AI Smart Academic Project Evaluation System

30-Day Implementation Plan

Complete Day-by-Day Guide to Build Your Capstone Project

🗓️ Plan Overview

This 30-day plan breaks down the entire project into manageable daily tasks. Each day has specific goals, code to write, and checkpoints to ensure you're on track. Follow this plan step-by-step, and you'll have a complete, working AI project evaluation system.

Key Milestones

- Day 5: Backend API with authentication working
- Day 10: Database and file upload functional
- Day 15: Core AI evaluation engine complete
- Day 20: Frontend with all major features
- Day 25: Complete integration and testing
- Day 30: Deployed and production-ready system

Daily Time Commitment

Estimated: 4-6 hours per day

Weekdays: Focus on coding and implementation

Weekends: Testing, bug fixing, and catch-up

🗓️ Week 1: Foundation & Backend Setup (Days 1-7)

Day 1: Project Setup & Environment Configuration

Goal: Set up your development environment and create the project structure

Morning Tasks (2-3 hours)

- Install Python 3.11+, Node.js 18+, PostgreSQL, Redis
- Set up VS Code with extensions (Python, ESLint, Prettier)
- Create GitHub repository for version control
- Create main project folder structure

```
# Create project structure
mkdir ai-project-evaluation-system
cd ai-project-evaluation-system

# Create backend
mkdir -p backend/app/{api,models,schemas,services,ai_engine,utils,database}
touch backend/requirements.txt backend/.env

# Create frontend
npx create-react-app frontend
cd frontend
npm install react-router-dom axios

# Initialize Git
git init
git add .
git commit -m "Initial project setup"
```

Afternoon Tasks (2-3 hours)

- Create requirements.txt with all dependencies
- Set up virtual environment for Python
- Install all backend dependencies
- Create .gitignore file
- Set up PostgreSQL database

```
# backend/requirements.txt
fastapi==0.104.1
uvicorn[standard]==0.24.0
sqlalchemy==2.0.23
psycopg2-binary==2.9.9
pydantic==2.5.0
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
python-multipart==0.0.6
python-dotenv==1.0.0
```

```
pytest==7.4.3
```

✓Day 1 Checkpoint

- Project folders created
- All tools installed and working
- Git repository initialized
- Dependencies installed

Day 2: Database Models & Configuration

Goal: Create all database models and configure database connection

Morning Tasks (2-3 hours)

- Create database connection file
- Create User model with roles (Student, Faculty, Admin)
- Create Project model
- Create Evaluation model

```
# backend/app/database/connection.py
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
import os
from dotenv import load_dotenv

load_dotenv()

DATABASE_URL = os.getenv("DATABASE_URL", "postgresql://user:password@localhost/evaldb")

engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()


# backend/app/models/user.py
from sqlalchemy import Column, Integer, String, Enum, DateTime
from sqlalchemy.orm import relationship
from app.database.connection import Base
from datetime import datetime
import enum
```

```
class UserRole(str, enum.Enum):
    STUDENT = "student"
    FACULTY = "faculty"
    ADMIN = "admin"

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True, nullable=False)
    username = Column(String, unique=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    full_name = Column(String)
    role = Column(Enum(UserRole), nullable=False)
    department = Column(String)
    created_at = Column(DateTime, default=datetime.utcnow)

    projects = relationship("Project", back_populates="student")
```

Afternoon Tasks (2-3 hours)

- Create config.py with all settings
- Create database initialization script
- Test database connection
- Create first migration

✓Day 2 Checkpoint

- All models created
- Database connection working
- Can create tables in PostgreSQL

Day 3: Authentication System

Goal: Implement complete JWT-based authentication

Morning Tasks (3 hours)

- Create security utility functions (password hashing)
- Create JWT token generation and verification
- Create authentication schemas (Pydantic models)
- Create auth dependencies

```
# backend/app/utils/security.py
from passlib.context import CryptContext
from jose import jwt, JWTError
from datetime import datetime, timedelta
import os
```

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
SECRET_KEY = os.getenv("SECRET_KEY", "your-secret-key-change-in-production")
ALGORITHM = "HS256"

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)

def create_access_token(data: dict, expires_delta: timedelta = None):
    to_encode = data.copy()
    expire = datetime.utcnow() + (expires_delta or timedelta(minutes=15))
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def verify_token(token: str):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        return payload
    except JWTError:
        return None
```

Afternoon Tasks (2-3 hours)

- Create authentication API endpoints (register, login)
- Implement get current user endpoint
- Test authentication with Postman/Thunder Client
- Create sample test users

```
# backend/app/api/auth.py
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from app.database.connection import get_db
from app.models.user import User
from app.schemas.user import UserCreate, UserResponse, Token
from app.utils.security import get_password_hash, verify_password, create_access_token
from datetime import timedelta

router = APIRouter()

@router.post("/register", response_model=UserResponse)
def register(user: UserCreate, db: Session = Depends(get_db)):
    # Check if user exists
    if db.query(User).filter(User.email == user.email).first():
        raise HTTPException(status_code=400, detail="Email already registered")

    # Create new user
    db_user = User(
        email=user.email,
        username=user.username,
        hashed_password=get_password_hash(user.password),
```

```
        full_name=user.full_name,
        role=user.role
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

@router.post("/login", response_model=Token)
def login(username: str, password: str, db: Session = Depends(get_db)):
    user = db.query(User).filter(User.username == username).first()
    if not user or not verify_password(password, user.hashed_password):
        raise HTTPException(status_code=401, detail="Invalid credentials")

    access_token = create_access_token(
        data={"sub": user.username, "role": user.role},
        expires_delta=timedelta(days=7)
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

✅Day 3 Checkpoint

- Can register new users
- Can login and receive JWT token
- Token verification working

Day 4: Main FastAPI Application & Basic Endpoints

Goal: Set up the main FastAPI app and create basic CRUD endpoints

Morning Tasks (2-3 hours)

- Create main.py with FastAPI app initialization
- Add CORS middleware
- Include authentication router
- Create health check endpoint
- Test server runs successfully

```
# backend/app/main.py
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.api import auth
from app.database.connection import engine, Base

# Create database tables
Base.metadata.create_all(bind=engine)

app = FastAPI(
    title="AI Project Evaluation System",
    description="Automated academic project evaluation with AI",
    version="1.0.0"
)

# CORS configuration
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)

# Include routers
app.include_router(auth.router, prefix="/api/auth", tags=["Authentication"])

@app.get("/")
def root():
    return {"message": "AI Project Evaluation System API", "status": "running"}

@app.get("/health")
def health_check():
    return {"status": "healthy"}

# Run with: uvicorn app.main:app --reload
```

Afternoon Tasks (2-3 hours)

- Create user management endpoints (get user, update profile)
- Create project schemas (Pydantic models)

- Create basic project endpoints structure
- Test all endpoints with API client

✅Day 4 Checkpoint

- FastAPI server running on localhost:8000
- Can access /docs for Swagger UI
- Authentication endpoints working
- All endpoints returning proper responses

Day 5: File Upload System

Goal: Implement file upload functionality for projects

Morning Tasks (3 hours)

- Create file service for handling uploads
- Create upload directory structure
- Implement file validation (size, type)
- Create file storage logic

```
# backend/app/services/file_service.py
import os
import shutil
from pathlib import Path
from fastapi import UploadFile, HTTPException
from typing import Tuple
import uuid

UPLOAD_DIR = Path("uploads")
UPLOAD_DIR.mkdir(exist_ok=True)

ALLOWED_CODE_EXTENSIONS = {".py", ".js", ".java", ".cpp", ".c", ".zip"}
ALLOWED_DOC_EXTENSIONS = {".pdf", ".md", ".txt", ".docx"}
MAX_FILE_SIZE = 50 * 1024 * 1024  # 50MB

class FileService:
    @staticmethod
    def validate_file(file: UploadFile, allowed_extensions: set) -> None:
        ext = Path(file.filename).suffix.lower()
        if ext not in allowed_extensions:
            raise HTTPException(400, f"File type {ext} not allowed")

    @staticmethod
    async def save_file(file: UploadFile, subfolder: str) -> str:
        file_id = str(uuid.uuid4())
        ext = Path(file.filename).suffix
        filename = f"{file_id}{ext}"
```

```
        file_path = UPLOAD_DIR / subfolder / filename
        file_path.parent.mkdir(parents=True, exist_ok=True)

        with file_path.open("wb") as buffer:
            shutil.copyfileobj(file.file, buffer)

        return str(file_path)

    @staticmethod
    async def save_project_files(
        code_file: UploadFile,
        doc_file: UploadFile
    ) -> Tuple[str, str]:
        # Validate files
        FileService.validate_file(code_file, ALLOWED_CODE_EXTENSIONS)
        FileService.validate_file(doc_file, ALLOWED_DOC_EXTENSIONS)

        # Save files
        code_path = await FileService.save_file(code_file, "code")
        doc_path = await FileService.save_file(doc_file, "docs")

        return code_path, doc_path
```

Afternoon Tasks (2-3 hours)

- Create project upload API endpoint
- Integrate file service with project creation
- Test file upload with Postman
- Verify files are saved correctly

```
# backend/app/api/projects.py
from fastapi import APIRouter, Depends, UploadFile, File, Form, HTTPException
from sqlalchemy.orm import Session
from app.database.connection import get_db
from app.models.project import Project
from app.services.file_service import FileService
from app.utils.dependencies import get_current_user

router = APIRouter()

@router.post("/upload")
async def upload_project(
    title: str = Form(...),
    description: str = Form(""),
    programming_language: str = Form("python"),
    code_file: UploadFile = File(...),
    doc_file: UploadFile = File(...),
    current_user = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    # Save files
    code_path, doc_path = await FileService.save_project_files(code_file, doc_file)
```

```
# Create project record
project = Project(
    title=title,
    description=description,
    programming_language=programming_language,
    student_id=current_user.id,
    code_file_path=code_path,
    doc_file_path=doc_path,
    status="uploaded"
)

db.add(project)
db.commit()
db.refresh(project)

return {
    "id": project.id,
    "title": project.title,
    "status": project.status,
    "message": "Project uploaded successfully"
}
```

✅Day 5 Checkpoint

- Can upload files via API
- Files are validated and saved
- Project record created in database
- File paths stored correctly

 WEEK 1 COMPLETE! Backend foundation is solid.

🗓️ Week 2: AI/ML Engine Development (Days 8-14)

Day 8: Code Quality Analyzer - Part 1

Goal: Build the code quality analysis component

Morning Tasks (3 hours)

- Install analysis libraries (radon, pylint)
- Create CodeAnalyzer class structure
- Implement complexity analysis using Radon
- Test with sample Python files

```
# backend/app/ai_engine/code_analyzer.py
import radon.complexity as radon_cc
import radon.metrics as radon_metrics
from pylint import epylint as lint
import ast

class CodeAnalyzer:
    def __init__(self):
        self.weights = {
            'complexity': 0.3,
            'maintainability': 0.3,
            'quality': 0.4
        }

    def analyze(self, file_path: str) -> dict:
        """Main analysis method"""
        try:
            with open(file_path, 'r', encoding='utf-8') as f:
                code = f.read()

            # Run all analysis
            complexity_score = self._analyze_complexity(code)
            maintainability_score = self._analyze_maintainability(code)
            quality_score = self._analyze_quality(file_path)

            # Calculate final score
            final_score = (
                complexity_score * self.weights['complexity'] +
                maintainability_score * self.weights['maintainability'] +
                quality_score * self.weights['quality']
            )

            return {
                'final_score': round(final_score, 2),
                'complexity': round(complexity_score, 2),
                'maintainability': round(maintainability_score, 2),
                'quality': round(quality_score, 2),
                'details': self._get_code_details(code)
            }
```

```
        except Exception as e:
            return {
                'final_score': 0,
                'error': str(e)
            }

    def _analyze_complexity(self, code: str) -> float:
        """Calculate complexity using Radon"""
        try:
            results = radon_cc.cc_visit(code)
            if not results:
                return 100.0

            avg_complexity = sum(r.complexity for r in results) / len(results)
            # Lower complexity is better, convert to score
            score = max(0, 100 - (avg_complexity * 10))
            return score
        except:
            return 50.0

    def _analyze_maintainability(self, code: str) -> float:
        """Calculate maintainability index"""
        try:
            mi_score = radon_metrics.mi_visit(code, multi=True)
            return min(100, max(0, mi_score))
        except:
            return 50.0
```

Afternoon Tasks (2-3 hours)

- Implement quality analysis using Pylint
- Add helper methods for code metrics
- Create test files for different complexity levels
- Test analyzer with real Python projects

✓Day 8 Checkpoint

- CodeAnalyzer class working
- Can analyze Python files
- Returns complexity and maintainability scores

Day 9: Documentation Evaluator

Goal: Create NLP-based documentation evaluation

Morning Tasks (3 hours)

- Install sentence-transformers library
- Create DocumentationEvaluator class
- Implement completeness checker

- Implement clarity assessment

```
# backend/app/ai_engine/doc_evaluator.py
from sentence_transformers import SentenceTransformer
import re

class DocumentationEvaluator:
    def __init__(self):
        self.model = SentenceTransformer('all-MiniLM-L6-v2')
        self.required_sections = [
            'introduction', 'installation', 'usage',
            'features', 'requirements', 'examples'
        ]

    def evaluate(self, doc_file_path: str) -> dict:
        """Evaluate documentation quality"""
        try:
            with open(doc_file_path, 'r', encoding='utf-8') as f:
                content = f.read()

            completeness = self._check_completeness(content)
            clarity = self._assess_clarity(content)
            structure = self._evaluate_structure(content)

            final_score = (
                completeness * 0.4 +
                clarity * 0.3 +
                structure * 0.3
            )

            return {
                'final_score': round(final_score, 2),
                'completeness': round(completeness, 2),
                'clarity': round(clarity, 2),
                'structure': round(structure, 2),
                'word_count': len(content.split()),
                'missing_sections': self._find_missing_sections(content)
            }
        except Exception as e:
            return {
                'final_score': 0,
                'error': str(e)
            }

    def _check_completeness(self, content: str) -> float:
        """Check for required sections"""
        content_lower = content.lower()
        found = sum(1 for sec in self.required_sections if sec in content_lower)
        return (found / len(self.required_sections)) * 100

    def _assess_clarity(self, content: str) -> float:
        """Assess readability"""
```

```
        sentences = [s.strip() for s in content.split('.') if s.strip()]
        if not sentences:
            return 0

        avg_length = sum(len(s.split()) for s in sentences) / len(sentences)

        # Optimal: 15-20 words per sentence
        if 15 <= avg_length <= 20:
            return 100.0
        elif avg_length < 10:
            return 60.0
        elif avg_length > 30:
            return 50.0
        return 80.0

    def _evaluate_structure(self, content: str) -> float:
        """Check document structure"""
        has_headings = bool(re.search(r'#|\*\*', content))
        has_code = bool(re.search(r'```|    ', content))
        has_lists = bool(re.search(r'^[-*]\s', content, re.MULTILINE))

        score = sum([has_headings, has_code, has_lists]) / 3 * 100
        return score

    def _find_missing_sections(self, content: str) -> list:
        content_lower = content.lower()
        return [s for s in self.required_sections if s not in content_lower]
```

Afternoon Tasks (2-3 hours)

- Test with various documentation formats (PDF, Markdown)
- Add support for PDF parsing (if needed)
- Fine-tune scoring weights
- Create sample documentation for testing

✅Day 9 Checkpoint

- Documentation evaluator working
- Can assess completeness and clarity
- Returns detailed feedback

Day 10: Plagiarism Detector

Goal: Implement semantic similarity-based plagiarism detection

Morning Tasks (3 hours)

- Create PlagiarismDetector class
- Implement embedding generation
- Implement similarity calculation
- Test with sample code pairs

```
# backend/app/ai_engine/plagiarism_detector.py
from sentence_transformers import SentenceTransformer, util
import numpy as np

class PlagiarismDetector:
    def __init__(self):
        self.model = SentenceTransformer('all-MiniLM-L6-v2')
        self.threshold = 0.75  # 75% similarity = flagged

    def detect(self, current_code: str, existing_projects: list) -> dict:
        """
        Compare current code with existing projects

        Args:
            current_code: Code to check
            existing_projects: List of dicts with 'id' and 'code'
        """
        if not existing_projects:
            return {
                'originality_score': 100.0,
                'flagged': False,
                'similar_projects': []
            }

        # Generate embedding for current code
        current_embedding = self.model.encode(
            current_code,
            convert_to_tensor=True
        )

        similarities = []
        for project in existing_projects:
            project_embedding = self.model.encode(
                project['code'],
                convert_to_tensor=True
            )

            similarity = util.cos_sim(current_embedding, project_embedding)

            similarities.append({
                'project_id': project['id'],
                'student_name': project.get('student_name', 'Unknown'),
```

```
                'similarity': float(similarity)
            })

        # Sort by similarity
        similarities.sort(key=lambda x: x['similarity'], reverse=True)

        max_similarity = similarities[0]['similarity'] if similarities else 0
        originality_score = (1 - max_similarity) * 100

        return {
            'originality_score': round(originality_score, 2),
            'flagged': max_similarity > self.threshold,
            'max_similarity_percent': round(max_similarity * 100, 2),
            'similar_projects': similarities[:5],
            'risk_level': self._get_risk_level(max_similarity)
        }

    def _get_risk_level(self, similarity: float) -> str:
        if similarity >= 0.90:
            return "HIGH"
        elif similarity >= 0.75:
            return "MEDIUM"
        elif similarity >= 0.60:
            return "LOW"
        return "MINIMAL"
```

Afternoon Tasks (2-3 hours)

- Integrate with database to fetch existing projects
- Optimize performance for large datasets
- Test with similar and dissimilar code
- Add caching for embeddings

✓Day 10 Checkpoint

- Plagiarism detector functional
- Can compare code semantically
- Returns similarity scores and risk levels

Day 11: AI Feedback Generator

Goal: Integrate LLM for generating personalized feedback

Morning Tasks (3 hours)

- Sign up for OpenAI API (or alternative)
- Create FeedbackGenerator class
- Design effective prompts
- Implement feedback generation

```
# backend/app/ai_engine/feedback_generator.py
from openai import OpenAI
import os

class FeedbackGenerator:
    def __init__(self):
        self.client = OpenAI(api_key=os.getenv('OPENAI_API_KEY'))
        self.model = "gpt-3.5-turbo"  # or gpt-4 for better quality

    def generate(self, evaluation_data: dict) -> str:
        """Generate personalized feedback"""
        prompt = self._create_prompt(evaluation_data)

        try:
            response = self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {
                        "role": "system",
                        "content": "You are an experienced computer science professor "
                                   "providing constructive feedback on student projects."
                    },
                    {
                        "role": "user",
                        "content": prompt
                    }
                ],
                temperature=0.7,
                max_tokens=500
            )

            return response.choices[0].message.content

        except Exception as e:
            return self._generate_fallback_feedback(evaluation_data)

    def _create_prompt(self, data: dict) -> str:
        return f"""
Provide constructive feedback for a student project:

Project: {data.get('project_title', 'Untitled')}

Scores:
- Code Quality: {data['code_score']}/100
  * Complexity: {data.get('complexity', 0)}/100
  * Maintainability: {data.get('maintainability', 0)}/100
- Documentation: {data['doc_score']}/100
  * Completeness: {data.get('completeness', 0)}/100
  * Clarity: {data.get('clarity', 0)}/100
- Originality: {data['originality_score']}/100

Please provide:
1. Brief overall assessment (2-3 sentences)
```

```
2. Key strengths (2-3 points)
3. Areas for improvement (2-3 specific points)
4. Actionable recommendations

Keep feedback encouraging yet honest.
"""

    def _generate_fallback_feedback(self, data: dict) -> str:
        performance = self._get_performance_level(data.get('final_score', 0))
        return f"""
Project Evaluation Summary:

Your project demonstrates {performance} performance with a final score of
{data.get('final_score', 0)}/100.

Key Metrics:
- Code Quality: {data['code_score']}/100
- Documentation: {data['doc_score']}/100
- Originality: {data['originality_score']}/100

Please review these scores and consult with your instructor for detailed feedback.
"""

    def _get_performance_level(self, score: float) -> str:
        if score >= 90:
            return "excellent"
        elif score >= 75:
            return "good"
        elif score >= 60:
            return "satisfactory"
        return "needs improvement"
```

Afternoon Tasks (2-3 hours)

- Test feedback generation with various scores
- Refine prompts for better quality
- Implement fallback for API failures
- Add cost tracking for API usage

✓Day 11 Checkpoint

- Feedback generator working
- Produces quality, personalized feedback
- Has fallback mechanism

Day 12: Grading Engine & Integration

Goal: Complete the grading system and integrate all AI components

Morning Tasks (3 hours)

- Create GradingEngine class
- Implement weighted score calculation
- Create letter grade mapping
- Create main evaluation orchestrator

```
# backend/app/ai_engine/grading_engine.py

class GradingEngine:
    def __init__(self):
        self.weights = {
            'code_quality': 0.40,
            'documentation': 0.30,
            'originality': 0.30
        }

        self.grade_scale = {
            'A+': (95, 100), 'A': (90, 94), 'A-': (85, 89),
            'B+': (80, 84), 'B': (75, 79), 'B-': (70, 74),
            'C+': (65, 69), 'C': (60, 64), 'C-': (55, 59),
            'D': (50, 54), 'F': (0, 49)
        }

    def calculate_final_score(self, scores: dict) -> dict:
        """Calculate weighted final score"""
        final_score = (
            scores['code_score'] * self.weights['code_quality'] +
            scores['doc_score'] * self.weights['documentation'] +
            scores['originality_score'] * self.weights['originality']
        )

        letter_grade = self._get_letter_grade(final_score)

        return {
            'final_score': round(final_score, 2),
            'letter_grade': letter_grade,
            'breakdown': {
                'code_quality': {
                    'score': scores['code_score'],
                    'weight': self.weights['code_quality'],
                    'contribution': round(scores['code_score'] *
self.weights['code_quality'], 2)
                },
                'documentation': {
                    'score': scores['doc_score'],
                    'weight': self.weights['documentation'],
                    'contribution': round(scores['doc_score'] *
self.weights['documentation'], 2)
```

```
                },
                'originality': {
                    'score': scores['originality_score'],
                    'weight': self.weights['originality'],
                    'contribution': round(scores['originality_score'] *
self.weights['originality'], 2)
                }
            }
        }

    def _get_letter_grade(self, score: float) -> str:
        for grade, (min_score, max_score) in self.grade_scale.items():
            if min_score <= score <= max_score:
                return grade
        return 'F'


# backend/app/ai_engine/evaluator.py
from app.ai_engine.code_analyzer import CodeAnalyzer
from app.ai_engine.doc_evaluator import DocumentationEvaluator
from app.ai_engine.plagiarism_detector import PlagiarismDetector
from app.ai_engine.feedback_generator import FeedbackGenerator
from app.ai_engine.grading_engine import GradingEngine

class ProjectEvaluator:
    def __init__(self):
        self.code_analyzer = CodeAnalyzer()
        self.doc_evaluator = DocumentationEvaluator()
        self.plagiarism_detector = PlagiarismDetector()
        self.feedback_generator = FeedbackGenerator()
        self.grading_engine = GradingEngine()

    def evaluate_project(self, project_data: dict) -> dict:
        """Complete evaluation pipeline"""
        # Step 1: Analyze code
        code_results = self.code_analyzer.analyze(project_data['code_file_path'])

        # Step 2: Evaluate documentation
        doc_results = self.doc_evaluator.evaluate(project_data['doc_file_path'])

        # Step 3: Check plagiarism
        plagiarism_results = self.plagiarism_detector.detect(
            self._read_file(project_data['code_file_path']),
            project_data.get('existing_projects', [])
        )

        # Step 4: Calculate final grade
        scores = {
            'code_score': code_results['final_score'],
            'doc_score': doc_results['final_score'],
            'originality_score': plagiarism_results['originality_score']
        }
```

```
        grading_results = self.grading_engine.calculate_final_score(scores)

        # Step 5: Generate feedback
        feedback_data = {
            'project_title': project_data['title'],
            **scores,
            'complexity': code_results.get('complexity', 0),
            'maintainability': code_results.get('maintainability', 0),
            'completeness': doc_results.get('completeness', 0),
            'clarity': doc_results.get('clarity', 0),
            'risk_level': plagiarism_results.get('risk_level', 'UNKNOWN'),
            'final_score': grading_results['final_score']
        }

        ai_feedback = self.feedback_generator.generate(feedback_data)

        return {
            'code_results': code_results,
            'doc_results': doc_results,
            'plagiarism_results': plagiarism_results,
            'grading_results': grading_results,
            'ai_feedback': ai_feedback
        }

    def _read_file(self, file_path: str) -> str:
        with open(file_path, 'r', encoding='utf-8') as f:
            return f.read()
```

Afternoon Tasks (2-3 hours)

- Create evaluation API endpoint
- Test complete evaluation pipeline
- Debug any integration issues
- Verify all scores are calculated correctly

✓Day 12 Checkpoint

- Complete evaluation pipeline working
- All AI components integrated
- Returns comprehensive results

 WEEK 2 COMPLETE! AI Engine is functional.

? Week 3: Frontend Development (Days 15-21)

Day 15: React Setup & Authentication UI

Goal: Set up React frontend with authentication

Morning Tasks (3 hours)

- Configure Tailwind CSS
- Create folder structure
- Create AuthContext
- Create API service layer

```
// src/context/AuthContext.jsx
import { createContext, useState, useEffect } from 'react';
import api from '../services/api';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      // Verify token and get user
      api.get('/auth/me')
        .then(res => setUser(res.data))
        .catch(() => localStorage.removeItem('token'))
        .finally(() => setLoading(false));
    } else {
      setLoading(false);
    }
  }, []);

  const login = async (username, password) => {
    const formData = new FormData();
    formData.append('username', username);
    formData.append('password', password);

    const response = await api.post('/auth/login', formData);
    localStorage.setItem('token', response.data.access_token);
    setUser(response.data.user);
  };

  const logout = () => {
    localStorage.removeItem('token');
    setUser(null);
  };

  return (
```

```
    <AuthContext.Provider value={{ user, loading, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

Afternoon Tasks (2-3 hours)

- Create Login component
- Create Register component
- Style with Tailwind CSS
- Test authentication flow

```
// src/components/Auth/Login.jsx
import { useState, useContext } from 'react';
import { useNavigate } from 'react-router-dom';
import { AuthContext } from '../../context/AuthContext';

const Login = () => {
  const [formData, setFormData] = useState({ username: '', password: '' });
  const [error, setError] = useState('');
  const { login } = useContext(AuthContext);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await login(formData.username, formData.password);
      navigate('/dashboard');
    } catch (err) {
      setError('Invalid credentials');
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-100">
      <div className="bg-white p-8 rounded-lg shadow-md w-96">
        <h2 className="text-2xl font-bold mb-6 text-center">Login</h2>

        {error && (
          <div className="bg-red-100 text-red-700 p-3 rounded mb-4">
            {error}
          </div>
        )}

        <form onSubmit={handleSubmit}>
          <div className="mb-4">
            <label className="block text-sm font-medium mb-2">Username</label>
            <input
              type="text"
              value={formData.username}
              onChange={(e) => setFormData({...formData, username: e.target.value})}
```

```
              className="w-full px-3 py-2 border rounded-lg focus:outline-none
focus:ring-2"
              required
            />
          </div>

          <div className="mb-6">
            <label className="block text-sm font-medium mb-2">Password</label>
            <input
              type="password"
              value={formData.password}
              onChange={(e) => setFormData({...formData, password: e.target.value})}
              className="w-full px-3 py-2 border rounded-lg focus:outline-none
focus:ring-2"
              required
            />
          </div>

          <button
            type="submit"
            className="w-full bg-blue-600 text-white py-2 rounded-lg hover:bg-blue-700"
          >
            Login
          </button>
        </form>
      </div>
    </div>
  );
};

export default Login;
```

✓Day 15 Checkpoint

- Login/Register pages working
- Can authenticate with backend
- Token stored in localStorage

Day 16: Dashboard & Navigation

All Day Tasks (5-6 hours)

- Create Navbar component
- Create Sidebar component
- Create Student Dashboard
- Create Faculty Dashboard
- Implement routing
- Add protected routes

✅Day 16 Checkpoint

- Navigation working
- Role-based dashboards
- Protected routes functional

Day 17: Project Upload Interface

All Day Tasks (5-6 hours)

- Create ProjectUpload component
- Implement file selection UI
- Add form validation
- Implement upload progress indicator
- Test file upload to backend

✅Day 17 Checkpoint

- Can upload projects from UI
- Files sent to backend correctly
- User feedback during upload

Day 18: Project List & Details

All Day Tasks (5-6 hours)

- Create ProjectList component
- Create ProjectDetails component
- Implement status badges
- Add pagination
- Test data fetching

✅Day 18 Checkpoint

- Can view list of projects

- Can view project details
- Status updates display correctly

Day 19: Evaluation Results Display

All Day Tasks (5-6 hours)

- Create EvaluationResults component
- Create ScoreCard component with charts
- Create FeedbackDisplay component
- Add charts using Recharts library
- Style results page

✅Day 19 Checkpoint

- Evaluation results display beautifully
- Charts showing score breakdown
- Feedback readable and formatted

Day 20: Faculty Review Interface

All Day Tasks (5-6 hours)

- Create pending evaluations list
- Create review interface for faculty
- Add ability to modify grades
- Add comments section
- Implement finalize button

✅Day 20 Checkpoint

- Faculty can review evaluations
- Can modify and finalize grades
- Changes save to backend

 WEEK 3 COMPLETE! Frontend is functional.

🗓️ Week 4: Integration, Testing & Deployment (Days 22-30)

Day 22: Backend-Frontend Integration

All Day Tasks (5-6 hours)

- Test complete user flow end-to-end
- Fix CORS issues
- Fix authentication bugs
- Test file upload/download
- Verify all API calls work

✅Day 22 Checkpoint

- Complete flow works: register → upload → evaluate → view results
- No console errors
- All features working together

Day 23: Async Processing with Celery

All Day Tasks (5-6 hours)

- Install and configure Celery
- Create Celery tasks for evaluation
- Set up Redis as message broker
- Implement status polling in frontend
- Test async evaluation

```
# backend/app/tasks/evaluation_tasks.py
from celery import Celery
from app.ai_engine.evaluator import ProjectEvaluator
from app.database.connection import SessionLocal
from app.models import Project, Evaluation

celery_app = Celery('tasks', broker='redis://localhost:6379/0')

@celery_app.task
def evaluate_project_async(project_id: int):
    db = SessionLocal()

    try:
        project = db.query(Project).filter(Project.id == project_id).first()
        if not project:
            return {"error": "Project not found"}

        # Update status
        project.status = "processing"
        db.commit()
```

```
        # Run evaluation
        evaluator = ProjectEvaluator()

        project_data = {
            'title': project.title,
            'code_file_path': project.code_file_path,
            'doc_file_path': project.doc_file_path,
            'existing_projects': []  # Fetch from DB
        }

        results = evaluator.evaluate_project(project_data)

        # Save results
        evaluation = Evaluation(
            project_id=project_id,
            code_quality_score=results['code_results']['final_score'],
            documentation_score=results['doc_results']['final_score'],
            originality_score=results['plagiarism_results']['originality_score'],
            final_score=results['grading_results']['final_score'],
            ai_feedback=results['ai_feedback']
        )

        db.add(evaluation)
        project.status = "evaluated"
        db.commit()

        return {"status": "success", "project_id": project_id}

    except Exception as e:
        project.status = "error"
        db.commit()
        return {"status": "error", "message": str(e)}
    finally:
        db.close()
```

✓Day 23 Checkpoint

- Celery worker running
- Evaluation happens asynchronously
- Frontend shows processing status

Day 24: Testing - Backend

All Day Tasks (5-6 hours)

- Write unit tests for AI components
- Write API endpoint tests
- Test authentication flows
- Test file upload

- Run all tests and fix failures

✓Day 24 Checkpoint
- All tests passing
- Code coverage > 70%
- No critical bugs found

Day 25: Testing - Frontend & E2E

All Day Tasks (5-6 hours)
- Write React component tests
- Test user interactions
- Manual testing of all features
- Create test user accounts
- Document any bugs found

✓Day 25 Checkpoint
- Frontend tests passing
- Manual testing complete
- Bug list created and prioritized

Day 26: Bug Fixes & Polish

All Day Tasks (5-6 hours)
- Fix all critical bugs
- Improve UI/UX
- Add loading indicators
- Add error handling
- Improve validation messages

✓Day 26 Checkpoint
- All critical bugs fixed
- UI polished and professional
- Good error messages

Day 27: Docker Setup

All Day Tasks (5-6 hours)

- Create Dockerfile for backend
- Create Dockerfile for frontend
- Create docker-compose.yml
- Test Docker setup locally
- Document Docker commands

```
# docker-compose.yml
version: '3.8'

services:
  backend:
    build: ./backend
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/evaldb
      - REDIS_URL=redis://redis:6379/0
    depends_on:
      - db
      - redis

  frontend:
    build: ./frontend
    ports:
      - "3000:80"

  db:
    image: postgres:15-alpine
    environment:
      - POSTGRES_DB=evaldb
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass

  redis:
    image: redis:7-alpine

  celery_worker:
    build: ./backend
    command: celery -A app.tasks.celery_app worker --loglevel=info
    depends_on:
      - redis
      - db
```

✓Day 27 Checkpoint

- Docker containers running
- Application works in Docker

- All services communicating

Day 28: Deployment Preparation

All Day Tasks (5-6 hours)

- Sign up for cloud provider (AWS/DigitalOcean)
- Configure production environment variables
- Set up PostgreSQL database
- Configure Nginx
- Set up SSL certificates

✅Day 28 Checkpoint

- Cloud server provisioned
- Database created
- SSL certificates obtained

Day 29: Production Deployment

All Day Tasks (5-6 hours)

- Deploy backend to server
- Deploy frontend to server
- Configure domain name
- Test production deployment
- Set up monitoring

```
# SSH into server
ssh user@your-server-ip

# Clone repository
git clone https://github.com/yourusername/ai-project-evaluation.git
cd ai-project-evaluation

# Create .env file with production values
nano backend/.env

# Start with Docker Compose
docker-compose -f docker-compose.prod.yml up -d

# Check logs
docker-compose logs -f
```

✅**Day 29 Checkpoint**

- Application deployed and accessible
- HTTPS working
- All features functional in production

Day 30: Final Testing & Documentation

All Day Tasks (5-6 hours)

- Complete end-to-end testing in production
- Write user documentation
- Create setup instructions
- Record demo video
- Prepare presentation slides

✅**Day 30 Checkpoint - PROJECT COMPLETE!**

- System fully deployed and working
- Documentation complete
- Demo ready
- Presentation prepared

 CONGRATULATIONS! You've completed a full-stack AI-powered application in 30 days!

❓ Additional Resources & Tips

Recommended Learning Resources

- FastAPI Documentation: https://fastapi.tiangolo.com/
- React Documentation: https://react.dev/
- Sentence-Transformers: https://www.sbert.net/
- Docker Documentation: https://docs.docker.com/
- PostgreSQL Tutorial: https://www.postgresqltutorial.com/

Time Management Tips

- Start early in the day when you're most focused
- Take breaks every 2 hours to avoid burnout
- If stuck on a bug for >1 hour, ask for help or move on
- Commit your code to Git at the end of each day
- Don't aim for perfection - aim for completion
- Use weekends to catch up if you fall behind
- Test frequently - don't wait until the end
- Read error messages carefully before Googling

Common Issues & Solutions

- CORS Errors:

Add proper CORS middleware in FastAPI with correct origins

- Module Not Found:

Check virtual environment is activated and dependencies installed

- Database Connection Failed:

Verify PostgreSQL is running and credentials are correct

- File Upload Fails:

Check file size limits and upload directory permissions

- OpenAI API Errors:

Verify API key is correct and you have credits

What to Do If You Fall Behind

- Skip optional features (e.g., admin dashboard)
- Use simpler AI models if API costs are high
- Skip deployment and demo locally instead

- Focus on core features: upload, evaluate, display results
- Ask friends/classmates for help on specific problems
- Use ChatGPT or Claude to debug errors

Good luck with your capstone project! You've got this! □