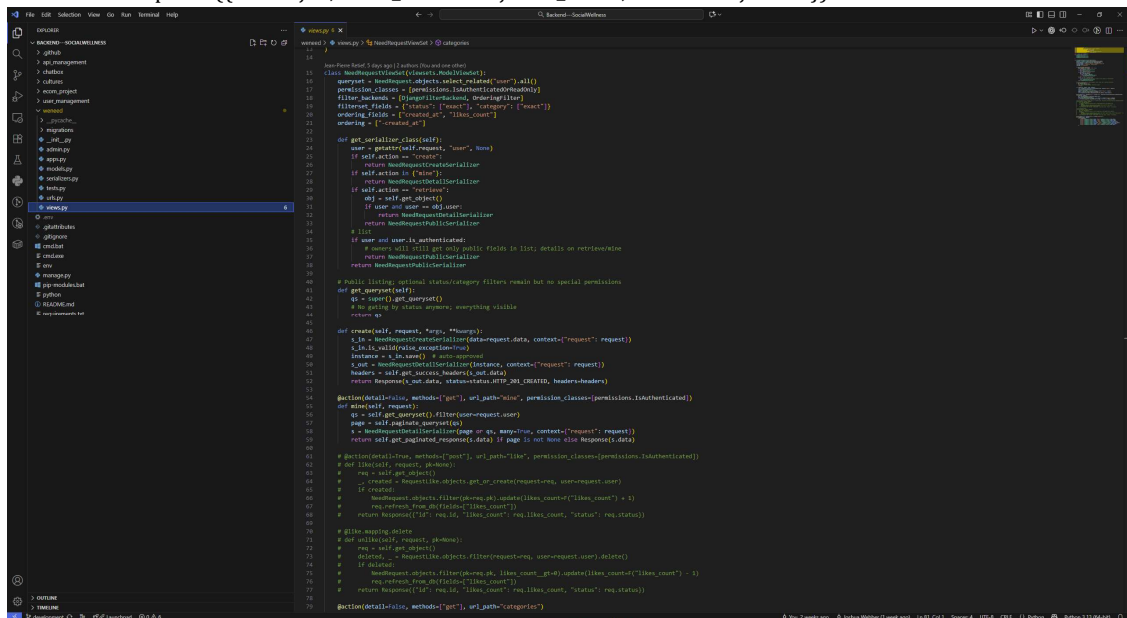
@action(detail=True, methods=['post'], url_path='like', permission_classes=[permissions.IsAuthenticated])
def like(self, request, pk=None):
    obj = self.get_object()
    like, created = RequestLike.objects.get_or_create(request=obj, user=request.user)
    if created:
        obj.likes_count = (obj.likes_count or 0) + 1
        obj.save(update_fields=['likes_count'])
    return Response({'id': obj.id, 'likes_count': obj.likes_count, 'status': obj.status})

```

```

@action(detail=True, methods=['delete'], url_path='like', permission_classes=[permissions.IsAuthenticated])
def unlike(self, request, pk=None):
    obj = self.get_object()
    deleted, _ = RequestLike.objects.filter(request=obj, user=request.user).delete()
    if deleted:
        obj.likes_count = max(0, (obj.likes_count or 0) - 1)
        obj.save(update_fields=['likes_count'])
    return Response({'id': obj.id, 'likes_count': obj.likes_count, 'status': obj.status})

```



weneed/views.py

Filters & Ordering Examples

GET /api/culture-posts/?approved=true&ordering=-date_created

GET /api/chat/messages/?user=7

GET /api/requests/?category=events&ordering=-likes_count

Seeding Cultures (management command pattern)

```

# cultures/management/commands/seed_cultures.py
from django.core.management.base import BaseCommand
from cultures.models import Culture

```

```

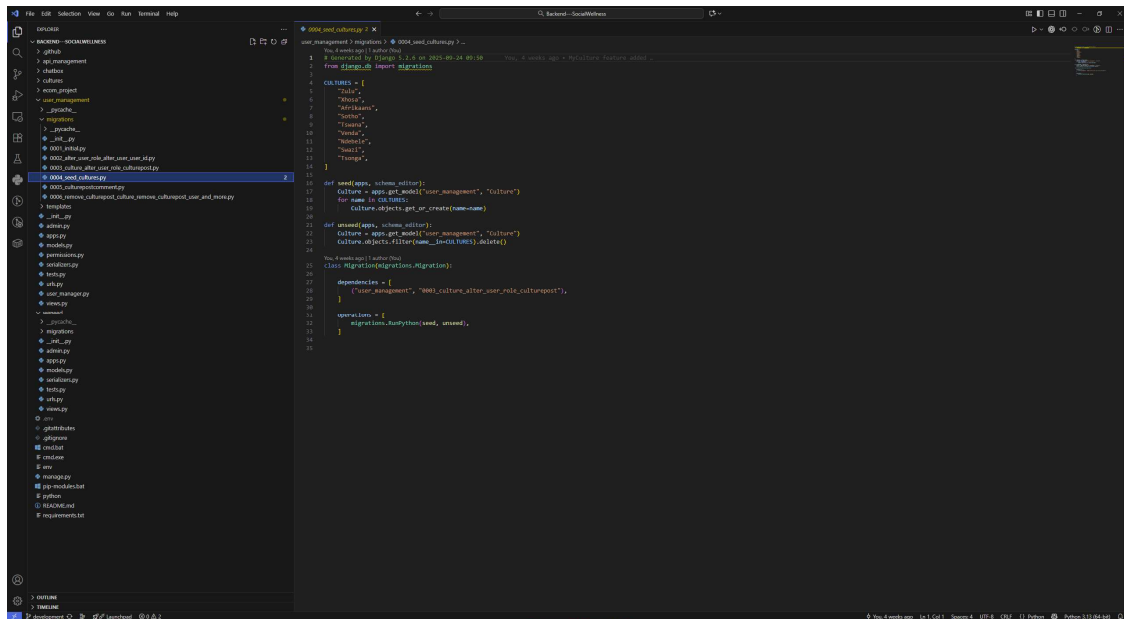
DEFAULTS = ['Zulu', 'Xhosa', 'Afrikaans', 'Sotho', 'Tswana', 'Venda', 'Ndebele', 'Swazi', 'Tsonga']

```

```

class Command(BaseCommand):
    help = 'Seed default cultures'
    def handle(self, *args, **kwargs):
        created = 0
        for name in DEFAULTS:
            _, was_created = Culture.objects.get_or_create(name=name)
            created += int(was_created)
        self.stdout.write(self.style.SUCCESS(f'Seeded {created} cultures.'))

```



Development Workflow

- 1) Copy .env.example → .env and fill in secrets & DB.
- 2) pip install -r requirements.txt
- 3) python manage.py makemigrations user_management cultures chatbox wenneed
- 4) python manage.py migrate
- 5) Seed cultures (9 defaults) if needed.
- 6) python manage.py createsuperuser
- 7) python manage.py runserver

Coding Conventions

- One Django app per feature.
- Keep serializers/viewsets within each app.
- Use django-filter + OrderingFilter for list endpoints.
- Frontend: put API logic in apiComponents; pages are mostly declarative UI.

Academic Wellness: Backend

Stack

- Django 5, Django REST Framework, PostgreSQL, JWT (SimpleJWT), WhiteNoise.
- Apps in this repo: `planner`, `wtd`, `box`, `user\_management`, `api\_management`.
- Python 3.11+ recommended.

Academic wellness

- Scope in this module
- Academic Planner: personal tasks, priorities, month view, optional reminders.
- Peer Academic Advice (WTD): seniors share tips; moderators approve; first-years mark helpful.
- Chat (Box): simple room-based messages for collaboration.
- Non-goals: Social/cultural content (covered by Social Wellness team).

Data Model (high-level)

- User/Profile
- Django `auth.User` with `Profile(student\_type, full\_name)` in `user\_management/models.py:1`.

user_management/models.py

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

```
class Profile(models.Model):
class Role(models.TextChoices):
    STUDENT = "student", "Student"
    MODERATOR = "moderator", "Moderator"
```

```
class StudentType(models.TextChoices):
    FIRST_YEAR = "first_year", "First Year"
    SENIOR = "senior", "Senior"
```

```

user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name="profile")

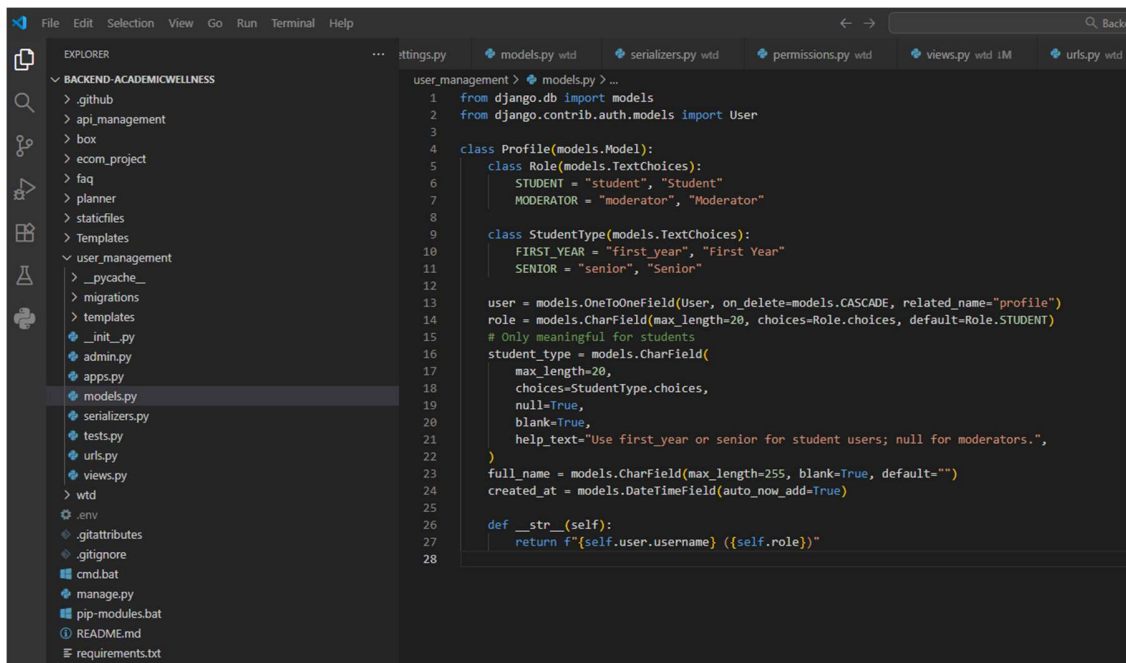
role = models.CharField(max_length=20, choices=Role.choices,
default=Role.STUDENT)

student_type = models.CharField(max_length=20, choices=StudentType.choices,
null=True, blank=True)

full_name = models.CharField(max_length=255, blank=True, default="")

created_at = models.DateTimeField(auto_now_add=True)

```



[user_management/models.py](#)

[planner/models.py](#)

```

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

```

```
User = settings.AUTH_USER_MODEL
```

```
class Task(models.Model):
```

```

class Priority(models.TextChoices):
    LOW = "low", "Low"

    MEDIUM = "medium", "Medium"

    HIGH = "high", "High"

owner = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="tasks")

date = models.DateField(help_text="Calendar day the task is scheduled for")

title = models.CharField(max_length=255)

description = models.TextField(blank=True)

priority = models.CharField(max_length=10, choices=Priority.choices,
default=Priority.MEDIUM)

allow_reminders = models.BooleanField(default=False)

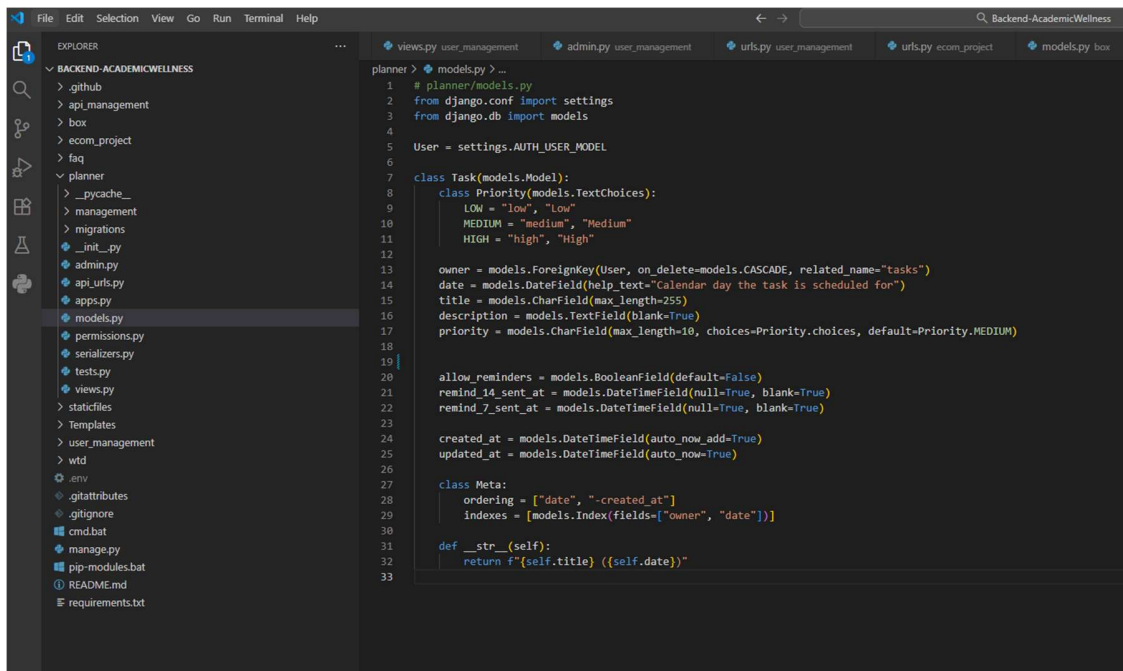
remind_14_sent_at = models.DateTimeField(null=True, blank=True)

remind_7_sent_at = models.DateTimeField(null=True, blank=True)

created_at = models.DateTimeField(auto_now_add=True)

updated_at = models.DateTimeField(auto_now=True)

```



[planner/models.py](#)

wtd/models.py

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

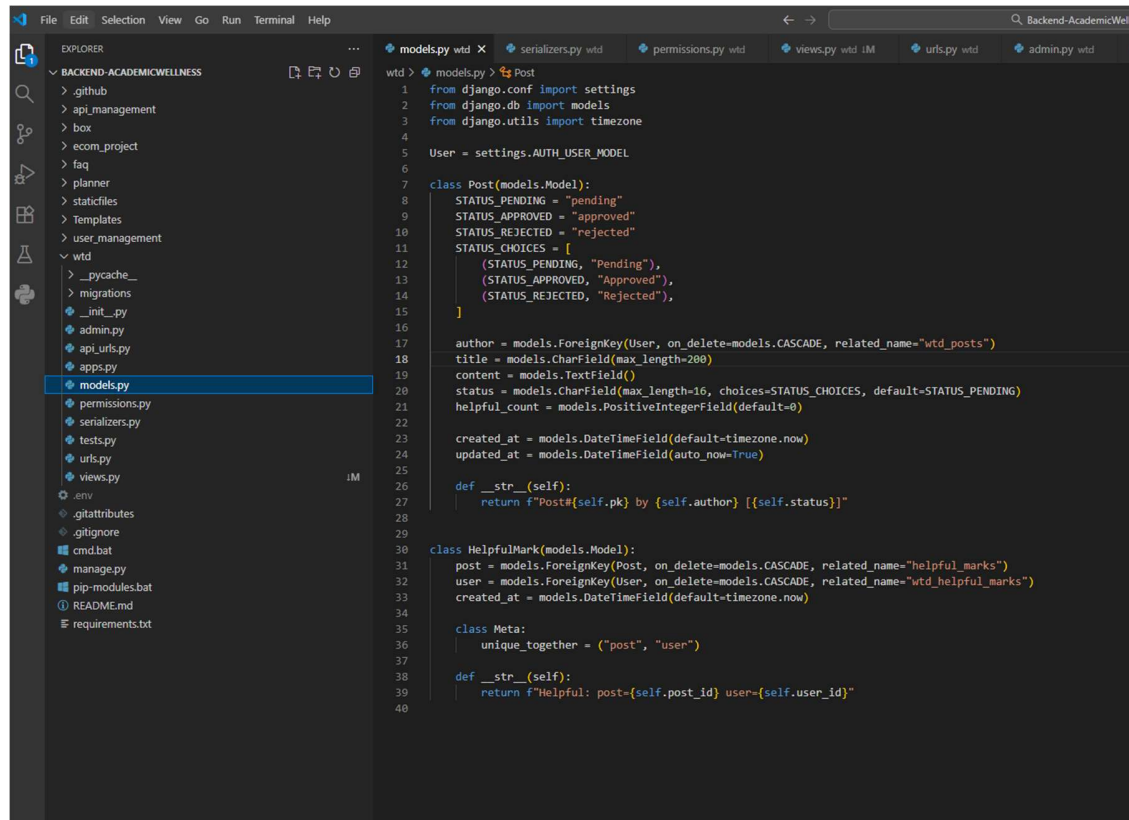
```
User = settings.AUTH_USER_MODEL
```

[illegible]

```
created_at = models.DateTimeField(default=timezone.now)
```

```
class Meta:
```

```
unique_together = ("post", "user")
```



[wtd/models.py](#)

box/models.py

```
from django.conf import settings
```

```
from django.db import models
```

```
class ChatMessage(models.Model):
```

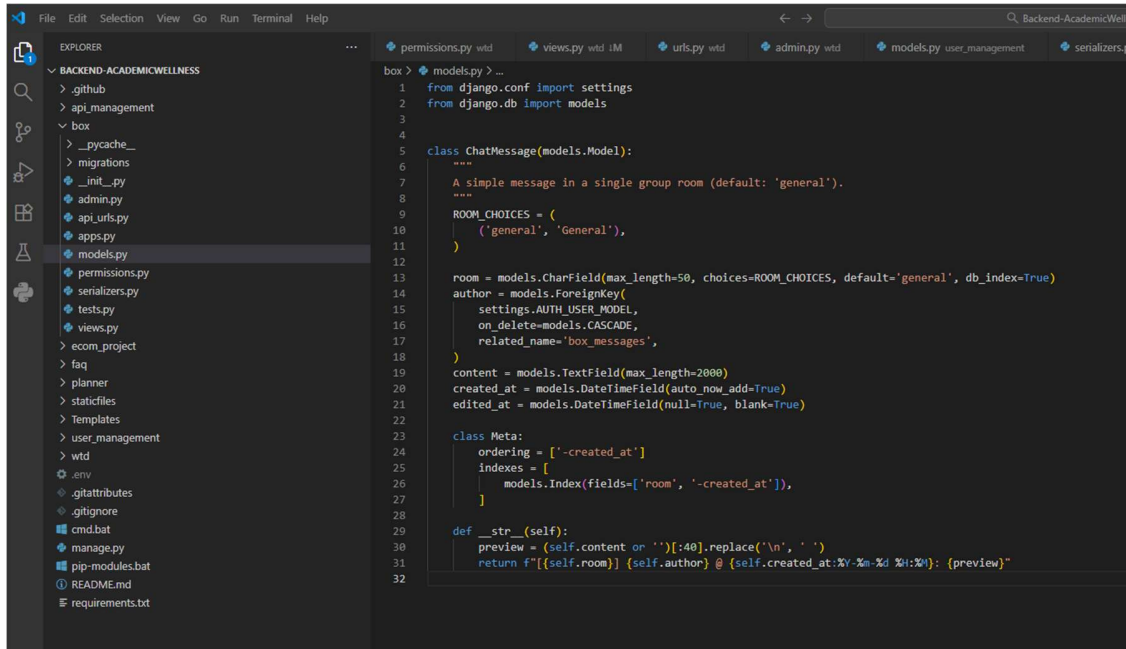
```
ROOM_CHOICES = (('general', 'General'),)
```

```
room = models.CharField(max_length=50, choices=ROOM_CHOICES,  
default='general', db_index=True)
```

```
author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE,  
related_name='box_messages')
```

```
content = models.TextField(max_length=2000)
```

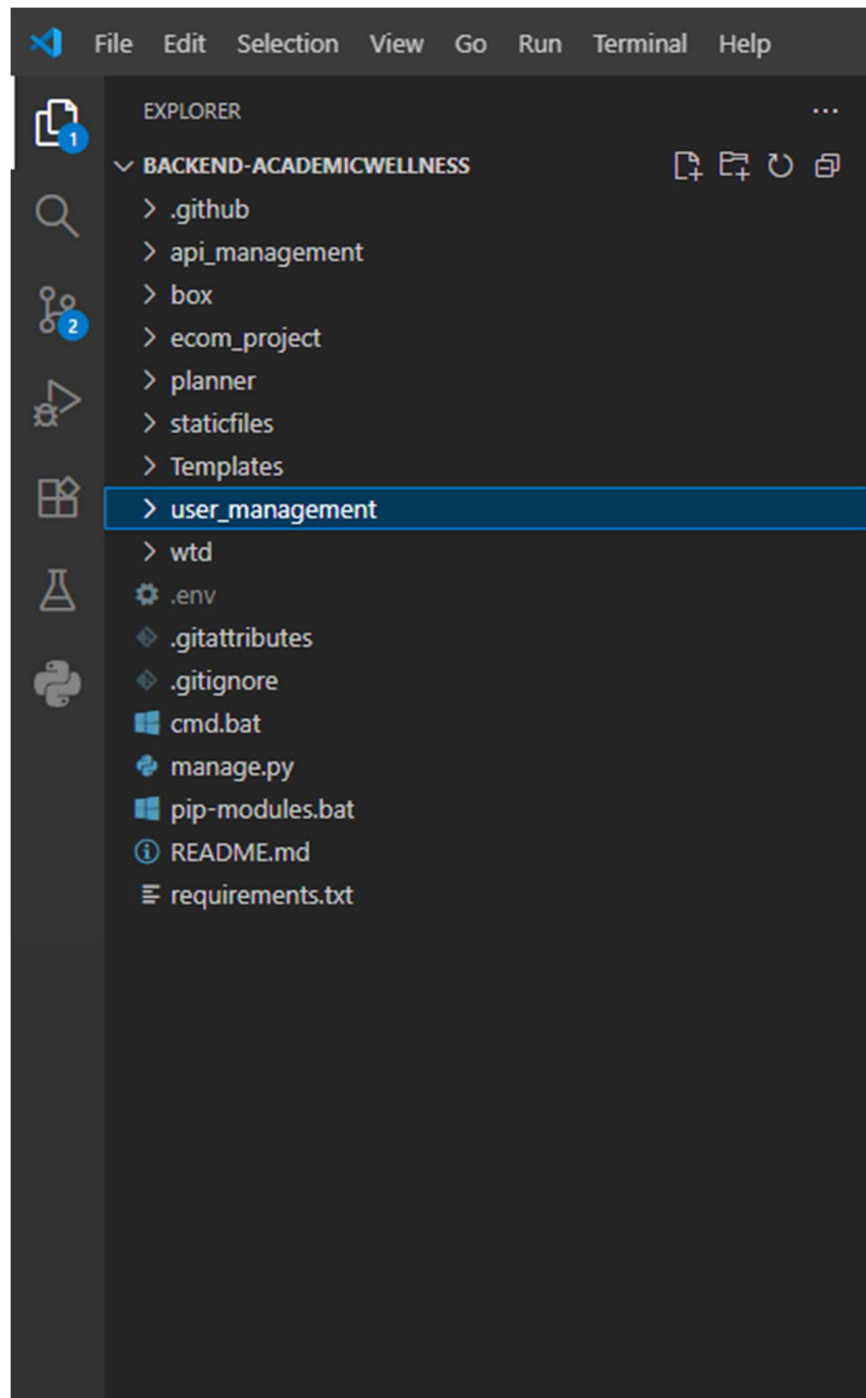
```
created_at = models.DateTimeField(auto_now_add=True)
edited_at = models.DateTimeField(null=True, blank=True)
```



[box/models.py](#)

Project Structure & Modularization - Academic Wellness Backend (Django)

- Settings and URLs
- Settings: `ecom\_project/settings.py:1` (DB via decouple; JWT; CORS; static).
- Root URLs: `ecom\_project/urls.py:1` mounting API modules.
- API entry
- Base API: `api\_management/urls.py:1` with health/auth routes.
- Modules
- Planner API: `planner/api\_urls.py:1`, `planner/views.py:1`.
- WTD API: `wtd/urls.py:1`, `wtd/views.py:1`.
- Chat API: `box/api\_urls.py:1`, `box/views.py:1`.

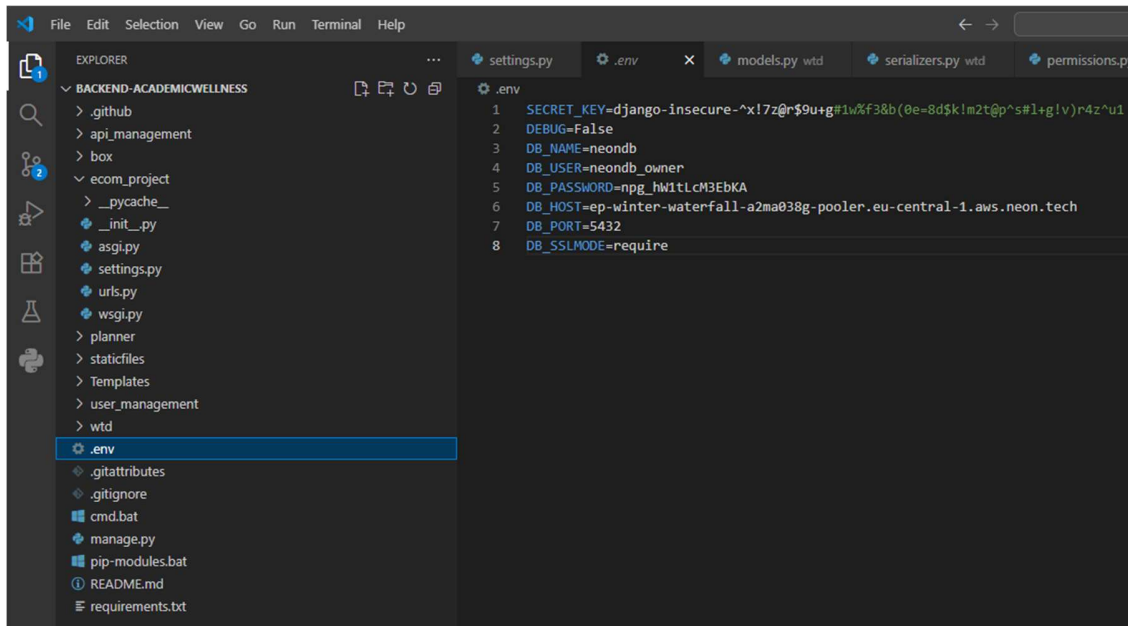


file structure

Environment & .env

Create `.env` at repo root with:

```
# .env (example)
SECRET_KEY="your-django-secret"
DEBUG=True
DB_NAME="your_db_name"
DB_USER="your_db_user"
DB_PASSWORD="your_db_password"
DB_HOST="your_db_host"
DB_PORT="5432"
DB_SSLMODE="require"
FRONTEND_ORIGIN="https://frontend-socialwellness.onrender.com"
```

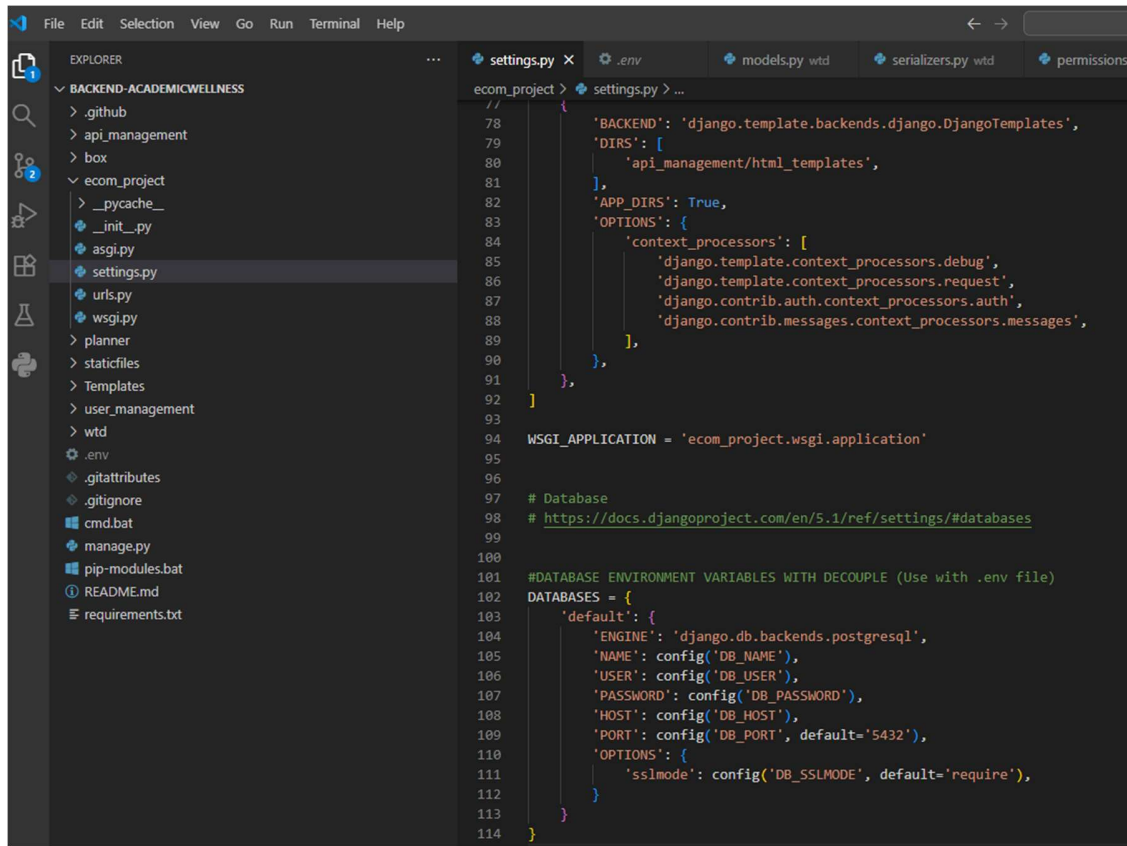


.env file

Where DB connection happens

ecom_project/settings.py (DATABASES)

```
DATABASES = {  
  
    'default': {  
  
        'ENGINE': 'django.db.backends.postgresql',  
  
        'NAME': config('DB_NAME'),  
  
        'USER': config('DB_USER'),  
  
        'PASSWORD': config('DB_PASSWORD'),  
  
        'HOST': config('DB_HOST'),  
  
        'PORT': config('DB_PORT', default='5432'),  
  
        'OPTIONS': {  
  
            'sslmode': config('DB_SSLMODE', default='require'),  
  
        }  
  
    }  
}
```



```
//
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [
        'api_management/html_templates',
    ],
    '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 = 'ecom_project.wsgi.application'

# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

#DATABASE ENVIRONMENT VARIABLES WITH DECOUPLE (Use with .env file)
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST'),
        'PORT': config('DB_PORT', default='5432'),
        'OPTIONS': {
            'sslmode': config('DB_SSLMODE', default='require'),
        }
    }
}
```

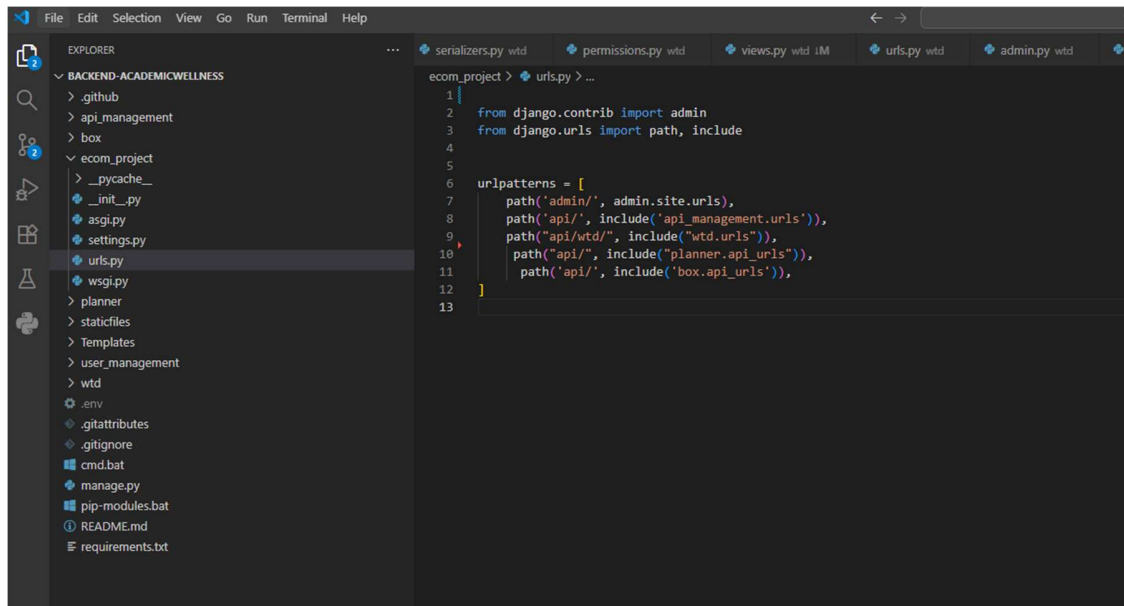
ecom_project/settings.py

URL Routing Overview

ecom_project/urls.py

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

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api_management.urls')),
    path("api/wtd/", include("wtd.urls")),
    path("api/", include("planner.api_urls")),
    path('api/', include('box.api_urls')),
]
```



[ecom_project/urls.py](#)

Auth & Custom User Model

- Auth: JWT via SimpleJWT; default permission is authenticated (settings).
- Users: built-in `auth.User` plus `Profile` for `student\_type`.
- Endpoints in `api\_management/urls.py:1` map to `user\_management/views.py:1`.

[api_management/urls.py](#)

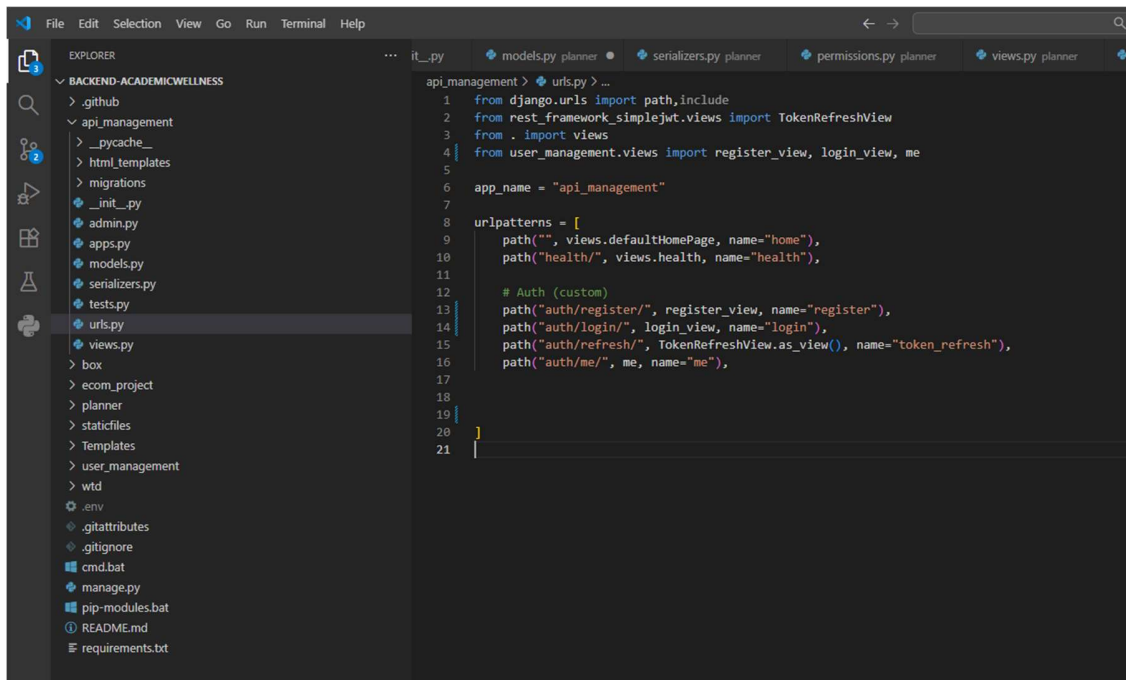
```
from django.urls import path,include
from rest_framework_simplejwt.views import TokenRefreshView
from . import views
from user_management.views import register_view, login_view, me

urlpatterns = [
```

```

path("", views.defaultHomePage, name="home"),
path("health/", views.health, name="health"),
path("auth/register/", register_view, name="register"),
path("auth/login/", login_view, name="login"),
path("auth/refresh/", TokenRefreshView.as_view(), name="token_refresh"),
path("auth/me/", me, name="me"),
]

```



[api_management/urls.py](#)

[user_management/views.py \(selected\)](#)

```
@api_view(["POST"])
```

```
@permission_classes([AllowAny])
```

```
@transaction.atomic
```

```
def register_view(request):
```

creates User + Profile(student) and returns JWT tokens

...

```
@api_view(["POST"])
```

```
@permission_classes([AllowAny])
```

```
def login_view(request):
```

login by email or username; returns JWT tokens

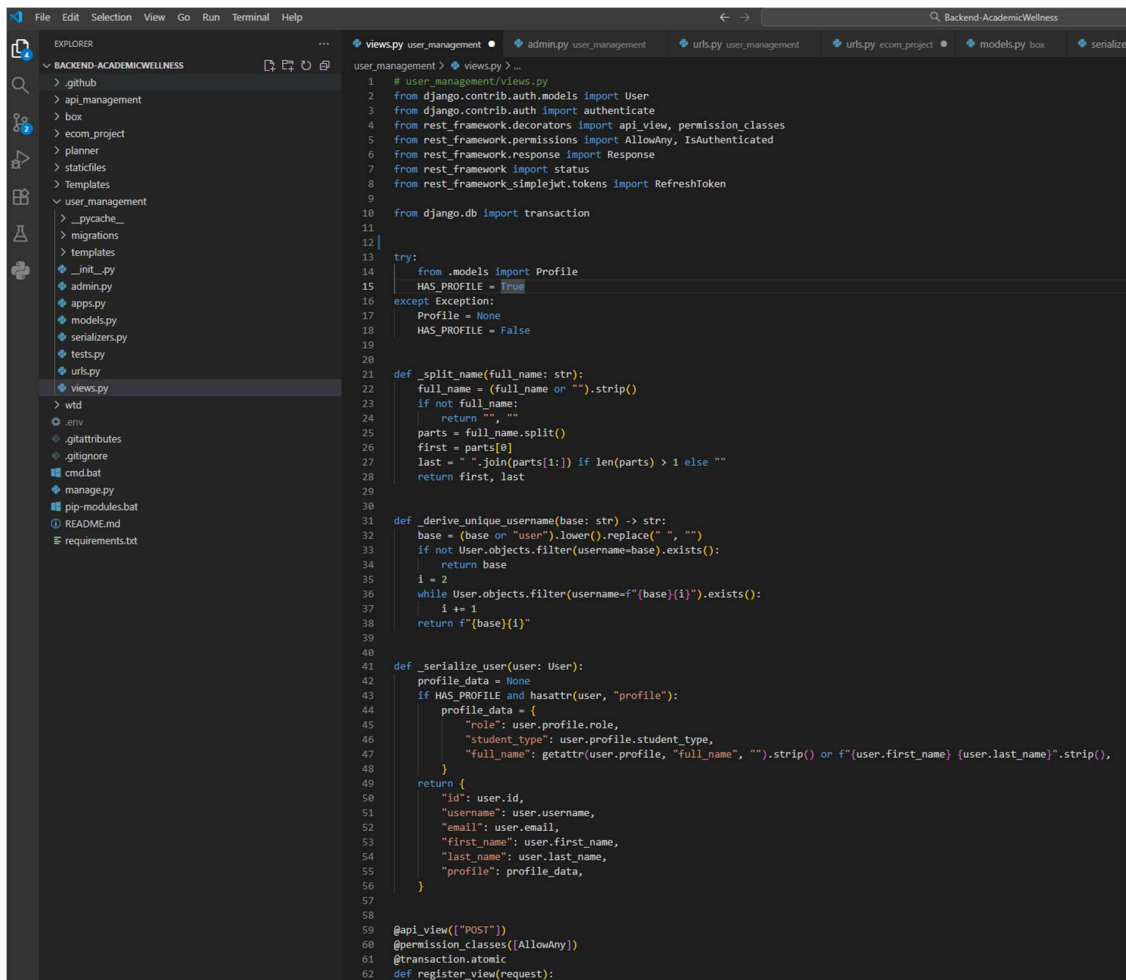
...

```
@api_view(["GET"])
```

```
@permission_classes([IsAuthenticated])
```

```
def me(request):
```

```
return Response(_serialize_user(request.user), status=200)
```



```
1 # user_management/views.py
2 from django.contrib.auth.models import User
3 from django.contrib.auth import authenticate
4 from rest_framework.decorators import api_view, permission_classes
5 from rest_framework.permissions import AllowAny, IsAuthenticated
6 from rest_framework.response import Response
7 from rest_framework import status
8 from rest_framework_simplejwt.tokens import RefreshToken
9
10 from django.db import transaction
11
12
13 try:
14     from .models import Profile
15     HAS_PROFILE = True
16 except Exception:
17     Profile = None
18     HAS_PROFILE = False
19
20
21 def _split_name(full_name: str):
22     full_name = (full_name or "").strip()
23     if not full_name:
24         return "", ""
25     parts = full_name.split()
26     first = parts[0]
27     last = " ".join(parts[1:]) if len(parts) > 1 else ""
28     return first, last
29
30
31 def _derive_unique_username(base: str) -> str:
32     base = (base or "user").lower().replace(" ", "")
33     if not User.objects.filter(username=base).exists():
34         return base
35     i = 2
36     while User.objects.filter(username=f"{base}{i}").exists():
37         i += 1
38     return f"{base}{i}"
39
40
41 def _serialize_user(user: User):
42     profile_data = None
43     if HAS_PROFILE and hasattr(user, "profile"):
44         profile_data = {
45             "role": user.profile.role,
46             "student_type": user.profile.student_type,
47             "full_name": getattr(user.profile, "full_name", "").strip() or f"{user.first_name} {user.last_name}".strip(),
48         }
49     return {
50         "id": user.id,
51         "username": user.username,
52         "email": user.email,
53         "first_name": user.first_name,
54         "last_name": user.last_name,
55         "profile": profile_data,
56     }
57
58
59 @api_view(["POST"])
60 @permission_classes([AllowAny])
61 @transaction.atomic
62 def register_view(request):
```

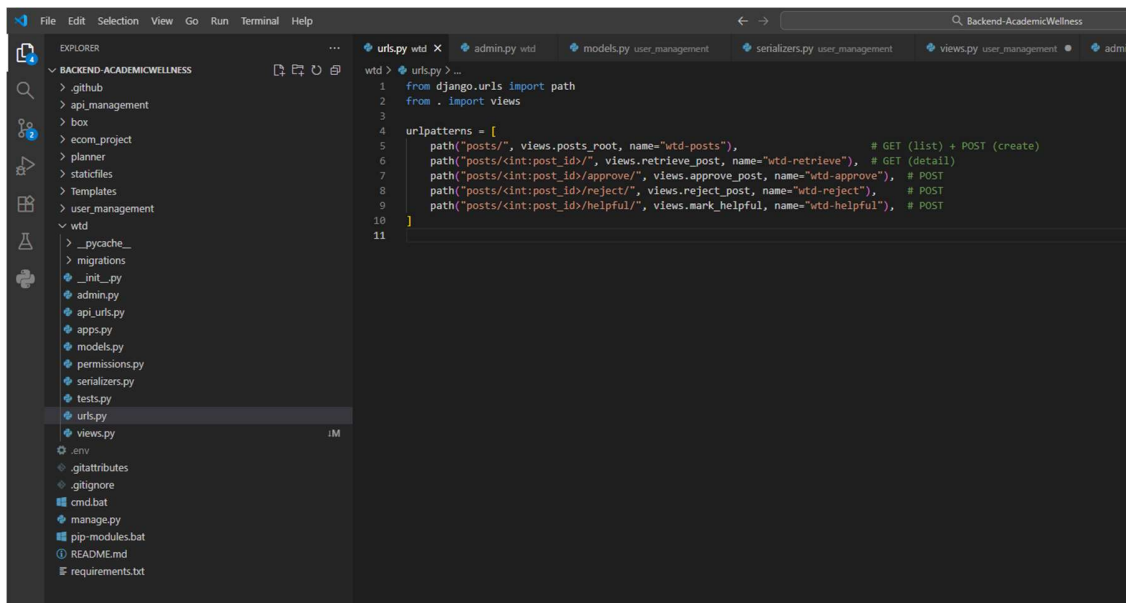
login/ user_managemnt/views.py

WTD App (Posts, Helpful, Moderation)

wtd/urls.py

```
from django.urls import path
from . import views
```

```
urlpatterns = [
    path("posts/", views.posts_root, name="wtd-posts"),
    path("posts/<int:post_id>/", views.retrieve_post, name="wtd-retrieve"),
    path("posts/<int:post_id>/approve/", views.approve_post, name="wtd-approve"),
    path("posts/<int:post_id>/reject/", views.reject_post, name="wtd-reject"),
    path("posts/<int:post_id>/helpful/", views.mark_helpful, name="wtd-helpful"),
]
```



wtd/urls.py

wtd/permissions.py (helpers)

```
def is_moderator(user):
    if getattr(user, "is_superuser", False) or getattr(user, "is_staff", False):
```

```
return True
```

```
prof = getattr(user, "profile", None)
```

```
return bool(prof and prof.role == "moderator")
```

```
def is_first_year(user):
```

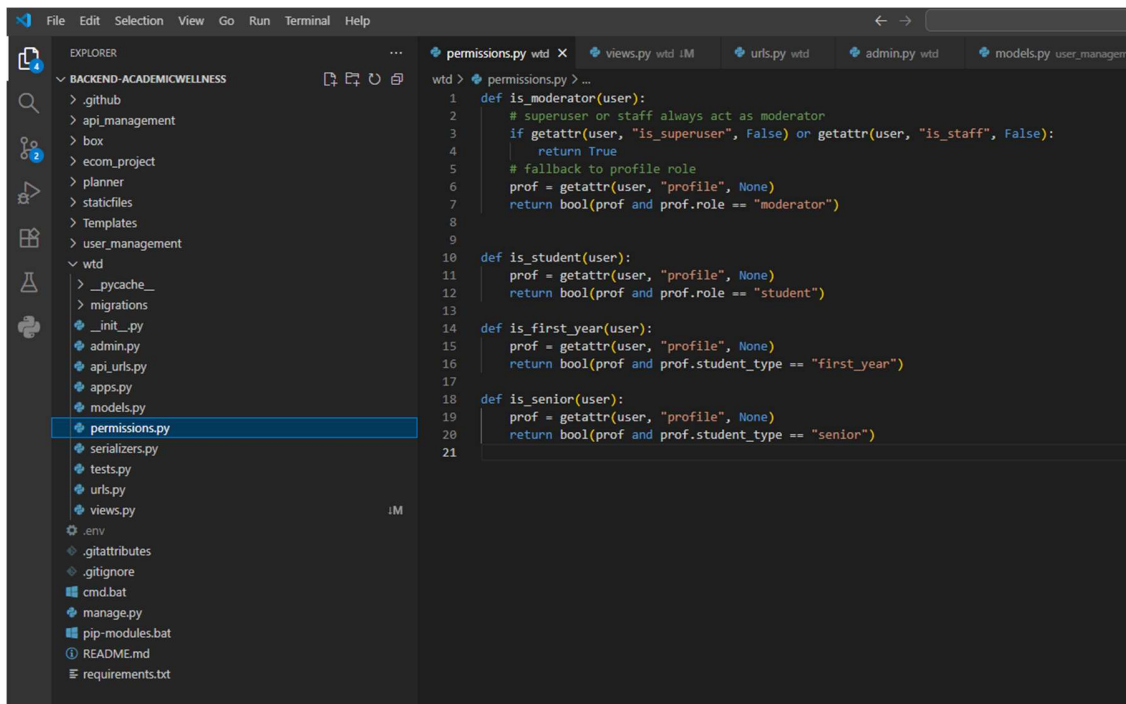
```
prof = getattr(user, "profile", None)
```

```
return bool(prof and prof.student_type == "first_year")
```

```
def is_senior(user):
```

```
prof = getattr(user, "profile", None)
```

```
return bool(prof and prof.student_type == "senior")
```



[wtd/permissions.py](#)

[wtd/serializers.py \(selected\)](#)

```
class PostCreateSerializer(serializers.ModelSerializer):
```

```
class Meta:
```

```
model = Post
```

```
fields = ["id","title","content"]
```

```
class PostDetailSerializer(serializers.ModelSerializer):
```

```
author_username = serializers.CharField(source="author.username", read_only=True)
```

```
class Meta:
```

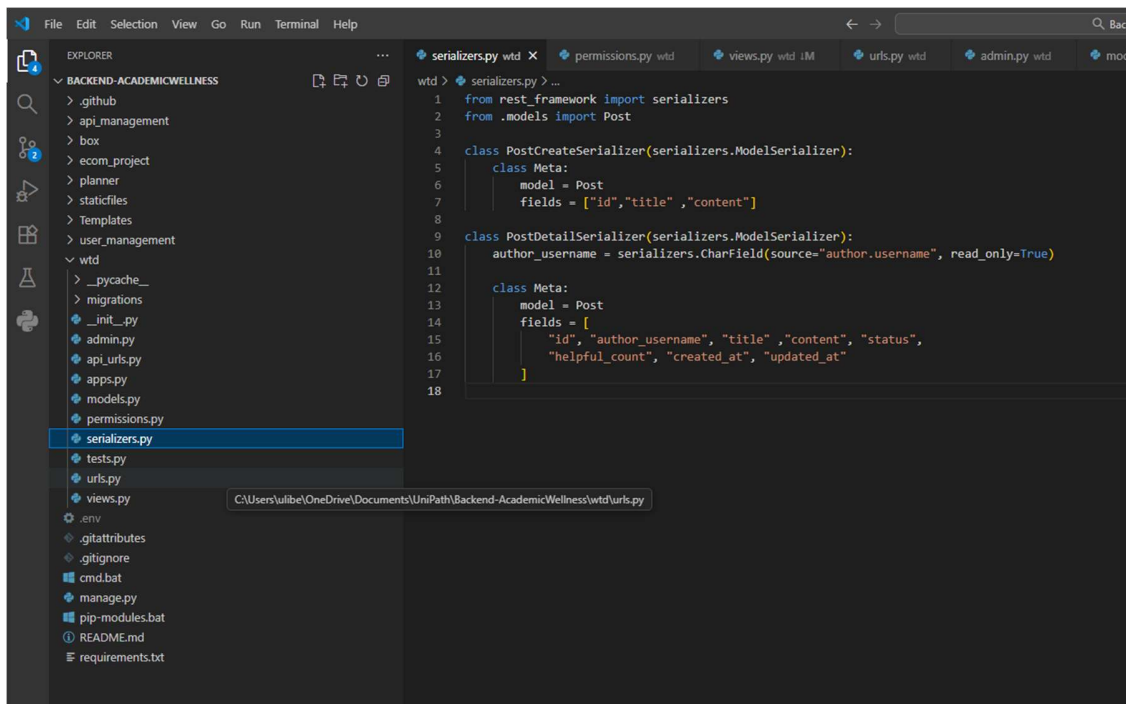
```
model = Post
```

```
fields = [
```

```
"id", "author_username", "title", "content", "status",
```

```
"helpful_count", "created_at", "updated_at"
```

```
]
```



[wtd/serializers.py](#)

[wtd/views.py \(selected\)](#)

```
@api_view(["GET", "POST"])
```

```
@permission_classes([IsAuthenticated])
```

```
def posts_root(request):
```

GET with role-based visibility; POST (seniors) creates pending post

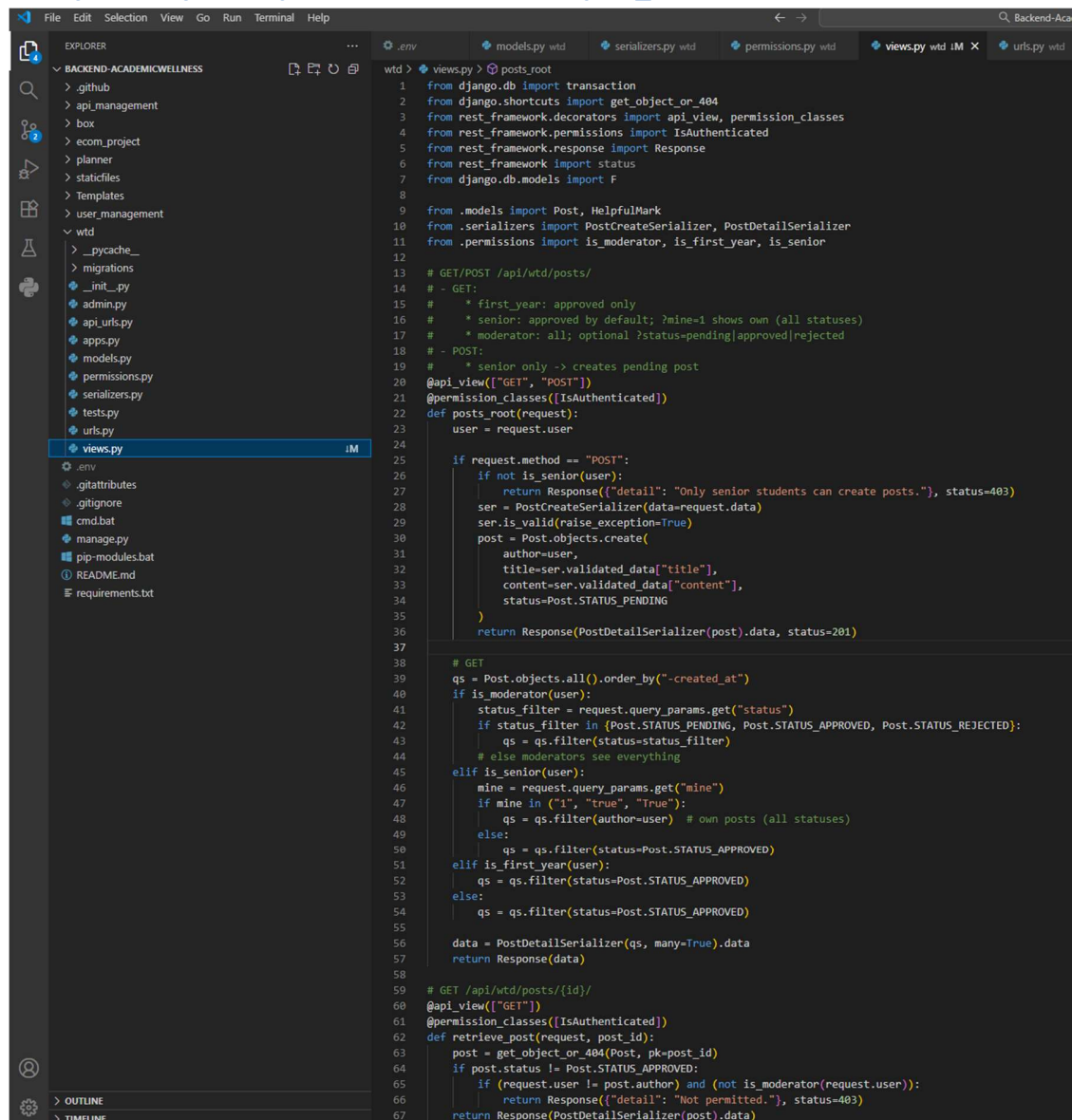
...

```
@api_view(["POST"])
```

```
@permission_classes([IsAuthenticated])
```

```
def mark_helpful(request, post_id):
```

first-year only; idempotent; increments helpful_count



```
1 from django.db import transaction
2 from django.shortcuts import get_object_or_404
3 from rest_framework.decorators import api_view, permission_classes
4 from rest_framework.permissions import IsAuthenticated
5 from rest_framework.response import Response
6 from rest_framework import status
7 from django.db.models import F
8
9 from .models import Post, HelpfulMark
10 from .serializers import PostCreateSerializer, PostDetailSerializer
11 from .permissions import is_moderator, is_first_year, is_senior
12
13 # GET /api/wtd/posts/
14 # - GET:
15 #   * first_year: approved only
16 #   * senior: approved by default; ?mine=1 shows own (all statuses)
17 #   * moderator: all; optional ?status=pending|approved|rejected
18 # - POST:
19 #   * senior only -> creates pending post
20 @api_view(["GET", "POST"])
21 @permission_classes([IsAuthenticated])
22 def posts_root(request):
23     user = request.user
24
25     if request.method == "POST":
26         if not is_senior(user):
27             return Response({"detail": "Only senior students can create posts."}, status=403)
28         ser = PostCreateSerializer(data=request.data)
29         ser.is_valid(raise_exception=True)
30         post = Post.objects.create(
31             author=user,
32             title=ser.validated_data["title"],
33             content=ser.validated_data["content"],
34             status=Post.STATUS_PENDING
35         )
36         return Response(PostDetailSerializer(post).data, status=201)
37
38 # GET
39 qs = Post.objects.all().order_by("-created_at")
40 if is_moderator(user):
41     status_filter = request.query_params.get("status")
42     if status_filter in {Post.STATUS_PENDING, Post.STATUS_APPROVED, Post.STATUS_REJECTED}:
43         qs = qs.filter(status=status_filter)
44     # else moderators see everything
45 elif is_senior(user):
46     mine = request.query_params.get("mine")
47     if mine in ("1", "true", "True"):
48         qs = qs.filter(author=user) # own posts (all statuses)
49     else:
50         qs = qs.filter(status=Post.STATUS_APPROVED)
51 elif is_first_year(user):
52     qs = qs.filter(status=Post.STATUS_APPROVED)
53 else:
54     qs = qs.filter(status=Post.STATUS_APPROVED)
55
56 data = PostDetailSerializer(qs, many=True).data
57 return Response(data)
58
59 # GET /api/wtd/posts/{id}/
60 @api_view(["GET"])
61 @permission_classes([IsAuthenticated])
62 def retrieve_post(request, post_id):
63     post = get_object_or_404(Post, pk=post_id)
64     if post.status != Post.STATUS_APPROVED:
65         if (request.user != post.author and (not is_moderator(request.user))):
66             return Response({"detail": "Not permitted."}, status=403)
67     return Response(PostDetailSerializer(post).data)
```

wtd/views.py

Chatbox App (Messages, Report, Delete)

box/api_urls.py

```
from django.urls import path

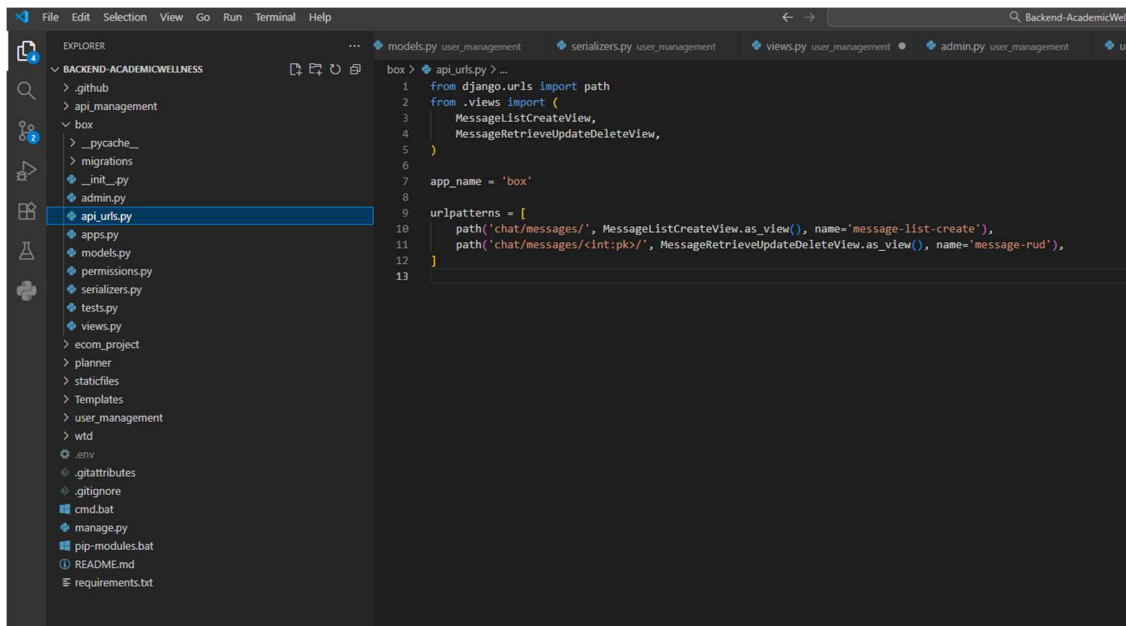
from .views import (
    MessageListCreateView,
    MessageRetrieveUpdateDeleteView,
)

urlpatterns = [

    path('chat/messages/', MessageListCreateView.as_view(), name='message-list-create'),

    path('chat/messages/<int:pk>/', MessageRetrieveUpdateDeleteView.as_view(),
        name='message-rud'),

]
```



box/api_urls.py

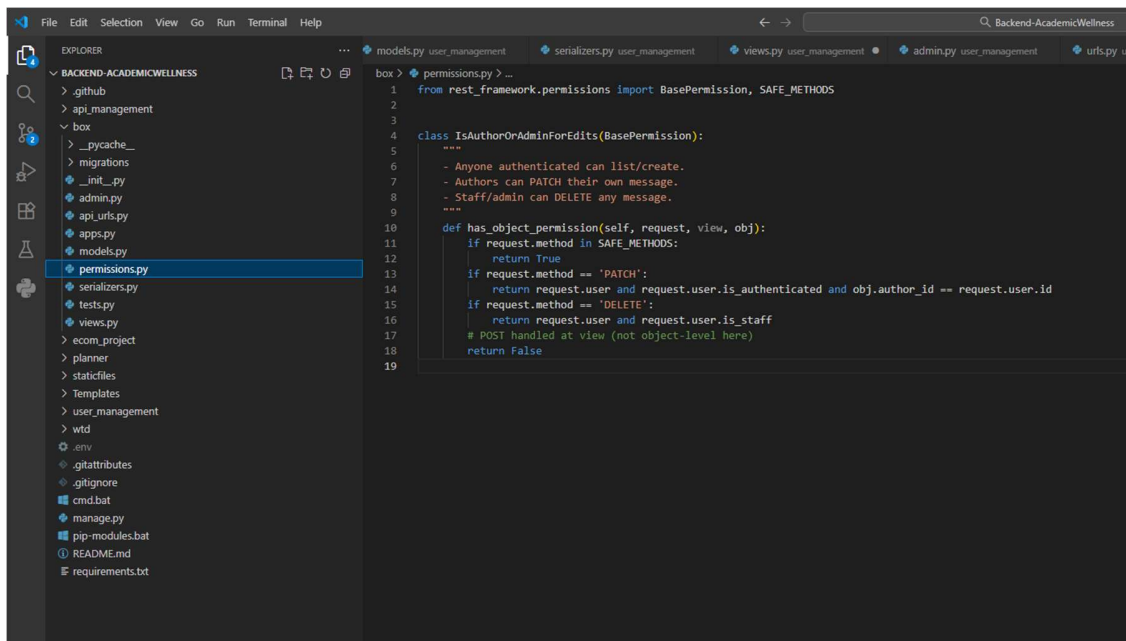
box/permissions.py

```
from rest_framework.permissions import BasePermission, SAFE_METHODS
```

```

class IsAuthorOrAdminForEdits(BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.method in SAFE_METHODS:
            return True
        if request.method == 'PATCH':
            return request.user and request.user.is_authenticated and obj.author_id == request.user.id
        if request.method == 'DELETE':
            return request.user and request.user.is_staff
        return False

```



[box/permissions.py](#)

[box/views.py \(selected\)](#)

```

class MessageListCreateView(generics.ListCreateAPIView):
    serializer_class = ChatMessageSerializer
    permission_classes = [IsAuthenticated]
    pagination_class = MessagePagination
    def get_queryset(self):

```

```
room = self.request.query_params.get('room', 'general')
```

```
qs = ChatMessage.objects.filter(room=room)
```

```
after_id = self.request.query_params.get('after_id')
```

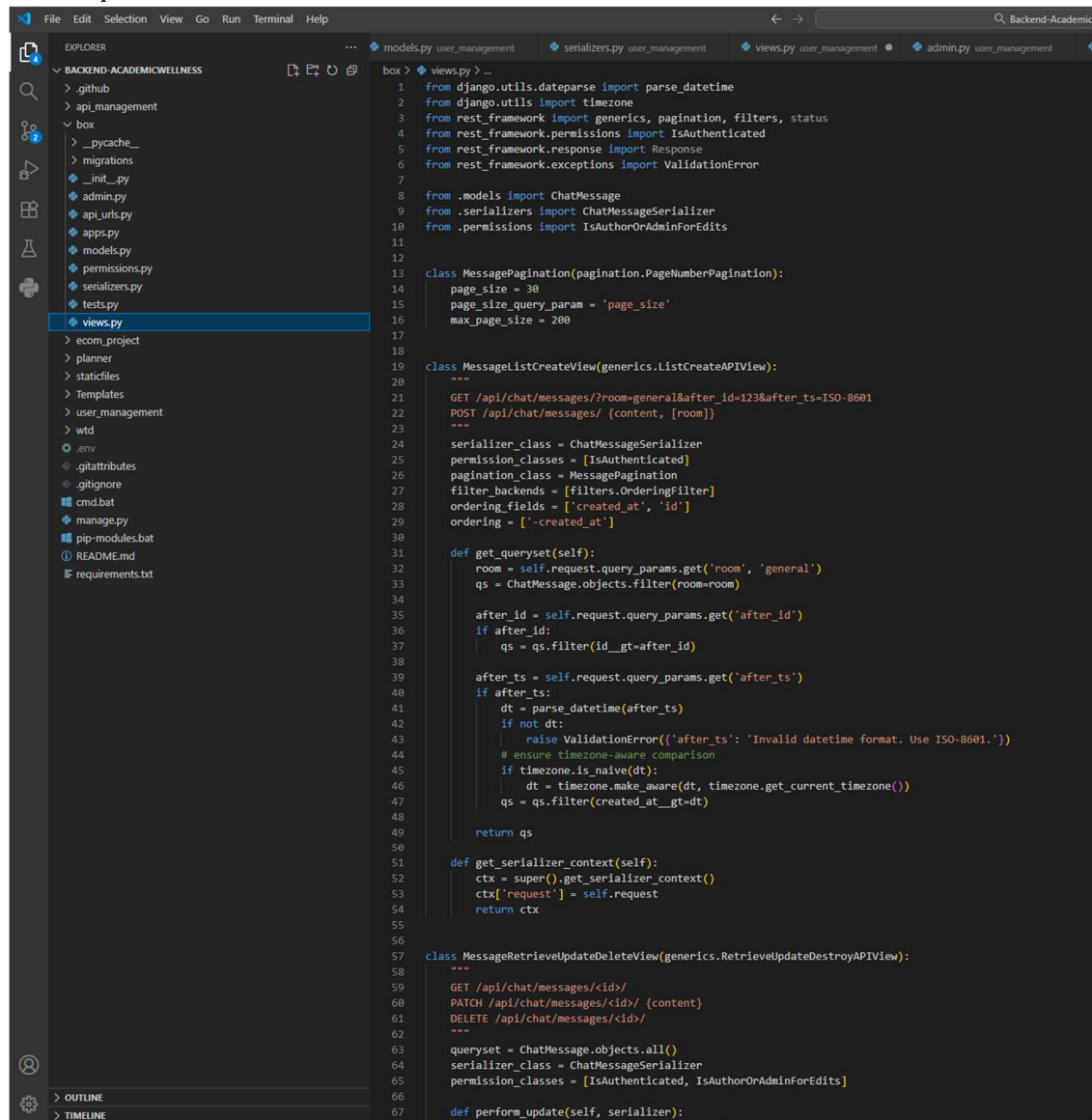
```
if after_id:
```

```
qs = qs.filter(id__gt=after_id)
```

```
after_ts = self.request.query_params.get('after_ts')
```

```
if after_ts:
```

```
return qs
```



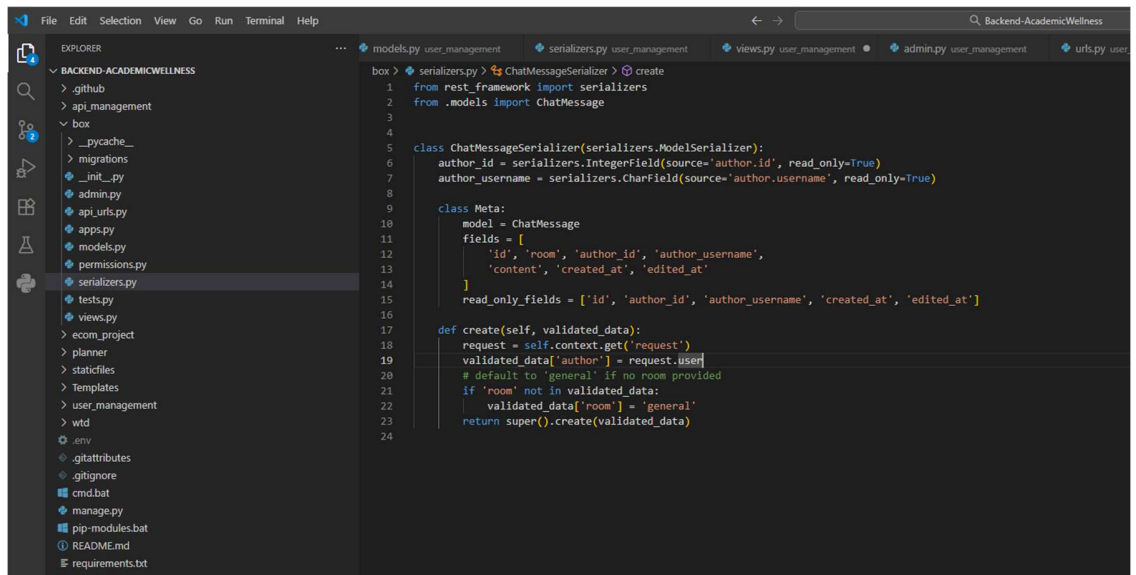
```
File Edit Selection View Go Run Terminal Help
EXPLORER
BACKEND-ACADEMICWELLNESS
  .github
  api_management
  box
  _pycache_
  migrations
  _init_.py
  admin.py
  api_urls.py
  apps.py
  models.py
  permissions.py
  serializers.py
  tests.py
  views.py
ecom_project
planner
staticfiles
Templates
user_management
wtd
.env
.gitattributes
.gitignore
cmd.bat
manage.py
pip-modules.bat
README.md
requirements.txt

box > views.py > ...
1 from django.utils.dateparse import parse_datetime
2 from django.utils import timezone
3 from rest_framework import generics, pagination, filters, status
4 from rest_framework.permissions import IsAuthenticated
5 from rest_framework.response import Response
6 from rest_framework.exceptions import ValidationError
7
8 from .models import ChatMessage
9 from .serializers import ChatMessageSerializer
10 from .permissions import IsAuthorOrAdminForEdits
11
12
13 class MessagePagination(pagination.PageNumberPagination):
14     page_size = 30
15     page_size_query_param = 'page_size'
16     max_page_size = 200
17
18
19 class MessageListCreateView(generics.ListCreateAPIView):
20     """
21     GET /api/chat/messages/?room=general&after_id=123&after_ts=ISO-8601
22     POST /api/chat/messages/ {content, [room]}
23     """
24     serializer_class = ChatMessageSerializer
25     permission_classes = [IsAuthenticated]
26     pagination_class = MessagePagination
27     filter_backends = [filters.OrderingFilter]
28     ordering_fields = ['created_at', 'id']
29     ordering = ['-created_at']
30
31     def get_queryset(self):
32         room = self.request.query_params.get('room', 'general')
33         qs = ChatMessage.objects.filter(room=room)
34
35         after_id = self.request.query_params.get('after_id')
36         if after_id:
37             qs = qs.filter(id__gt=after_id)
38
39         after_ts = self.request.query_params.get('after_ts')
40         if after_ts:
41             dt = parse_datetime(after_ts)
42             if not dt:
43                 raise ValidationError({'after_ts': 'Invalid datetime format. Use ISO-8601.'})
44             # ensure timezone-aware comparison
45             if timezone.is_naive(dt):
46                 dt = timezone.make_aware(dt, timezone.get_current_timezone())
47             qs = qs.filter(created_at__gt=dt)
48
49         return qs
50
51     def get_serializer_context(self):
52         ctx = super().get_serializer_context()
53         ctx['request'] = self.request
54         return ctx
55
56
57 class MessageRetrieveUpdateDeleteView(generics.RetrieveUpdateDestroyAPIView):
58     """
59     GET /api/chat/messages/<id>/
60     PATCH /api/chat/messages/<id>/ {content}
61     DELETE /api/chat/messages/<id>/
62     """
63     queryset = ChatMessage.objects.all()
64     serializer_class = ChatMessageSerializer
65     permission_classes = [IsAuthenticated, IsAuthorOrAdminForEdits]
66
67     def perform_update(self, serializer):
```

[box/views.py](#)

[box/serializers.py \(selected\)](#)

```
class ChatMessageSerializer(serializers.ModelSerializer):  
    author_id = serializers.IntegerField(source='author.id', read_only=True)  
  
    author_username = serializers.CharField(source='author.username', read_only=True)  
  
    class Meta:  
        model = ChatMessage  
  
    fields = ['id', 'room', 'author_id', 'author_username', 'content', 'created_at', 'edited_at']
```



[box/serializers.py](#)

Planner App (Tasks, Month View, Reminders)

[planner/api_urls.py](#)

```
from django.urls import path, include  
  
from rest_framework.routers import DefaultRouter  
  
from .views import TaskViewSet
```

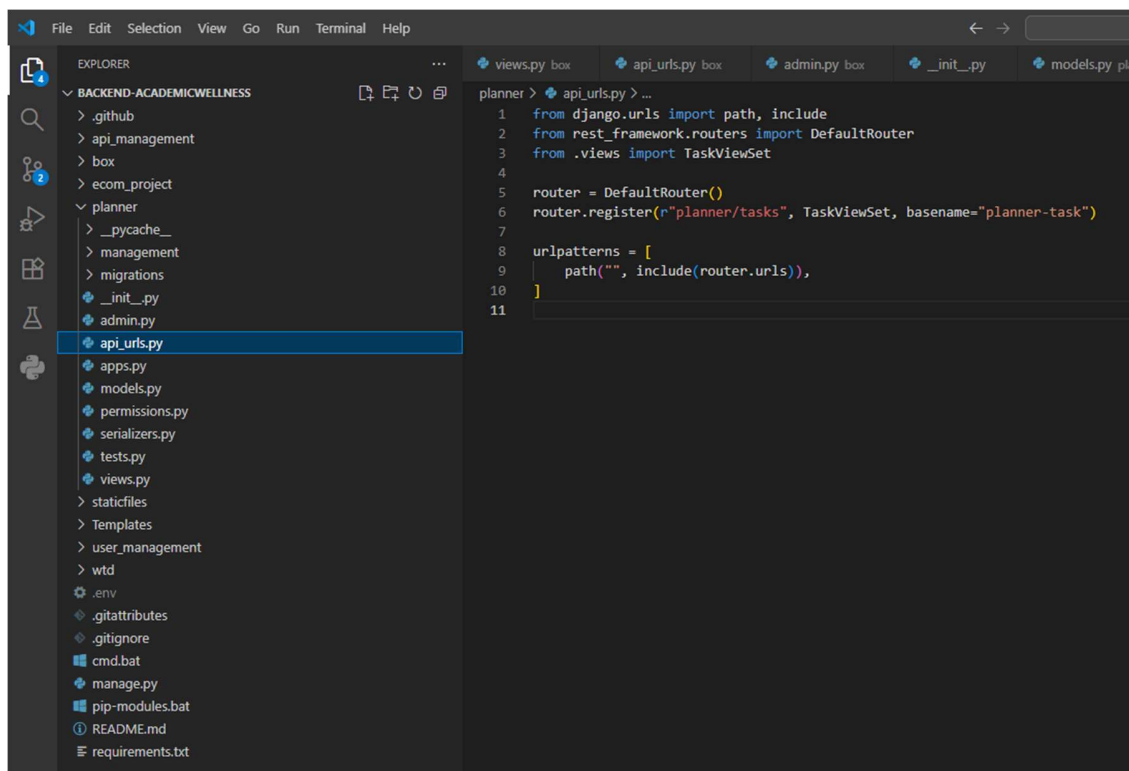
```
router = DefaultRouter()
```

```
router.register(r"planner/tasks", TaskViewSet, basename="planner-task")
```

```
urlpatterns = [
```

```
path("", include(router.urls)),
```

```
]
```



[planner/api_urls.py](#)

[planner/permissions.py](#)

```
from rest_framework.permissions import BasePermission
```

```
from .models import Task
```

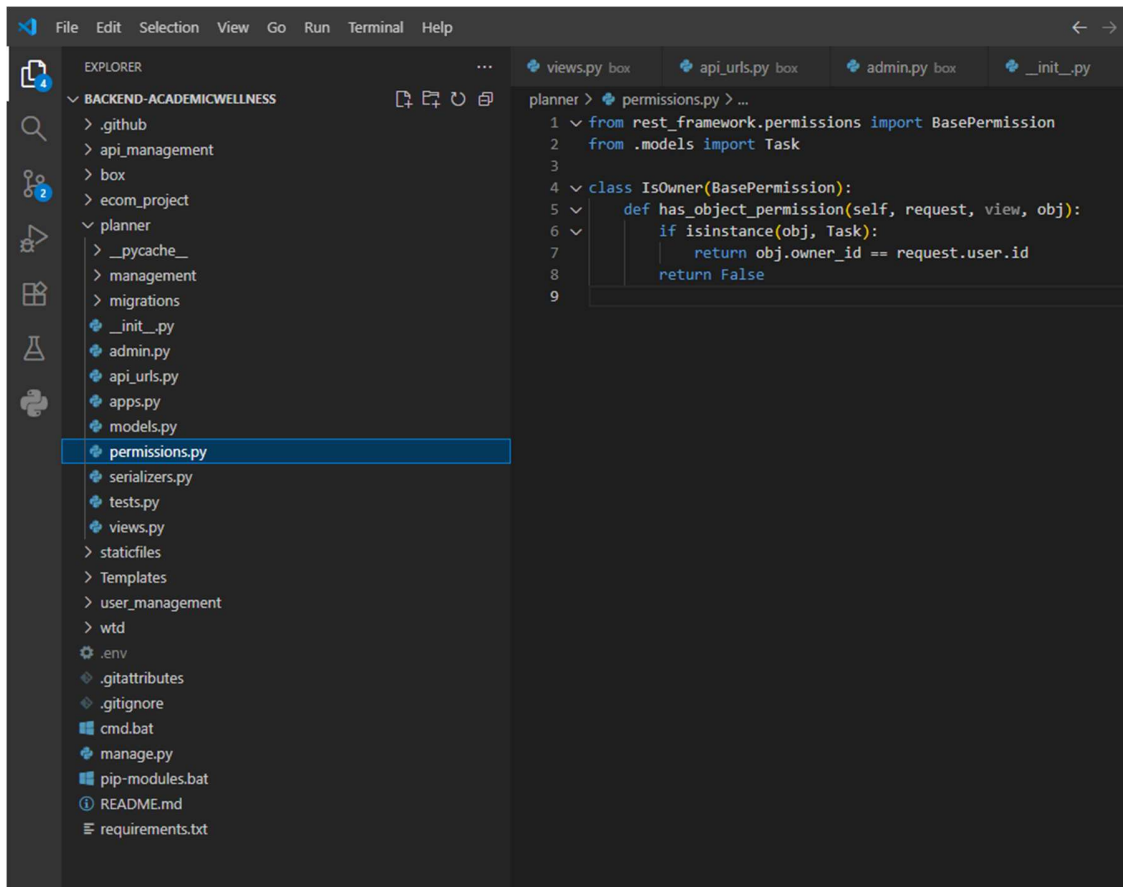
```
class IsOwner(BasePermission):
```

```
def has_object_permission(self, request, view, obj):
```

```
if isinstance(obj, Task):
```

```
return obj.owner_id == request.user.id
```

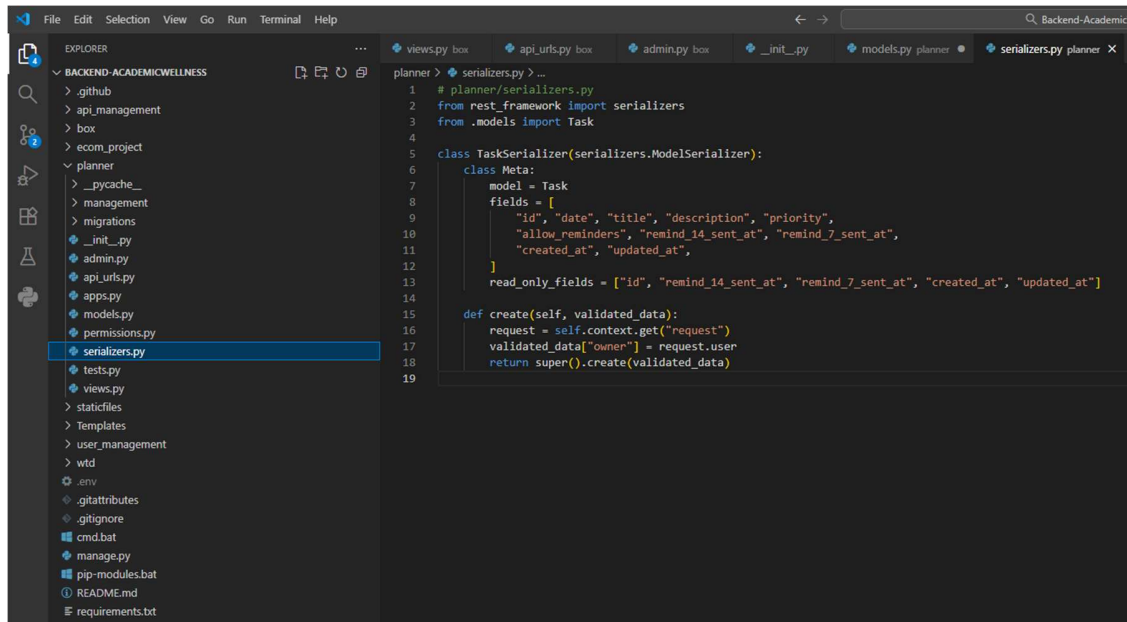
return False



[planner/permissions.py](#)

[planner/serializers.py](#)

```
class TaskSerializer(serializers.ModelSerializer):
    class Meta:
        model = Task
        fields = [
            "id", "date", "title", "description", "priority",
            "allow_reminders", "remind_14_sent_at", "remind_7_sent_at",
            "created_at", "updated_at",
        ]
        read_only_fields = ["id", "remind_14_sent_at", "remind_7_sent_at", "created_at",
            "updated_at"]
```



planner/serializers.py

planner/views.py (selected)

```
class TaskViewSet(viewsets.ModelViewSet):
```

```
    serializer_class = TaskSerializer
```

```
    permission_classes = [IsAuthenticated, IsOwner]
```

```
    pagination_class = StandardPagination
```

```
    def get_queryset(self):
```

```
        user = self.request.user
```

```
        qs = Task.objects.filter(owner=user)
```

```
        start = self.request.query_params.get("start")
```

```
        end = self.request.query_params.get("end")
```

parse and filter dates if provided

```
        return qs.order_by("date", "-created_at")
```

```
@action(detail=False, methods=["get"], url_path="month")
```

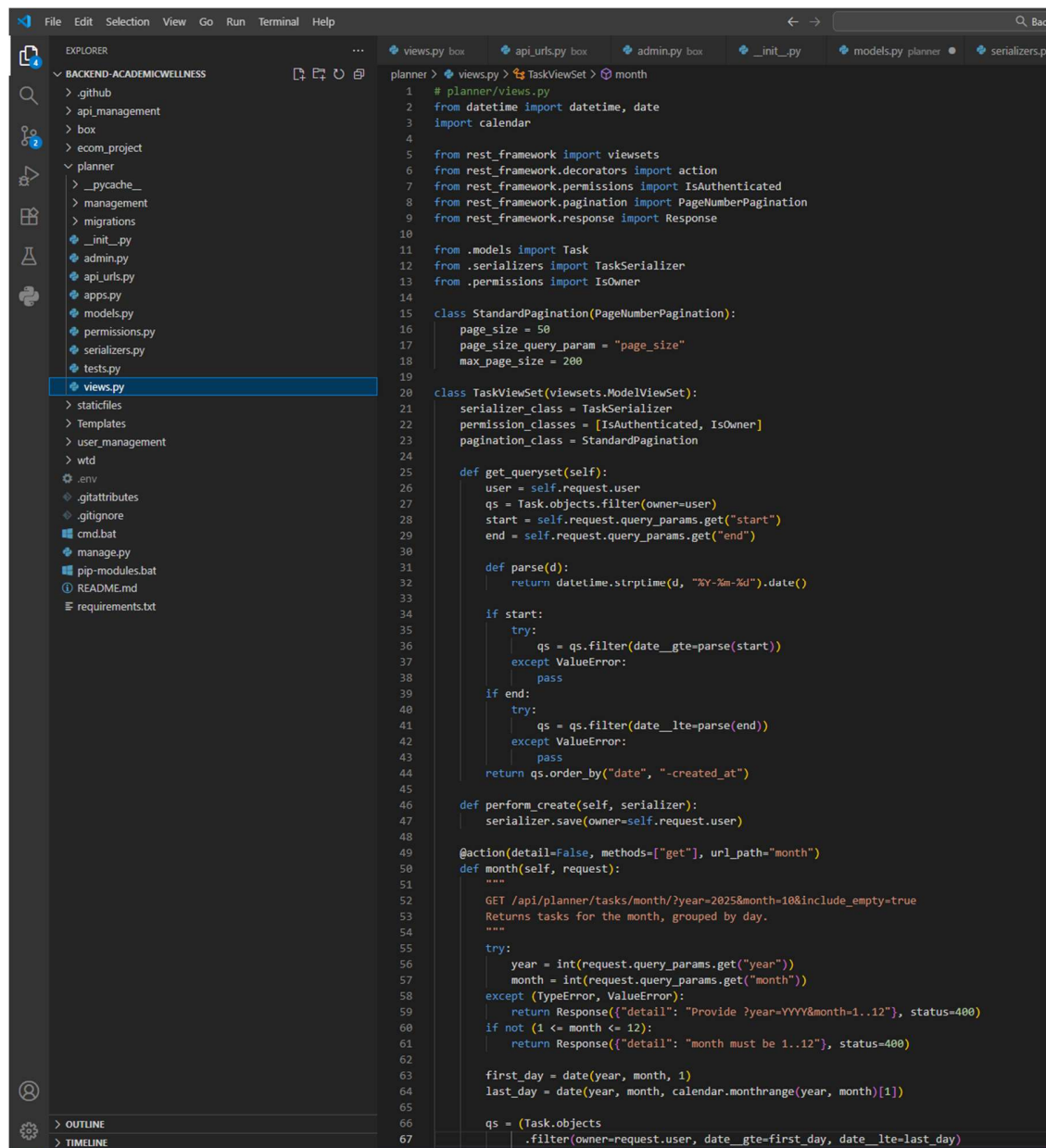
```
def month(self, request):
```

planner/management/commands/send_deadline_reminders.py (selected)

```
class Command(BaseCommand):
```

```
    help = "Send email reminders 14 and 7 days before task date for tasks with  
    allow_reminders=True."
```

```
    def handle(self, *args, **options):
```



```
1  # planner/views.py
2  from datetime import datetime, date
3  import calendar
4
5  from rest_framework import viewsets
6  from rest_framework.decorators import action
7  from rest_framework.permissions import IsAuthenticated
8  from rest_framework.pagination import PageNumberPagination
9  from rest_framework.response import Response
10
11 from .models import Task
12 from .serializers import TaskSerializer
13 from .permissions import IsOwner
14
15 class StandardPagination(PageNumberPagination):
16     page_size = 50
17     page_size_query_param = "page_size"
18     max_page_size = 200
19
20 class TaskViewSet(viewsets.ModelViewSet):
21     serializer_class = TaskSerializer
22     permission_classes = [IsAuthenticated, IsOwner]
23     pagination_class = StandardPagination
24
25     def get_queryset(self):
26         user = self.request.user
27         qs = Task.objects.filter(owner=user)
28         start = self.request.query_params.get("start")
29         end = self.request.query_params.get("end")
30
31         def parse(d):
32             return datetime.strptime(d, "%Y-%m-%d").date()
33
34         if start:
35             try:
36                 qs = qs.filter(date__gte=parse(start))
37             except ValueError:
38                 pass
39         if end:
40             try:
41                 qs = qs.filter(date__lte=parse(end))
42             except ValueError:
43                 pass
44         return qs.order_by("date", "-created_at")
45
46     def perform_create(self, serializer):
47         serializer.save(owner=self.request.user)
48
49 @action(detail=False, methods=["get"], url_path="month")
50 def month(self, request):
51     """
52     GET /api/planner/tasks/month/?year=2025&month=10&include_empty=true
53     Returns tasks for the month, grouped by day.
54     """
55     try:
56         year = int(request.query_params.get("year"))
57         month = int(request.query_params.get("month"))
58     except (TypeError, ValueError):
59         return Response({"detail": "Provide ?year=YYYY&month=1..12"}, status=400)
60     if not (1 <= month <= 12):
61         return Response({"detail": "month must be 1..12"}, status=400)
62
63     first_day = date(year, month, 1)
64     last_day = date(year, month, calendar.monthrange(year, month)[1])
65
66     qs = Task.objects
67     .filter(owner=request.user, date__gte=first_day, date__lte=last_day)
```

planner/views.py

Filters & Ordering Examples

- Planner: `start`, `end` date filters; ordering by `date`, `-created\_at`.
- Chat: `after\_id`, `after\_ts` ISO-8601; ordering by `-created\_at` (paginates).
- WTD: `status` filter for moderators, `?mine=1` for seniors; default approved.

Development Workflow

- Create `.env`, install deps, run migrations.
- Create superuser and test auth flows.
- Use Postman/Insomnia collection to verify Planner, WTD, Chat.
- Add moderators via Django admin (or set `is\_staff`).

Coding Conventions

- DRF serializers/viewsets; pagination via `PageNumberPagination`.
- Use ISO `YYYY-MM-DD` dates and timezone-aware datetimes.
- Keep owner scoping in queriesets; validate inputs defensively.
- Minimal public exposure: health only unauthenticated.

Academic Wellness: Frontend

Project Overview

The **Academic Wellness Platform** frontend was developed to support university students in managing academic wellness, fostering collaboration, and improving productivity.

The frontend is built using **React.js**, with a modular design that ensures scalability, maintainability, and seamless integration with the backend services.

Purpose:

This manual serves as a technical guide for developers and testers to understand how the **frontend architecture, code modules, and functionalities** are structured and implemented.

Technologies used:

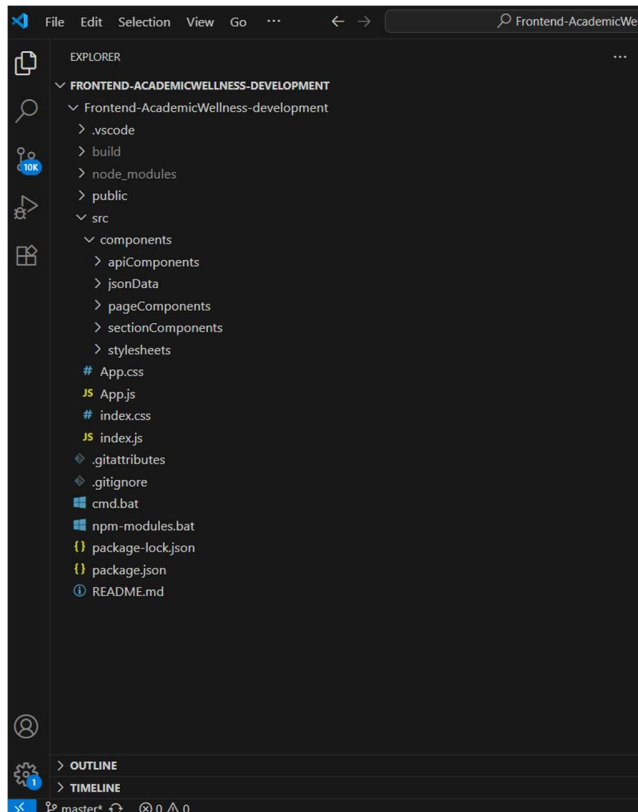
- **React.js** – Main JavaScript library for building the user interface.
- **React Router DOM** – Handles page navigation and protected routes.
- **CSS (App.css, module CSS files)** – Provides styling and layout consistency.
- **LocalStorage** – Temporarily stores user and task data on the client side.
- **RESTful API Integration** – Connects to the backend for login, registration, posts, etc.
- **VS Code** - Integrated Development Environment used for project development.

Project Structure and Modularization

Project Folder Structure:

Caption:

Figure 1: Modular folder structure of the Academic Wellness Frontend project.



The project follows a modular architecture where each folder and file has a dedicated purpose.

This structure ensures separation of concerns, improves collaboration between teams, and simplifies debugging.

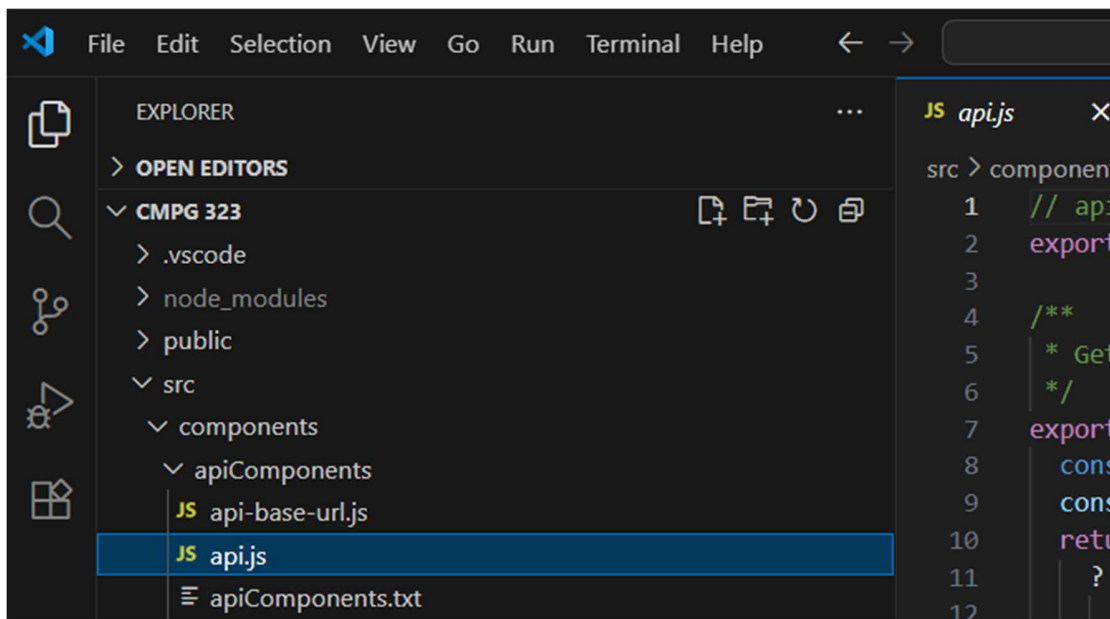
Folder Breakdown:

- **/src** - Main source directory for all frontend code.
- **/components** - Central folder containing reusable and functional code components.
- **/components/apiComponents** – Manages API requests and backend connectivity.
- **/components/jsonData** - Stores temporary or static JSON files for mock data.
- **/components/pageComponents** - Contains full pages such as Login, Register, Dashboard, FAQ, etc. Each page has its own styling and logic files.
- **/components/sectionComponents** - Includes reusable layout sections such as the Header and Footer, along with their respective CSS files.
- **/components/stylesheets** - Currently empty — initially intended for global styles, but most styling is handled within each pageComponents file.

- **App.js** - Main file defining routes, layout, and page rendering logic.
- **index.js** - Application entry point that renders <App /> into the DOM.

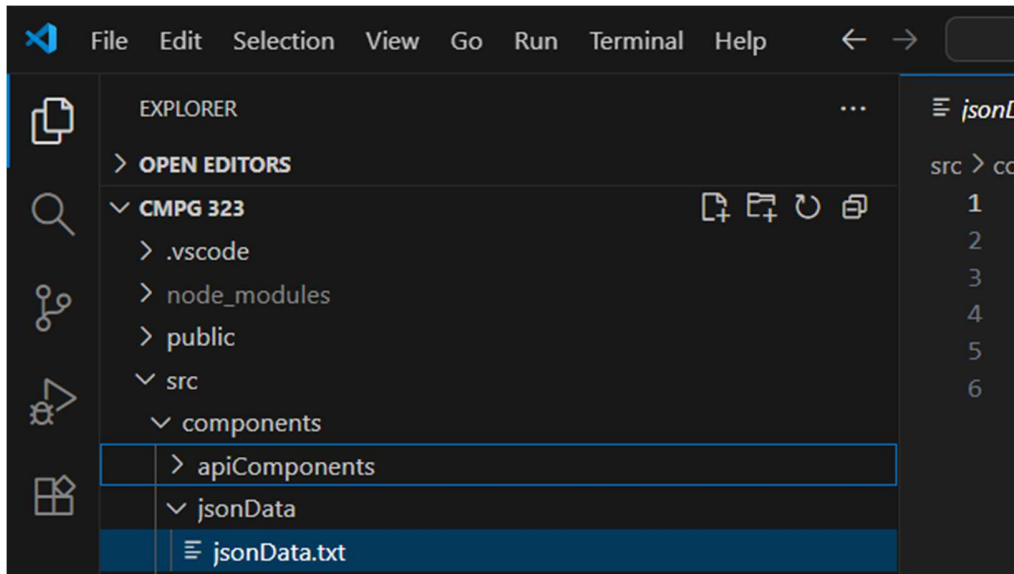
/src/components/apiComponents

- Contains all API integration logic.
- Functions here handle HTTP requests to fetch or send data between the frontend and backend.
- Example: api.js for reusable GET, POST, PUT, and DELETE calls.



/src/components/jsonData

- Stores **static JSON files** used to simulate data locally or as mock data before backend integration.



/src/components/pageComponents

- Contains **individual page components** that represent screens in the application.
- Each file here corresponds to a route in App.js.
Examples:
 - Home.js – Landing page
 - Login.js and Register.js – Authentication screens
 - Dashboard.js – Main student dashboard
 - WhatsTheDifference.js – Discussion page
 - TwoWeekPlanner.js and MonthlyPlanner.js – Academic planners
 - FAQ.js and About.js – Informational pages
- Each page may have its **own CSS file** inside the same directory or use shared styles from /stylesheets.

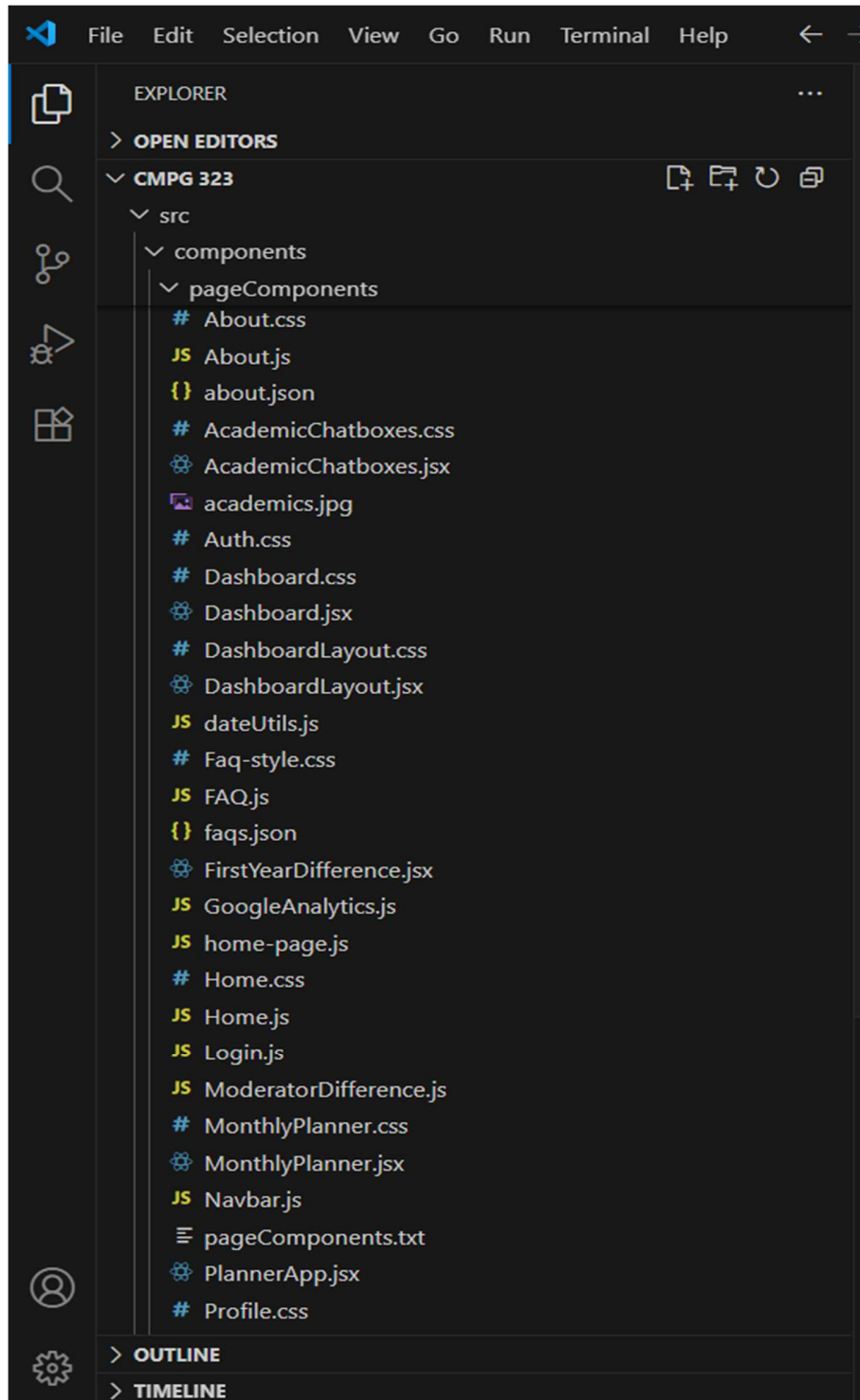


Figure 2: Page components modularized under pageComponents folder.

/src/components/sectionComponents

- Contains **reusable layout sections** like Header and Footer.
- These components are imported globally in App.js so they persist across all routes.
 - header.js + header.css → Top navigation bar
 - footer.js + footer.css → Application footer section

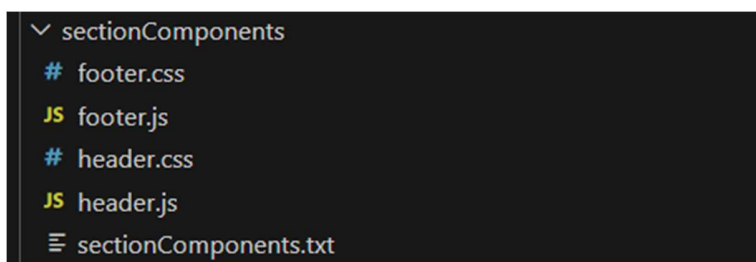


Figure 3: Section components like Header and Footer that maintain consistent layout across pages.

/src/components/stylesheets

- This folder contains global or shared styling files.
- While each page may have individual styles, this directory can hold theme colors, fonts, and utility classes.
- The styling aligns with UniPath's theme — primarily white and light blue gradients.



Figure 4: Stylesheets folder managing global and shared styles for consistency.

Main Frontend File – App.js

App.js acts as the **core of the application**, defining routes, layouts, and rendering logic.

It also manages **authentication, routing protection, and lazy loading** of larger components.

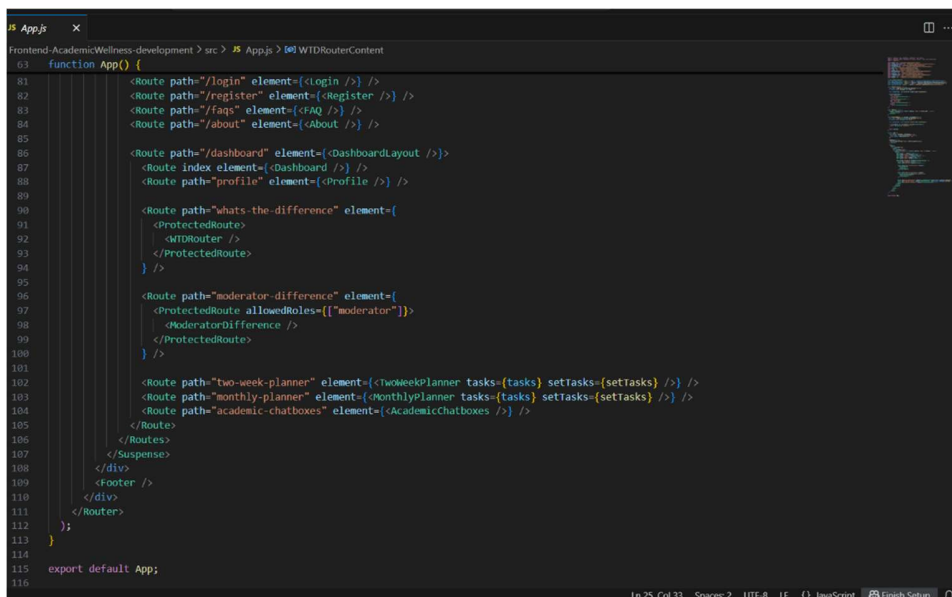
Key Functionalities Explained

Routing

The app uses HashRouter with nested routes inside Routes.

Each route corresponds to a page in /pageComponents.

Code snippets:



```
63 function App() {
81   <Route path="/login" element={<login />} />
82   <Route path="/register" element={<Register />} />
83   <Route path="/faqs" element={<FAQ />} />
84   <Route path="/about" element={<About />} />
85
86   <Route path="/dashboard" element={<DashboardLayout />}
87     <Route index element={<Dashboard />} />
88     <Route path="/profile" element={<Profile />} />
89
90     <Route path="/whats-the-difference" element={
91       <ProtectedRoute>
92         <WIDRouter />
93       </ProtectedRoute>
94     } />
95
96     <Route path="/moderator-difference" element={
97       <ProtectedRoute allowedRoles={['moderator']}>
98         <ModeratorDifference />
99       </ProtectedRoute>
100     } />
101
102     <Route path="/two-week-planner" element={<twoWeekPlanner tasks={tasks} setTasks={setTasks} />} />
103     <Route path="/monthly-planner" element={<monthlyPlanner tasks={tasks} setTasks={setTasks} />} />
104     <Route path="/academic-chatboxes" element={<AcademicChatboxes />} />
105   </Routes>
106 </Suspense>
107 </div>
108 <Footer />
109 </div>
110 </Router>
111 }
112 };
113
114
115 export default App;
116
```

ProtectedRoute Component

Ensures only authenticated users access certain pages.

Redirects to /login if a user isn't logged in.

Code Snippet:

```

49
50 const ProtectedRoute = ({ children, allowedRoles }) => {
51   const user = JSON.parse(localStorage.getItem("user"));
52   if (!user) return <Navigate to="/login" />;
53
54   const studentType = user?.profile?.student_type?.toLowerCase();
55
56   if (allowedRoles && !allowedRoles.includes(studentType)) {
57     return <Navigate to="/dashboard" />;
58   }
59
60   return children;
61 };
62

```

Dynamic Routing for WTD

The WTDRouterContent automatically loads the correct WTD page based on the user's role:

- Moderator → ModeratorDifference.js
- Senior → SeniorDifference.js
- First-year → FirstYearDifference.js

This logic ensures **personalized content** without duplicating routes.

Code snippet:

```

24
25 const WTDRouterContent = () => {
26   const user = JSON.parse(localStorage.getItem("user"));
27   if (!user) return <Navigate to="/login" />;
28
29   const studentType = user?.profile?.student_type?.toLowerCase();
30
31   switch(studentType) {
32     case "moderator":
33       return <ModeratorDifference />;
34     case "senior":
35       return <SeniorDifference />;
36     case "first-year":
37     case "first_year":
38       return <FirstYearDifference />;
39     default:
40       return <FirstYearDifference />;
41   }
42 };
43

```

Lazy Loading

To optimize performance, large components are loaded **only when needed**:

Code snippet:

```
18
19 // Lazy loaded components
20 const WhatsTheDifference = lazy(() => import("./components/pageComponents/WhatsTheDifference"));
21 const ModeratorDifference = lazy(() => import("./components/pageComponents/ModeratorDifference"));
22 const FirstYearDifference = lazy(() => import("./components/pageComponents/FirstYearDifference"));
23 const SeniorDifference = lazy(() => import("./components/pageComponents/SeniorDifference"));
24
```

Figure 5: App.js implementing dynamic routing and lazy loading for optimized performance.

Login & Registration

Purpose:

Allows users to create an account or log in using their credentials, linking directly to the backend API for authentication.

Frontend Implementation:

- Located in /components/pageComponents/Login.js and /Register.js
- Includes form validation and error handling for incorrect input.
- On success, user details are saved in localStorage and the dashboard is loaded.

User Flow:

1. User opens the platform and selects Login or Register.
2. The form validates all required fields.
3. Upon successful login, the system stores a session and redirects the user to the dashboard.

Code snippets:

Login.js

```

JS Login.js x
Frontend-AcademicWellness-development > src > components > pageComponents > JS Login.js > Login > handleLogin
1 import { useState } from "react";
2 import { FaGoogle } from "react-icons/fa";
3 import { Link, useNavigate } from "react-router-dom";
4 import "./Auth.css";
5 import Navbar from "../Navbar";
6 import bgImage from "../academics.jpg";
7
8 function Login() {
9   const [email, setEmail] = useState("");
10  const [password, setPassword] = useState("");
11  const [error, setError] = useState("");
12  const [loading, setLoading] = useState(false);
13  const navigate = useNavigate();
14
15  const handleLogin = async (e) => {
16    e.preventDefault();
17    setError("");
18    setLoading(true);
19
20    // Special case for moderator login
21    if (email === "mialeroux@gmail.com" && password === "stock@234") {
22      setLoading(false);
23      alert("Moderator login successful!");
24      localStorage.setItem(
25        "user",
26        JSON.stringify({
27          profile: { student_type: "moderator", full_name: "Mia" },
28          username: "Mia",
29        })
30      );
31      navigate("/dashboard/whats-the-difference");
32      return;
33    }
34
35    try {
36      const response = await fetch(
37        "https://backend-academicwellness.onrender.com/api/auth/login/",

```

- Mia Leroux is the only moderator who can log in to the system. Her credentials are preconfigured in both the backend database and the frontend logic, allowing direct access to moderator functionalities without additional registration.

Register.js

```

JS Register.js x
Frontend-AcademicWellness-development > src > components > pageComponents > JS Register.js > Register > handleSubmit
1 import { useState } from "react";
2 import { FaGoogle } from "react-icons/fa";
3 import { Link, useNavigate } from "react-router-dom";
4 import "./Auth.css";
5 import Navbar from "../Navbar";
6 import bgImage from "../academics.jpg";
7
8 function Register() {
9   const [fullName, setFullName] = useState("");
10  const [email, setEmail] = useState("");
11  const [password, setPassword] = useState("");
12  const [confirmPassword, setConfirmPassword] = useState("");
13  const [studentType, setStudentType] = useState("First-year");
14  const [error, setError] = useState("");
15  const [loading, setLoading] = useState(false);
16  const navigate = useNavigate();
17
18  const handleSubmit = async (e) => {
19    e.preventDefault();
20
21    if (password !== confirmPassword) {
22      setError("Passwords do not match!");
23      return;
24    }
25
26    setError("");
27    setLoading(true);
28
29    try {
30      const response = await fetch(
31        "https://backend-academicwellness.onrender.com/api/auth/register/",
32        {
33          method: "POST",
34          headers: { "Content-Type": "application/json" },
35          body: JSON.stringify({
36            full_name: fullName,
37            email: email,

```

- The registration interface allows new users to create accounts by entering their details and selecting their student type (First-year or

Senior). Form validation ensures passwords match before submission. Upon successful registration, users are redirected to the Login page. The registration request is processed by the backend API to securely store user information.

“What’s The Difference”

Purpose:

This feature helps senior students share insights about transitioning from school to university. Posts are moderated for quality and visibility.

Frontend Integration:

- Located in /components/pageComponents/WhatsTheDifference.js
- Moderation handled via /ModeratorDifference.js
- User role determines which version of the feature they see:
 - **First-Year:** Can read and mark posts as helpful.
 - **Senior:** Can create new posts.
 - **Moderator:** Can review and approve posts before publishing.

Key Components:

- WTDRouter in App.js dynamically loads correct component per role.
- ProtectedRoute ensures only logged-in users can access.

Code Snippets:

FirstYearDifference.jsx

```

1 import { useEffect, useState } from "react";
2 import { apiRequest } from "../apiComponents/api";
3 import "../whatstheDifference_style.css";
4
5 const FirstYearDifference = () => {
6   const [posts, setPosts] = useState([]);
7   const [loading, setLoading] = useState(false);
8   const [error, setError] = useState("");
9   const user = JSON.parse(localStorage.getItem("user"));
10
11   useEffect(() => {
12     fetchPosts();
13
14     const handleStorage = (event) => {
15       if (event.key === "newWTDPost") fetchPosts();
16     };
17     window.addEventListener("storage", handleStorage);
18     return () => window.removeEventListener("storage", handleStorage);
19   }, []);
20
21   const fetchPosts = async () => {
22     try {
23       setLoading(true);
24       setError("");
25       const data = await apiRequest("/api/wtd/posts/", "GET", null, { status: "approved" });
26       setPosts(Array.isArray(data) ? data : []);
27     } catch (err) {
28       console.error("Error fetching posts:", err);
29       setError("Failed to load posts: " + err.message);
30     } finally {
31       setLoading(false);
32     }
33   };
34
35   const handleHelpful = async (postId) => {
36     try {
37       setError("");

```

- The First-Year Hub displays approved mentorship posts shared by senior students. First-year users can mark posts as “Helpful” to appreciate valuable content. The component retrieves data from the backend via API calls and updates in real time when new posts are added or marked as helpful.

SeniorDifference.jsx

```

1 import { useEffect, useState } from "react";
2 import { apiRequest } from "../apiComponents/api";
3 import "../whatstheDifference_style.css";
4
5 const SeniorDifference = () => {
6   const [posts, setPosts] = useState([]);
7   const [newPost, setNewPost] = useState({ title: "", content: "" });
8   const [showForm, setShowForm] = useState(false);
9   const [loading, setLoading] = useState(false);
10  const [error, setError] = useState("");
11  const user = JSON.parse(localStorage.getItem("user"));
12
13  useEffect(() => {
14    fetchPosts();
15  }, []);
16
17  const fetchPosts = async () => {
18    try {
19      setLoading(true);
20      setError("");
21      if (user) return;
22
23      const data = await apiRequest("/api/wtd/posts/", "GET", null, { author_id: user.id });
24      const filtered = Array.isArray(data) ? data.filter(post => post.status === "approved") : [];
25      setPosts(filtered);
26    } catch (err) {
27      console.error("Error fetching posts:", err);
28      setError("Failed to load posts: " + err.message);
29    } finally {
30      setLoading(false);
31    }
32  };
33
34  const handlePost = async () => {
35    if (newPost.title.trim() === "" || newPost.content.trim() === "") {
36      setError("Please fill in both title and content");
37      return;

```

- The Senior Hub enables senior students to create and manage mentorship posts. Submissions are sent to the backend for moderation before being displayed to first-year students. Approved posts show engagement metrics, such as the number of first-years who found them helpful.

ModeratorDifference.js

```
1 import { useEffect, useState } from "react";
2 import { apiRequest } from "../apiComponents/api";
3 import "../whatstheDifference_style.css";
4
5 const ModeratorDifference = () => {
6   const [posts, setPosts] = useState([]);
7   const [filter, setFilter] = useState("all");
8   const [loading, setLoading] = useState(false);
9   const [error, setError] = useState("");
10  const [user, setUser] = useState(null);
11
12  useEffect(() => {
13    const userData = JSON.parse(localStorage.getItem("user"));
14    setUser(userData);
15
16    if (userData) {
17      fetchPosts();
18
19      const handleStorage = (event) => {
20        if (event.key === "newMDPost") {
21          fetchPosts();
22        }
23      };
24      window.addEventListener("storage", handleStorage);
25
26      return () => {
27        window.removeEventListener("storage", handleStorage);
28      };
29    }
30  }, [filter]);
31
32  const fetchPosts = async () => {
33    try {
34      setLoading(true);
35      setError("");
36      const queryParams = filter !== "all" ? { status: filter } : {};
37      const data = await apiRequest("/api/v1/posts/", "GET", null, queryParams);
```

- The Moderator Dashboard enables authorized users (like Mia Leroux) to review, approve, or reject posts submitted by senior students. It communicates with backend moderation endpoints and dynamically updates the post list, ensuring that only approved and appropriate content is displayed to first-year students.

Two-Week Task Planner (Eisenhower Matrix)

Purpose:

Combines a 2-week planner with the Eisenhower Matrix to help students organize and prioritize academic tasks effectively.

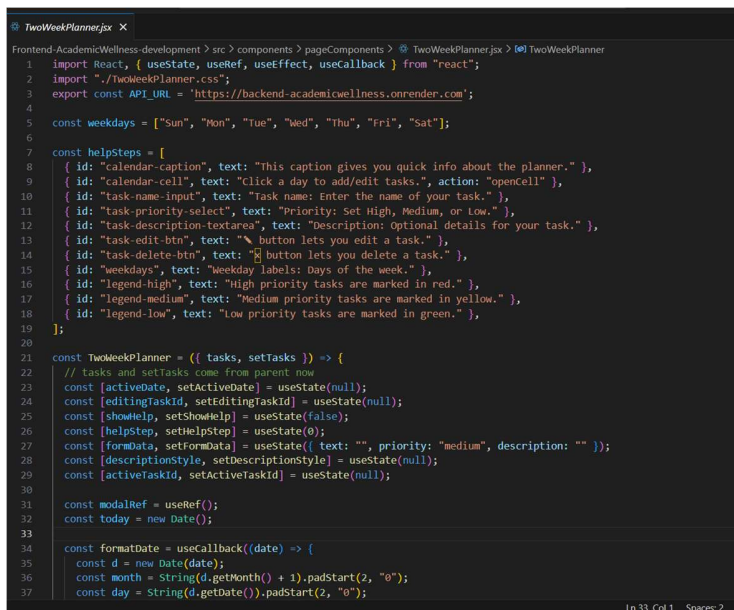
Frontend Implementation:

- Located in /components/pageComponents/TwoWeekPlanner.js
- Tasks are categorized as:
 - **High Importance**
 - **Medium Importance**
 - **Low Importance**
- Users can move tasks between quadrants and automatically generate a two-week calendar schedule.

Features:

- LocalStorage saves tasks even after refresh.
- Real-time editing and reordering of tasks.
- Integrated instructional section for prioritization strategies.

Code snippet:



```
1 import React, { useState, useRef, useEffect, useCallack } from "react";
2 import "../TwoWeekPlanner.css";
3 export const API_URL = "https://backend-academicwellness.onrender.com";
4
5 const weekdays = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
6
7 const helpSteps = [
8   { id: "calendar-caption", text: "This caption gives you quick info about the planner." },
9   { id: "calendar-cell", text: "Click a day to add/edit tasks.", action: "opencell" },
10  { id: "task-name-input", text: "Task name: Enter the name of your task." },
11  { id: "task-priority-select", text: "Priority: Set High, Medium, or Low." },
12  { id: "task-description-textarea", text: "Description: Optional details for your task." },
13  { id: "task-edit-btn", text: "button lets you edit a task." },
14  { id: "task-delete-btn", text: "button lets you delete a task." },
15  { id: "weekdays", text: "Weekday labels: Days of the week." },
16  { id: "legend-high", text: "High priority tasks are marked in red." },
17  { id: "legend-medium", text: "Medium priority tasks are marked in yellow." },
18  { id: "legend-low", text: "Low priority tasks are marked in green." },
19 ];
20
21 const TwoWeekPlanner = ({ tasks, setTasks }) => {
22   // tasks and setTasks come from parent now
23   const [activeDate, setActiveDate] = useState(null);
24   const [editingTaskId, setEditingTaskId] = useState(null);
25   const [showHelp, setShowHelp] = useState(false);
26   const [helpStep, setHelpStep] = useState(0);
27   const [formData, setFormData] = useState({ text: "", priority: "medium", description: "" });
28   const [descriptionStyle, setDescriptionStyle] = useState(null);
29   const [activeTaskId, setActiveTaskId] = useState(null);
30
31   const modalRef = useRef();
32   const today = new Date();
33
34   const formatDate = useCallack((date) => {
35     const d = new Date(date);
36     const month = String(d.getMonth() + 1).padStart(2, "0");
37     const day = String(d.getDate()).padStart(2, "0");
```

- A fully interactive two-week calendar planner with task creation, editing, deletion, priority labels, backend integration, and an in-app help tour highlighting key UI elements.

Monthly Academic Planner with Automated Reminders

Purpose:

Visualizes academic deadlines for the entire month and sends automated reminders (via backend API).

Frontend Implementation:

- Located in /components/pageComponents/MonthlyPlanner.js
- Features include:
 - Add, edit, and delete deadlines.
 - Generate printable PDF schedule.
 - Save all planner data to the database for reminder automation.

Code Snippet:

```
MonthlyPlanner.jsx
Frontend-AcademicWellness-development > src > components > pageComponents > MonthlyPlanner.jsx > MonthlyPlanner
1 import React, { useState, useRef, useEffect, useCallback } from "react";
2 import "./MonthlyPlanner.css";
3 export const API_URL = "https://backend-academicwellness.onrender.com";
4
5 const WEEKDAYS = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
6 const THEME_COLORS = ["#6366f1", "#4a90e2", "#e57373", "#81c784", "#ffb74d"];
7
8 const MonthlyPlanner = ({ tasks: propTasks = [], setTasks: propSetTasks }) => {
9   const [tasks, setTasks] = useState(propTasks);
10   const setTasksSafe = propSetTasks || setTasks;
11
12   useEffect(() => {
13     setTasks(propTasks);
14   }, [propTasks]);
15
16   const [currentDate, setCurrentDate] = useState(new Date());
17   const [selectedDate, setSelectedDate] = useState(null);
18   const [editingTaskId, setEditingTaskId] = useState(null);
19   const [formData, setFormData] = useState({
20     text: "",
21     priority: "medium",
22     description: "",
23     time: "",
24   });
25
26   const [showSettings, setShowSettings] = useState(false);
27   const [themeColor, setThemeColor] = useState(THEME_COLORS[0]);
28   const [startWeekday, setStartWeekday] = useState(false);
29   const [activeTaskId, setActiveTaskId] = useState(null);
30   const [highlightWeekends, setHighlightWeekends] = useState(true);
31
32   const formRef = useRef();
33
34   const monthName = currentDate.toLocaleString(undefined, {
35     month: "long",
36     year: "numeric",
37   });
```

- The Monthly Planner allows students to view and organize their academic tasks using a color-coded calendar. Tasks are synchronized with the backend, enabling secure creation, editing, and deletion. Users can also customize appearance settings and print their planner for offline reference.

Academic Chatboxes

Purpose:

A moderated space where students communicate about assignments, exams, and academic topics.

Frontend Implementation:

- Located in /components/pageComponents/AcademicChatboxes.js
- Designed for real-time conversation (future WebSocket support possible).
- Displays instructions on joining and starting new chats.
- Initial version includes one general academic chatroom.

Code Snippet:

```

AcademicChatboxes.jsx X
Frontend-AcademicWellness-development > src > components > pageComponents > AcademicChatboxes.jsx > AcademicChatboxes
1 // 1. IMPORT usecallback from react
2 import React, { useState, useRef, useEffect, useCallback } from 'react';
3 import './AcademicChatboxes.css';
4
5 const API_BASE_URL = 'https://backend-academicwellness.onrender.com/api';
6
7 // --- Helper Functions to interact with localStorage ---
8 const getToken = () => localStorage.getItem('token');
9
10 const getCurrentUser = () => {
11   try {
12     const userString = localStorage.getItem('user');
13     return userString ? JSON.parse(userString) : null;
14   } catch (error) {
15     console.error("Failed to parse user from localStorage:", error);
16     return null;
17   }
18 };
19
20
21 const AcademicChatboxes = () => {
22   const [activeChannel, setActiveChannel] = useState('#General');
23   // 2. 'setchannels' is removed as it's not used
24   const [channels] = useState([
25     { id: '#General', name: 'General Lounge', description: 'General academic discussions', unread: 0 },
26   ]);
27   const [messages, setMessages] = useState({ '#General': [] });
28   const [newMessage, setNewMessage] = useState('');
29   // 3. 'isTyping' state is removed as it's not used
30   const [isLoading, setIsLoading] = useState(true);
31   const [isEditing, setIsEditing] = useState(false);
32   const [editingMessageId, setEditingMessageId] = useState(null);
33
34   const currentUser = getCurrentUser();
35   const messagesEndRef = useRef(null);
36   const inputRef = useRef(null);
37

```

- The Academic Chatboxes allows students to exchange academic messages in real time. Users can post, edit, and view messages securely via backend integration, while the responsive interface adapts seamlessly to desktop and mobile devices.

FAQ Page

Purpose:

Provides answers to common academic and platform-related questions.

Frontend Implementation:

- Located in /components/pageComponents/FAQ.js
- Users can:
 - Search for existing FAQs.
 - Submit new questions and answers (go through moderation).
- Simple accordion-style design for readability.

Code Snippet:

```

14 FAQ.js
Frontend-AcademicWellness-development > src > components > pageComponents > FAQ.js > FAQ
1 import React, { useState } from "react";
2 import faqs from "../faqs";
3 import "../faq-style.css";
4
5 function FAQ() {
6   const [searchTerm, setSearchTerm] = useState("");
7   const [feedback, setFeedback] = useState(null);
8   const [formData, setFormData] = useState({
9     name: "",
10    email: "",
11    message: ""
12  });
13
14  const filteredFaqs = faqs.filter(faq =>
15    faq.question.toLowerCase().includes(searchTerm.toLowerCase())
16  );
17
18  const handleInputChange = (e) => {
19    const { name, value } = e.target;
20    setFormData(prev => ({ ...prev, [name]: value }));
21  };
22
23  const handleFormSubmit = (e) => {
24    e.preventDefault();
25    // Handle form submission here (e.g., send to API or email service)
26    setFormData({ name: "", email: "", message: "" });
27    setFeedback("submitted");
28  };
29
30  const handleYesClick = () => { setFeedback("thankyou") }
31
32  return (
33    <div>
34      <div>
35        <div className="header">
36          <h1 className="headerf-title">FAQs</h1>
37          <p className="headerf-desc">Got a question? Below are some of the most frequently asked questions.</p>
38        </div>
39      </div>
40    </div>
41  );
42
43  // Using FAQ.js and its dependencies
44  In 31, Col 1, Spa

```

- The FAQ page allows users to search and browse common platform-related questions interactively. It also includes a feedback form where users can contact the development team if their queries are unresolved. The system is designed for future backend integration to store and manage user feedback.

Header & Footer Components

Purpose:

Maintain a consistent layout across all pages.

Frontend Implementation:

- Located in /components/sectionComponents/header.js and /footer.js
- Styles defined in /components/sectionComponents/header.css and /footer.css
- Contain navigation links, logo, and contact info.

Styling and Theme

Color Scheme:

- Primary: Light blue and white (symbolizing calmness and clarity).
- Accent: Deep blue highlights for contrast and focus.
- Typography: *Segoe UI* and *Verdana* for readability.

Styling Approach:

- Each page under /pageComponents includes its own dedicated CSS file for isolated styling.
- Shared components like Header and Footer have their own CSS in /sectionComponents.
- The global layout styles and responsive adjustments are defined in App.css.

Installation guide

The installation guide based on the frontend's academic wellness:

<https://drive.google.com/file/d/1bBr2Hl7HqPVAHxp2Qoefxo0GdQh93b6L/view?usp=sharing>

Social Wellness: Frontend

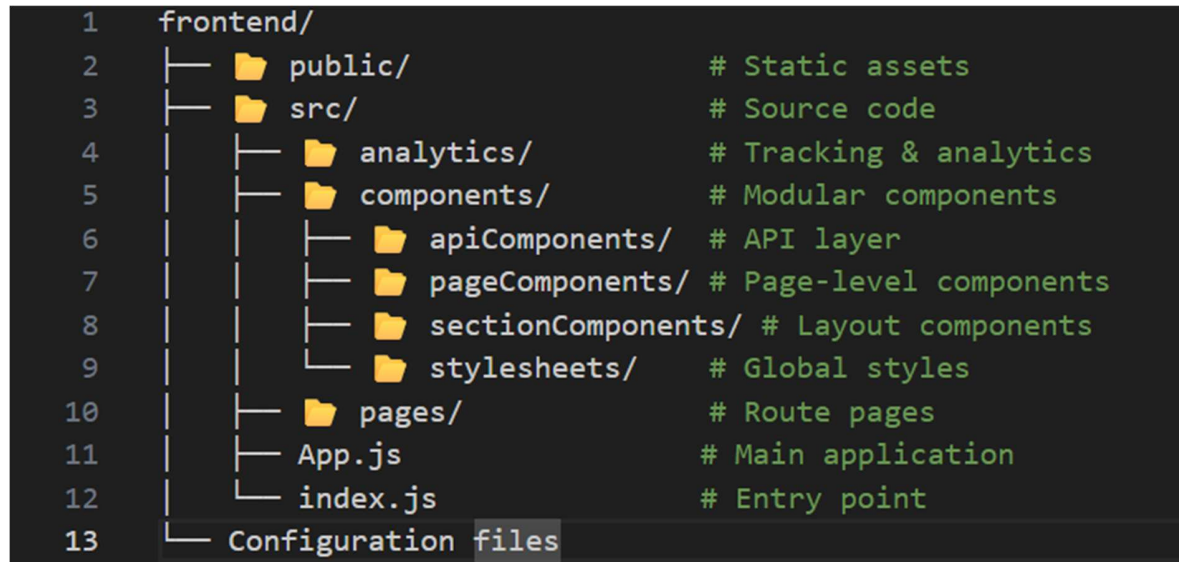
1. Project Overview

The frontend of Unipath Social wellness is developed using React.js, designed with modularity, scalability, and reusability.

Each page, component, and API integration is organized into structured directories to ensure ease of maintenance and collaboration.

- **Framework:** React.js
- **Language:** JavaScript
- **Styling:** CSS Modules + Styled Components
- **Routing:** React Router
- **Package Manager:** npm
- **Folder Structure:** Feature-based modularization

2. Folder structure: frontend



3. Code Modularization

3.1 API Functionality

Path: src/components/apiComponents/

All API interactions between the frontend and backend are handled here.

✓	apiComponents/	# Handles all API interactions between frontend and backend
	├ api-base-url.js	# Defines the base backend URL (http://127.0.0.1:8000)
	├ api.js	# Configures Axios instance/global interceptors for auth and errors
	├ apiComponents.txt	# Developer notes and explanations about API components
	├ authApi.js	# Handles authentication: login, register, token refresh, logout
	├ chatApi.js	# Manages chatbox endpoints (get, send, receive, delete messages)
	├ cultureApi.js	# Retrieves or updates culture-related posts and data
	├ cultureModeratorApi.js	# Handles moderator actions (approve, reject cultural content)
	├ mapsApi.js	# Fetches campus or map-related data
	└ weNeedAPI.js	# Handles "WeNeed" requests, post, and feedback endpoints

3.2 Page components

Path: src/components/pageComponents/

Houses feature-specific pages, each with its own styles and logic.

	pageComponents/	# Contains feature-specific pages and UI layouts
	├ AdminWeNeed/	
	├ AdminWeNeedPage.css	# Styles for Admin WeNeed Dashboard
	├ AdminWeNeedPage.js	# Admin interface logic for managing WeNeed submissions
	├ AuthPage/	
	├ AuthForm.styled.js	# Styled components for auth form elements
	├ LoginPage.js	# Login form with email/password authentication
	├ RegisterPage.js	# Registration form for new users
	├ CampusMapPage/	
	├ CampusMapPage.css	# Styles for the map view
	├ CampusMapPage.js	# Displays interactive campus map using Maps API
	├ CommunicationSkillsPage/	
	├ CommunicationSkillsPage.css	# Styles for communication page
	├ CommunicationSkillsPage.js	# Displays communication skills tips and activities
	├ HomePage/	
	├ home-page.js	# Main landing page logic and layout
	├ HomePage.styled.js	# Styled components for homepage design
	├ MyCultureModeratorPage/	
	├ MyCultureModeratorPage.css	# Styles for moderator dashboard
	├ MyCultureModeratorPage.js	# Logic for reviewing and moderating cultural posts
	├ MyCulturePage/	
	├ MyCulturePage.css	# Styles for user's culture page
	├ MyCulturePage.js	# Displays and manages user's cultural submissions

```

├─ SocialChatboxPage/
│  └─ SocialChatbox.css          # Chat UI styles
│  └─ SocialChatboxPage.js      # Real-time chat UI logic and API integration
├─ WeNeedPage/
│  └─ WeNeedPage.css           # Styles for WeNeed submission page
│  └─ WeNeedPage.js            # Handles "WeNeed" form submission and listing
├─ AppWrapper.styled.js        # Global wrapper for consistent app layout
└─ pageComponents.txt           # Developer documentation or layout reference

```

3.3 Section Components

Path: src/components/sectionComponents/

Reusable layout components used across multiple pages.

```

├─ sectionComponents/          # UI sections shared across multiple pages
├─ footer.js                   # Site footer
├─ header.js                   # Basic header for Authentication pages
├─ sectionComponents.txt       # Documentation for section components
└─ SiteHeader.js               # Top navigation bar with logo and layout for navigation

```

4. Analytics

Path: src/analytics/

Tracks page visits and user interactions for reporting.

```

├─ analytics/                  # User tracking and analytics integration
├─ ga.js                       # Google Analytics setup and event configuration
└─ PageViewTracker.js          # Tracks and logs page views during route changes

```

5. App Core Files

Path: src/

```

App.js           # Root component that defines app routes and structure
App.css          # Global app styling
index.js         # React entry point, renders <App /> to the DOM
index.css        # Global base styles and resets
custom.d.ts      # Type definitions

cmd.bat          # Windows batch file to run development commands
generate-react-cli.json # Config for React component generation tool
npm-modules.bat  # Batch file to reinstall npm dependencies
package-lock.json # Lock file for npm dependencies
package.json     # Project metadata, dependencies, and scripts
README.md        # Project documentation and setup guide

```

6. Routing Flow

Route	Page Component	Purpose
/	HomePage	App landing page
/login	LoginPage	User authentication
/register	RegisterPage	User registration
/map	CampusMapPage	NWU campus map
/chat	SocialChatboxPage	Real-time chat
/myculture	MyCulturePage	User's cultural activities
/myculturemoderatorpage	MyCultureModeratorPage	Moderator dashboard
/communicationskills	CommunicationskillsPage	Videos and information
/weneed	WeNeedPage	Student requests and feedback
/adminweneed	AdminWeNeedPage	Admin panel for WeNeed requests

7. Development Setup

- **Step 1:** Install Dependencies

npm install

- **Step 2:** Run the Application

npm start

This runs the app in development mode —

Access it via: <http://localhost:3000/>

- **Step 3:** Build for Production

npm run build

- **Step 4:** Lint & Format Code

npm run lint

npm run format

8. Authentication Workflow

1. User logs in via authAPI
2. API responds with access and refresh tokens.
3. Tokens are saved in localStorage.
4. Axios interceptors automatically attach the token to each request.

5. On expiration, the refresh token regenerates a new access token.

```
4  const api = axios.create({
5    |   baseURL: `${API_URL}/api/`,
6    |   headers: {
7    |     "Content-Type": "application/json",
8    |   },
9  | });
10
11  // 🌸 Add token if available
12  api.interceptors.request.use((config) => {
13    |   const token = localStorage.getItem("access_token");
14    |   if (token) {
15    |     config.headers.Authorization = `Bearer ${token}`;
16    |   }
17    |   return config;
18  | });
19
20  // 🚫 Auto redirect on 401 Unauthorized
21  api.interceptors.response.use(
22    | (response) => response,
23    | (error) => {
24    |   if (error.response?.status === 401) {
25    |     localStorage.removeItem("access_token");
26    |     localStorage.removeItem("refresh_token");
27    |     localStorage.removeItem("user");
28    |     window.location.href = "/login";
29    |   }
30    |   return Promise.reject(error);
31  | }
32  | );
33
34  export default api;
```

9. Security Practices

- Principle of Least Privilege -Only grant necessary permissions to users and components.
- Input Validation - Prevent buffer overflows or injection attacks.

10. Developer Notes

- Keep each API module lightweight and focused on one domain.

- Use async/await for cleaner API handling.
- Maintain consistent naming conventions
- Keep page and section components isolated to prevent code overlap.

11. Version Control Workflow

1. Create feature branch:

git checkout -b feature/new-component

2. Commit changes:

git add .

git commit -m "Added new component"

3. Merge to development:

4. git checkout development

git merge feature/new-component

12. Code implementations

12.1 Authentication Pages

Handling submit login authentication with the use of AuthApi.js

```
23 const handleSubmit = async (e) => {
24   e.preventDefault();
25   if (!email || !password) {
26     setError("Please enter both email and password.");
27     return;
28   }
29
30   try {
31     setLoading(true);
32     setError("");
33
34     // use modular API method
35     const data = await authAPI.login(email, password);
36     console.log("🔵 LOGIN RESPONSE DATA:", data); // Debug log
37
38     // Get the actual user data from backend
39     let user;
40     if (data.user) {
41       // If backend returns user data in login response
42       user = data.user;
43     } else {
44       // If not, fetch the complete user profile from /api/me/
45       try {
46         user = await authAPI.getUser(); // This should call your MeView
47       } catch (fetchError) {
48         console.error("Failed to fetch user profile:", fetchError);
49         // Fallback - but don't hardcode role
50         user = {
51           email: email,
52           // Role will be undefined, which is better than wrong
53         };
54       }
55     }
56   }
```

AuthApi.js: Get user

```
// GET CURRENT USER
getUser: async () => {
  try
  {
    const response = await api.get("auth/me/");
    localStorage.setItem("user", JSON.stringify(response.data));
    return response.data;
  }
  catch (error)
  {
    console.error("Error fetching user data:", error); throw error; }
},
;
```

Note: Administrators are able to access every page within the project, this includes the moderator panel, and We Need Submissions. When logged In the user will be able to see their role on site header as well as the console logs are set information indicating user's role.

Registration will need new users' full details(full name, surname, email address, password). This will then be stored to the database and process.

12.2 MyCulture Page

User:

Sends personal culture-related activities, posts, or achievements for the user, for Approval by Admin.

```
// Submit post for approval
const handlePostSubmit = async () => {
  if (!newPost || !selectedCulture) {
    setError("Please select a culture and write a message");
    return;
  }
  try {
    setPosting(true);
    setError("");
    const response = await cultureAPI.createPost({
      culture: selectedCulture,
      text_message: newPost
    });
    setNewPost("");
    setSelectedCulture("");
  } catch (err) {
    setError(err.response?.data?.message || "Failed to create post.");
  } finally {
    setPosting(false);
  }
};
```

chatApi.js: Api calls to get previous post and create post

```

export const cultureAPI = {
  // Posts
  getPosts: (filters = {}) => {
    const params = new URLSearchParams();
    Object.keys(filters).forEach(key => {
      if (filters[key] !== undefined && filters[key] !== null && filters[key] !== '') {
        params.append(key, filters[key]);
      }
    });
    return api.get(`culture-posts/?${params.toString()}`);
  },

  createPost: (postData) => api.post('culture-posts/', postData),

```

Admin: MyCulture Moderator Page

Administrators are able to approve and reject post, enabling moderations of the MyCulture Page for any sensitive and unacceptable post.

```

// Approve or Reject post
const handleApprove = async (postId) => {
  try {
    await cultureModeratorAPI.approvePost(postId);
    setPosts(posts.filter((p) => p.id !== postId)); // remove post from list
  } catch (err) {
    console.error("Error approving post:", err);
  }
};

const handleReject = async (postId) => {
  try {
    await cultureModeratorAPI.rejectPost(postId);
    setPosts(posts.filter((p) => p.id !== postId)); // remove post from list
  } catch (err) {
    console.error("Error rejecting post:", err);
  }
};

```

cultureModeratorApi.js

Api call to get all post not yet moderated by users. Administrators are able to approve and reject post that needs to be align with the community guidelines

```

getAllPosts: (status = "pending") => {
  const params = new URLSearchParams();
  if (status) params.append("status", status);
  return api.get(`culture-posts/?${params.toString()}`);
},

/**
 * Approve a user-submitted post
 *
 * @param {number} postId - ID of the post to approve
 * @returns Promise with updated post data
 */
approvePost: (postId) => {
  return api.post(`culture-posts/${postId}/approve/`);
},

/**
 * Reject a user-submitted post
 *
 * @param {number} postId - ID of the post to reject
 * @returns Promise with updated post data
 */
rejectPost: (postId) => {
  return api.post(`culture-posts/${postId}/reject/`);
},

```

12.3 Communications skills Page

Provides resources and exercises to improve communication skills. Videos are stored with:

```

const CommunicationSkillsPage = () => {
  const [seeMoreCategories, setSeeMoreCategories] = useState({});

  const categories = {
    'Learning to understand others': [
      { id: 1, title: 'Active Listening Techniques', videoId: 'Yq5pJ0q3xuc' },
      { id: 2, title: 'Reading Body Language', videoId: '4jwUXV4QaTw' },
      { id: 3, title: 'Empathy in Communication', videoId: 'pi86Nr9Mdms', start: 470 },
      { id: 4, title: 'Understanding Different Perspectives', videoId: 'iueVZJVEms' }
    ],
    'Express yourself with clarity': [
      { id: 5, title: 'Speaking with Confidence', videoId: 'eVFzbxmKNUw' },
      { id: 6, title: 'Structuring Your Thoughts', videoId: 'Z_z-Q0agXZU' },
      { id: 7, title: 'Effective Presentation Skills', videoId: 'yod8RMq20kU' },
      { id: 8, title: 'Writing Clear Messages', videoId: 'prUqCd2R3Zw', start: 20 }
    ]
  };
};

```

12.4 Campus Map Page

Displays a campus map and interactive map features for navigation.

Has a feature that finds users location via button

```
const handleFindMe = () => {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      (pos) => {
        const userLoc = { lat: pos.coords.latitude, lng: pos.coords.longitude };
        setCurrentPosition(userLoc);

        // Center map on user's position
        if (mapRef.current) {
          mapRef.current.panTo(userLoc);
        }

        // Add "My Location" to dropdowns
        const myLoc = {
          id: "my_location",
          name: "My Location",
          lat: userLoc.lat,
          lng: userLoc.lng,
        };

        setLocations((prev) => {
          const exists = prev.find((loc) => loc.id === "my_location");
          if (exists) return prev.map((loc) => (loc.id === "my_location" ? myLoc : loc));
          return [...prev, myLoc];
        });
      }
    );
  }
};
```

As well as to find route for the users with locations of campus' most popular sites

```
const handleFindRoute = () => {
  if (!start || !destination) {
    alert("Please select both starting point and destination.");
    return;
  }

  const startLoc = locations.find((loc) => loc.id === start);
  const destLoc = locations.find((loc) => loc.id === destination);

  if (!startLoc || !destLoc) return;

  const directionsService = new window.google.maps.DirectionsService();
  directionsService.route(
    {
      origin: { lat: startLoc.lat, lng: startLoc.lng },
      destination: { lat: destLoc.lat, lng: destLoc.lng },
      travelMode: window.google.maps.TravelMode.WALKING,
    },
    (result, status) => {
      if (status === "OK") setDirections(result);
      else alert("Could not calculate route. Try again.");
    }
  );
};
```

12.5. Social Chatbox Page

Provides real-time chat functionality for users to communicate.

Users are able to send and Delete messages of their own. Admin are able to moderate the real-time chat by deleting any and all message from all users.

```
// Send message
const handleSend = async () => {
  if (input.trim() === "") return;

  try {
    const newMsg = await chatAPI.sendMessage(input.trim());
    setMessages((prev) => [...prev, newMsg]);
    setInput("");
  } catch (err) {
    console.error("Error sending message:", err);
    alert("Failed to send message.");
  }
};

// Delete message
const handleDelete = async (msg) => {
  if (!msg.is_current_user && !msg.is_admin_user) {
    alert("You can only delete your own messages!");
    return;
  }

  try {
    await chatAPI.deleteMessage(msg.id);
    setMessages((prev) => prev.filter((m) => m.id !== msg.id));
  } catch (err) {
    console.error("Error deleting message:", err);
    alert("Failed to delete message.");
  }
};
```

chatApi.js: API calls to the backend to load, send and delete messages

```
export const chatAPI = {
  // Get all messages
  getMessages: async () => {
    const response = await api.get("chat/messages/");
    return response.data;
  },

  // Send a message
  sendMessage: async (messageText) => {
    const response = await api.post("chat/messages/", {
      message_text: messageText,
    });
    return response.data;
  },

  // Delete a message
  deleteMessage: async (id) => {
    const response = await api.delete(`chat/messages/${id}/`);
    return response.data;
  },
}
```

12.6 We Need Page

Sends community requests or posts where users can contribute or respond.

Submitting requests first stores the request for moderations of admins. It will be displayed in the Admin We Need page that will then be process in a series of events that will either be approved, rejected or deleted by the administrator.

```

async function handleSubmit(e) {
  e.preventDefault();
  if (!isAuthenticated) {
    setError("Please sign in to submit a request.");
    return;
  }
  if (!requestText.trim() || !category) return;

  setPosting(true);
  setError("");
  try {
    const created = await WeNeedAPI.createRequest({
      category,
      text: requestText.trim(),
    });

    if (created.status === "approved") {
      setSubmittedRequests((prev) => [
        {
          id: created.id,
          name: created.student?.name || created.display_name || "Anonymous",
          category: created.category,
          request: created.text,
          date: created.created_at
            ? new Date(created.created_at).toLocaleString()
            : "Just now",
          likes: created.likes_count ?? 0,
          liked: false,
        },
        ...prev,
      ]);
    }
  }
}

```

If approved. The request is stored in the database for further processing for user's request.

We need api: Approve and reject Api calls as well as Update request

```

export const WeNeedAPI = {
  async getApprovedRequests() {
    const res = await fetch(`${API_BASE_URL}/requests/?status=approved`, {
      headers: jsonHeaders(),
    });
    if (!res.ok) throw new Error(`getApprovedRequests: ${res.status}`);
    return res.json();
  },
}

```

```

async function updateStatus(id, newStatus) {
  const prev = [...items];
  setItems((cur) => cur.map((it) => (it.id === id ? { ...it, status: newStatus } : it)));

  try {
    await WeNeedAPI.updateRequestStatus(id, newStatus);
  } catch (err) {
    console.error(err);
    setError("Failed to update status.");
    setItems(prev);
  }
}

```