**ai\_quiz\_backend/core/utils/ai\_quiz\_generator.py:**

import openai

import os

from docx import Document

import fitz # PyMuPDF

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

def extract\_text\_from\_pdf(file\_path: str) -> str:

doc = fitz.open(file\_path)

text = ""

for page in doc:

text += page.get\_text()

return text

def extract\_text\_from\_docx(file\_path: str) -> str:

doc = Document(file\_path)

return "\n".join([p.text for p in doc.paragraphs])

def read\_material\_content(material\_path: str) -> str:

if material\_path.endswith(".pdf"):

return extract\_text\_from\_pdf(material\_path)

elif material\_path.endswith(".docx"):

return extract\_text\_from\_docx(material\_path)

else:

return "Unsupported format for AI generation."

def generate\_quiz(text: str, level: str) -> str:

prompt = f"""

You are an AI teacher. Generate a multiple-choice quiz with 5 questions from the following study material.

Use this difficulty level: {level}.

Make the questions child-friendly and suitable for 7-year-old Malaysian students.

Material Content:

{text}

"""

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=[{

"role": "user",

"content": prompt

}],

temperature=0.7

)

return response.choices[0].message["content"]

**ai\_quiz\_backend /core/admin.py:**

from django.contrib import admin

from .models import \*

admin.site.register([Subject, Chapter, Subchapter, StudyMaterial, Quiz, UserProfile])

**ai\_quiz\_backend /core/apps.py:**

from django.apps import AppConfig

class CoreConfig(AppConfig):

    default\_auto\_field = 'django.db.models.BigAutoField'

    name = 'core'

    def ready(self):

        import core.signals  # connect signal

**ai\_quiz\_backend /core/models.py:**

from django.db import models

from django.contrib.auth.models import User

# Subject (e.g., Science)

class Subject(models.Model):

    name = models.CharField(max\_length=255)

    @classmethod

    def initialize\_defaults(cls):

        defaults = ['Mathematics', 'Science', 'Bahasa Malaysia', 'English']

        for name in defaults:

            cls.objects.get\_or\_create(name=name)

# Chapter under a subject (e.g., Chapter 1: Plants)

class Chapter(models.Model):

    subject = models.ForeignKey(Subject, on\_delete=models.CASCADE, related\_name="chapters")

    name = models.CharField(max\_length=255)

    def \_\_str\_\_(self):

        return f"{self.subject.name} - {self.name}"

# Subchapter under a chapter (e.g., Subtopic: Photosynthesis)

class Subchapter(models.Model):

    chapter = models.ForeignKey(Chapter, on\_delete=models.CASCADE, related\_name="subchapters")

    name = models.CharField(max\_length=255)

    def \_\_str\_\_(self):

        return f"{self.chapter.name} - {self.name}"

# Study material (PDF, image, etc.) linked to a subchapter

class StudyMaterial(models.Model):

    subchapter = models.ForeignKey(Subchapter, on\_delete=models.CASCADE, related\_name="materials")

    title = models.CharField(max\_length=255)

    document = models.FileField(upload\_to='materials/')

    uploaded\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"{self.title} ({self.subchapter.name})"

# AI-Generated Quiz

class Quiz(models.Model):

    LEVEL\_CHOICES = [

        ("Beginner", "Beginner"),

        ("Intermediate", "Intermediate"),

        ("Advanced", "Advanced")

    ]

    subchapter = models.ForeignKey(Subchapter, on\_delete=models.CASCADE, related\_name="quizzes")

    question = models.TextField()

    answer = models.TextField()

    level = models.CharField(max\_length=20, choices=LEVEL\_CHOICES)

    def \_\_str\_\_(self):

        return f"{self.level} Quiz: {self.question[:50]}..."

# User Profile - to track quiz results and recommendations

class UserProfile(models.Model):

    user = models.OneToOneField(User, on\_delete=models.CASCADE)

    completed\_quizzes = models.ManyToManyField(Quiz, blank=True)

    total\_score = models.IntegerField(default=0)

    recommended\_topics = models.TextField(blank=True)

    def \_\_str\_\_(self):

        return self.user.username

class QuizScore(models.Model):

    LEVEL\_CHOICES = [

        ('Beginner', 'Beginner'),

        ('Intermediate', 'Intermediate'),

        ('Advanced', 'Advanced'),

    ]

    student = models.ForeignKey(User, on\_delete=models.CASCADE)

    material = models.ForeignKey('StudyMaterial', on\_delete=models.CASCADE)

    score = models.FloatField()

    level = models.CharField(max\_length=20, choices=LEVEL\_CHOICES)

    time\_taken = models.DurationField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"{self.student.username} - {self.material.title} - {self.score}"

**ai\_quiz\_backend /core/serializeres.py:**

from rest\_framework import serializers

from django.contrib.auth.models import User

from .models import Subject, Chapter, Subchapter, StudyMaterial, Quiz, UserProfile, QuizScore

from rest\_framework\_simplejwt.serializers import TokenObtainPairSerializer

class SubjectSerializer(serializers.ModelSerializer):

    class Meta:

        model = Subject

        fields = ['id', 'name']

class ChapterSerializer(serializers.ModelSerializer):

    class Meta:

        model = Chapter

        fields = ['id', 'name', 'subject']

class SubchapterSerializer(serializers.ModelSerializer):

    class Meta:

        model = Subchapter

        fields = ['id', 'name', 'chapter']

class StudyMaterialSerializer(serializers.ModelSerializer):

    class Meta:

        model = StudyMaterial

        fields = ['id', 'title', 'document', 'uploaded\_at', 'subchapter']

    def validate\_document(self, file):

        allowed\_extensions = ['pdf', 'docx', 'jpg', 'png']

        ext = file.name.split('.')[-1].lower()

        if ext not in allowed\_extensions:

            raise serializers.ValidationError("Unsupported file type.")

        return file

class QuizSerializer(serializers.ModelSerializer):

    class Meta:

        model = Quiz

        fields = ['id', 'subchapter', 'question', 'answer', 'level']

class UserProfileSerializer(serializers.ModelSerializer):

    completed\_quizzes = QuizSerializer(many=True, read\_only=True)

    class Meta:

        model = UserProfile

        fields = ['id', 'user', 'completed\_quizzes', 'total\_score', 'recommended\_topics']

class UserSerializer(serializers.ModelSerializer):

    profile = UserProfileSerializer(read\_only=True)

    class Meta:

        model = User

        fields = ['id', 'username', 'email', 'profile']

class QuizScoreSerializer(serializers.ModelSerializer):

    class Meta:

        model = QuizScore

        fields = '\_\_all\_\_'

        read\_only\_fields = ['student', 'created\_at']

class CustomTokenObtainPairSerializer(TokenObtainPairSerializer):

    def validate(self, attrs):

        data = super().validate(attrs)

        data['role'] = 'admin' if self.user.is\_staff else 'student'

        return data

**ai\_quiz\_backend /core/view.py:**

from rest\_framework import viewsets, permissions

from rest\_framework.decorators import action, api\_view, permission\_classes

from rest\_framework.response import Response

from .models import Subject, Chapter, Subchapter, StudyMaterial, Quiz, UserProfile, QuizScore

from .serializers import (SubjectSerializer, ChapterSerializer, SubchapterSerializer,

                          StudyMaterialSerializer, QuizSerializer, UserProfileSerializer, QuizScoreSerializer)

from rest\_framework import status

from django.contrib.auth.models import User

from rest\_framework\_simplejwt.tokens import RefreshToken

from rest\_framework.parsers import MultiPartParser, FormParser

from rest\_framework.permissions import IsAuthenticated, IsAuthenticatedOrReadOnly

from core.models import StudyMaterial

from core.utils.ai\_quiz\_generator import read\_material\_content, generate\_quiz

import os

class SubjectViewSet(viewsets.ModelViewSet):

    queryset = Subject.objects.all()

    serializer\_class = SubjectSerializer

    permission\_classes = [IsAuthenticatedOrReadOnly]

    def create(self, request, \*args, \*\*kwargs):

        if not request.user.is\_staff:

            return Response({"error": "Only teachers can add subjects."}, status=403)

        return super().create(request, \*args, \*\*kwargs)

    def destroy(self, request, \*args, \*\*kwargs):

        if not request.user.is\_staff:

            return Response({"error": "Only teachers can delete subjects."}, status=403)

        return super().destroy(request, \*args, \*\*kwargs)

class ChapterViewSet(viewsets.ReadOnlyModelViewSet):

    queryset = Chapter.objects.all()

    serializer\_class = ChapterSerializer

    def get\_queryset(self):

        subject\_id = self.request.query\_params.get('subject')

        if subject\_id:

            return self.queryset.filter(subject\_\_id=subject\_id)

        return self.queryset.none()

class SubchapterViewSet(viewsets.ReadOnlyModelViewSet):

    queryset = Subchapter.objects.all()

    serializer\_class = SubchapterSerializer

    def get\_queryset(self):

        chapter\_id = self.request.query\_params.get('chapter')

        if chapter\_id:

            return self.queryset.filter(chapter\_\_id=chapter\_id)

        return self.queryset.none()

class StudyMaterialViewSet(viewsets.ModelViewSet):

    queryset = StudyMaterial.objects.all()

    serializer\_class = StudyMaterialSerializer

    parser\_classes = [MultiPartParser, FormParser]

    permission\_classes = [permissions.IsAuthenticated]

    def get\_queryset(self):

        subchapter\_id = self.request.query\_params.get('subchapter')

        if subchapter\_id:

            return self.queryset.filter(subchapter\_\_id=subchapter\_id)

        return self.queryset

class QuizViewSet(viewsets.ReadOnlyModelViewSet):

    queryset = Quiz.objects.all()

    serializer\_class = QuizSerializer

    def get\_queryset(self):

        subchapter\_id = self.request.query\_params.get('subchapter')

        level = self.request.query\_params.get('level')

        qset = self.queryset

        if subchapter\_id:

            qset = qset.filter(subchapter\_\_id=subchapter\_id)

        if level:

            qset = qset.filter(level=level)

        return qset

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

    @permission\_classes([IsAuthenticated])

    def generate(self, request, pk=None):

        material = StudyMaterial.objects.filter(subchapter=pk).first()

        if not material:

            return Response({"error": "Study material not found."}, status=404)

        level = request.data.get('level', 'Beginner')

        file\_path = material.document.path

        content = read\_material\_content(file\_path)

        if not content.strip():

            return Response({"error": "No readable content in the file."}, status=400)

        # Generate quiz using AI

        quiz = generate\_quiz(content, level)

        # Optionally, save quiz to database if needed

        quiz\_object = Quiz.objects.create(subchapter=material.subchapter, level=level)

        for q in quiz:  # Assuming quiz is a list of questions

            # You can add question objects here if needed

            pass

        return Response({"quiz": quiz, "message": "Quiz generated successfully"})

class UserProfileViewSet(viewsets.ViewSet):

    permission\_classes = [permissions.IsAuthenticated]

    def list(self, request):

        profile = UserProfile.objects.get(user=request.user)

        serializer = UserProfileSerializer(profile)

        return Response(serializer.data)

@api\_view(['POST'])

def register\_user(request):

    data = request.data

    try:

        user = User.objects.create\_user(

            username=data['username'],

            email=data['email'],

            password=data['password']

        )

        # Attach role to profile

        UserProfile.objects.create(user=user, role=data.get('role', 'student'))

        refresh = RefreshToken.for\_user(user)

        return Response({

            'refresh': str(refresh),

            'access': str(refresh.access\_token),

            'user': {

                'id': user.id,

                'username': user.username,

                'email': user.email,

                'role': data.get('role', 'student')

            }

        }, status=status.HTTP\_201\_CREATED)

    except Exception as e:

        return Response({'error': str(e)}, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(["POST"])

@permission\_classes([IsAuthenticated])

def generate\_quiz\_view(request, material\_id):

    level = request.data.get("level", "Beginner")

    try:

        material = StudyMaterial.objects.get(id=material\_id)

        content = read\_material\_content(material.document.path)

        if not content.strip():

            return Response({"error": "No readable content found"}, status=400)

        # Generate and parse quiz

        quiz\_text = generate\_quiz(content, level)

        # Example parsing logic (adjust based on your AI's output format)

        questions = []

        current\_question = {}

        for line in quiz\_text.split('\n'):

            if line.startswith('Q:'):

                current\_question['question'] = line[2:].strip()

            elif line.startswith('A:'):

                current\_question['answer'] = line[2:].strip()

                questions.append(current\_question)

                current\_question = {}

        # Save to database

        for q in questions:

            Quiz.objects.create(

                subchapter=material.subchapter,

                question=q.get('question', ''),

                answer=q.get('answer', ''),

                level=level

            )

        return Response({"message": "Quiz generated and saved successfully"})

    except StudyMaterial.DoesNotExist:

        return Response({"error": "Material not found"}, status=404)

    except Exception as e:

        return Response({"error": str(e)}, status=500)

@api\_view(["POST"])

@permission\_classes([IsAuthenticated])

def submit\_quiz\_score(request):

    try:

        data = request.data.copy()

        data["student"] = request.user.id

        # Get correct answers from database

        material = StudyMaterial.objects.get(id=data['material'])

        quizzes = Quiz.objects.filter(subchapter=material.subchapter)

        correct\_answers = {q.id: q.answer for q in quizzes}

        # Calculate score

        score = sum(

            1 for q\_id, answer in data['answers'].items()

            if str(answer) == correct\_answers.get(int(q\_id))

        )

        # Save score

        serializer = QuizScoreSerializer(data={

            'student': request.user.id,

            'material': data['material'],

            'score': score,

            'level': data['level'],

            'time\_taken': data['time\_taken']

        })

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data)

        return Response(serializer.errors, status=400)

    except StudyMaterial.DoesNotExist:

        return Response({"error": "Material not found"}, status=404)

@api\_view(["GET"])

@permission\_classes([IsAuthenticated])

def leaderboard\_view(request, material\_id):

    scores = QuizScore.objects.filter(material\_id=material\_id).order\_by("-score", "time\_taken")

    # Optional: Filter based on level

    level = request.query\_params.get('level')

    if level:

        scores = scores.filter(level=level)

    serializer = QuizScoreSerializer(scores, many=True)

    return Response(serializer.data)

@api\_view(["GET"])

@permission\_classes([IsAuthenticated])

def student\_report(request):

    scores = QuizScore.objects.filter(student=request.user)

    report = {}

    for score in scores:

        title = score.material.title

        if title not in report:

            report[title] = {"attempts": 0, "avg\_score": 0, "levels": []}

        report[title]["attempts"] += 1

        report[title]["avg\_score"] += score.score

        report[title]["levels"].append(score.level)

    for title in report:

        report[title]["avg\_score"] /= report[title]["attempts"]

        # Basic AI recommendation

        if report[title]["avg\_score"] < 70:

            report[title]["recommendation"] = "📌 Review this topic again."

        else:

            report[title]["recommendation"] = "✅ Doing well!"

    return Response(report)

**ai\_quiz\_backend /core/urls.py:**

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import (

    SubjectViewSet, ChapterViewSet, SubchapterViewSet,

    StudyMaterialViewSet, QuizViewSet, UserProfileViewSet,

    register\_user, generate\_quiz\_view, submit\_quiz\_score,

    leaderboard\_view, student\_report

)

router = DefaultRouter()

router.register('subjects', SubjectViewSet, basename='subject')  # Add this

router.register('chapters', ChapterViewSet, basename='chapter')

router.register('subchapters', SubchapterViewSet, basename='subchapter')

router.register('materials', StudyMaterialViewSet, basename='material')

router.register('quizzes', QuizViewSet, basename='quiz')

router.register('profile', UserProfileViewSet, basename='profile')

urlpatterns = [

    path('', include(router.urls)),

    path('register/', register\_user, name='register'),  # Registration endpoint

    path('generate-quiz/<int:material\_id>/', generate\_quiz\_view, name='generate-quiz'),

    path('submit-score/', submit\_quiz\_score, name='submit-score'),

    path('leaderboard/<int:material\_id>/', leaderboard\_view, name='leaderboard'),

    path('student-report/', student\_report, name='student-report'),

]

**ai\_quiz\_backend /core/signals.py:**

from django.db.models.signals import post\_save

from django.dispatch import receiver

from django.contrib.auth.models import User

from .models import UserProfile

@receiver(post\_save, sender=User)

def create\_user\_profile(sender, instance, created, \*\*kwargs):

    if created:

        UserProfile.objects.create(user=instance)

@receiver(post\_save, sender=User)

def save\_user\_profile(sender, instance, \*\*kwargs):

    instance.userprofile.save()

**ai\_quiz\_backend /ai\_quiz\_backend/setting.py:**

"""

Django settings for ai\_quiz\_backend project.

Generated by 'django-admin startproject' using Django 5.2.

For more information on this file, see

https://docs.djangoproject.com/en/5.2/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.2/ref/settings/

"""

from pathlib import Path

from datetime import timedelta

import os

from dotenv import load\_dotenv

load\_dotenv()

OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-tnknbnup^(m^7)w-$\*ve=6&je1nl%@%rtgh41+y^q@&q8hu%c3'

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

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'rest\_framework',

    'corsheaders',

    'core',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    'corsheaders.middleware.CorsMiddleware',

]

CORS\_ALLOWED\_ORIGINS = [

    "http://localhost:5173",  # React frontend

]

# JWT Auth

REST\_FRAMEWORK = {

    'DEFAULT\_AUTHENTICATION\_CLASSES': (

        'rest\_framework\_simplejwt.authentication.JWTAuthentication',

    ),

}

SIMPLE\_JWT = {

    "ACCESS\_TOKEN\_LIFETIME": timedelta(minutes=30),

    "REFRESH\_TOKEN\_LIFETIME": timedelta(days=1),

    "ROTATE\_REFRESH\_TOKENS": True,

    "BLACKLIST\_AFTER\_ROTATION": True,

    "AUTH\_HEADER\_TYPES": ("Bearer",),

    'TOKEN\_OBTAIN\_SERIALIZER': 'core.serializers.CustomTokenObtainPairSerializer',

}

# Media Files

MEDIA\_URL = '/media/'

MEDIA\_ROOT = BASE\_DIR / 'media'

ROOT\_URLCONF = 'ai\_quiz\_backend.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'ai\_quiz\_backend.wsgi.application'

# Database

# https://docs.djangoproject.com/en/5.2/ref/settings/#databases

# Database (PostgreSQL)

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'ai\_quiz',

        'USER': 'postgres',

        'PASSWORD': '1234',

        'HOST': 'localhost',

        'PORT': '5432',

    }

}

# Password validation

# https://docs.djangoproject.com/en/5.2/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/5.2/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC\_URL = 'static/'

# Default primary key field type

# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

AUTH\_USER\_MODEL = 'auth.User'

**ai\_quiz\_backend /ai\_quiz\_backend/urls.py:**

"""

URL configuration for ai\_quiz\_backend project.

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

    https://docs.djangoproject.com/en/5.2/topics/http/urls/

Examples:

Function views

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

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

Class-based views

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

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

Including another URLconf

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

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

"""

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

from rest\_framework\_simplejwt.views import (

    TokenObtainPairView,

    TokenRefreshView,

    TokenBlacklistView,

)

urlpatterns = [

    path('admin/', admin.site.urls),

    path('api/', include('core.urls')),

    # JWT Token endpoints

    path('api/token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),  # Login

    path('api/token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),  # Refresh token

    path('api/token/logout/', TokenBlacklistView.as\_view(), name='token\_blacklist'),  # Logout

]

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**ai\_quiz\_backend/.env:**

OPENAI\_API\_KEY="ssk-proj-lp4jCf9Jin3e97Kiab\_LcMTV9D2TE6lStRw9AHCmT6jEcQSZPSFDUaaM9RIMDjEm-OIBzX-AEQT3BlbkFJ0P2mJ53p-nj0DztDgciJeQUO4n5TyP7wO508yIvpypKRErLHgrTSCRL6mG0N7PlYmk-RNA6dcA"