AI Smart Academic Project Evaluation System

Complete Implementation Guide

Full-Stack Development | AI/ML Integration | System Architecture

Table of Contents

1. Project Overview & Architecture

1.1 System Architecture

The AI Smart Academic Project Evaluation System follows a modern three-tier architecture with clear separation of concerns:

- Frontend Layer:

React.js framework with Redux/Context API for state management, Axios for HTTP requests, React Router for navigation, and Tailwind CSS for responsive styling.

- Backend Layer:

Python FastAPI framework providing RESTful APIs, JWT-based authentication, file processing capabilities, and integration with AI/ML services.

- AI Engine Layer:

Sentence-BERT and MiniLM for semantic similarity, LLM APIs for feedback generation, static code analyzers (Pylint, Radon, ESLint) for quality assessment.

- Database Layer:

PostgreSQL for relational data storage with SQLAlchemy ORM, Redis for caching and session management.

1.2 Data Flow Diagram

The system processes data through the following workflow:

12. 1. Student uploads project files (code + documentation) via React frontend
13. 2. FastAPI backend receives files, validates formats, and stores in database
14. 3. AI Engine triggered with three parallel processing pipelines: Code Quality Analysis, Documentation Evaluation, and Plagiarism Detection
15. 4. Results aggregated and LLM generates comprehensive feedback
16. 5. Scores stored in database, notifications sent to faculty via email
17. 6. Faculty reviews results via dashboard, can modify and finalize grades
18. 7. Final results published to students with detailed feedback reports

1.3 Key Features

- Automated code quality analysis using industry-standard tools
- NLP-based documentation evaluation for completeness and clarity
- AI-powered plagiarism detection using semantic similarity
- LLM-generated personalized feedback for students
- Role-based access control (Student, Faculty, Admin)
- Real-time notifications and email alerts

- Comprehensive PDF report generation
- Dashboard analytics for administrators

2. Complete Project Structure

Below is the recommended full-stack project structure organized for scalability and maintainability:

```
ai-project-evaluation-system/
│
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py                 # FastAPI application entry
│   │   ├── config.py               # Configuration settings
│   │   │
│   │   ├── api/                    # API routes
│   │   │   ├── __init__.py
│   │   │   ├── auth.py             # Authentication endpoints
│   │   │   ├── projects.py         # Project upload/management
│   │   │   ├── evaluations.py      # Evaluation endpoints
│   │   │   ├── users.py            # User management
│   │   │   └── reports.py          # Report generation
│   │   │
│   │   ├── models/                 # Database models
│   │   │   ├── __init__.py
│   │   │   ├── user.py
│   │   │   ├── project.py
│   │   │   ├── evaluation.py
│   │   │   └── feedback.py
│   │   │
│   │   ├── schemas/                # Pydantic schemas
│   │   │   ├── __init__.py
│   │   │   ├── user.py
│   │   │   ├── project.py
│   │   │   └── evaluation.py
│   │   │
│   │   ├── services/               # Business logic
│   │   │   ├── __init__.py
│   │   │   ├── auth_service.py
│   │   │   ├── file_service.py
│   │   │   └── email_service.py
│   │   │
│   │   ├── ai_engine/              # AI/ML components
│   │   │   ├── __init__.py
│   │   │   ├── code_analyzer.py    # Code quality analysis
│   │   │   ├── doc_evaluator.py    # Documentation assessment
│   │   │   ├── plagiarism_detector.py # Plagiarism detection
│   │   │   ├── feedback_generator.py  # LLM-based feedback
│   │   │   └── grading_engine.py   # Final grade calculation
│   │   │
│   │   ├── utils/                  # Utility functions
│   │   │   ├── __init__.py
│   │   │   ├── dependencies.py
│   │   │   ├── security.py
│   │   │   └── validators.py
│   │   │
│   │   └── database/
│   │       ├── __init__.py
│   │       ├── connection.py
│   │       └── migrations/
│   │
│   ├── tests/
│   ├── requirements.txt
│   ├── .env
│   └── Dockerfile
│
├── frontend/
│   ├── public/
│   ├── src/
│   │   ├── components/
│   │   │   ├── Auth/
│   │   │   │   ├── Login.jsx
│   │   │   │   └── Register.jsx
│   │   │   ├── Dashboard/
│   │   │   │   ├── StudentDashboard.jsx
│   │   │   │   ├── FacultyDashboard.jsx
│   │   │   │   └── AdminDashboard.jsx
│   │   │   ├── Project/
│   │   │   │   ├── ProjectUpload.jsx
│   │   │   │   ├── ProjectList.jsx
│   │   │   │   └── ProjectDetails.jsx
│   │   │   ├── Evaluation/
│   │   │   │   ├── EvaluationResults.jsx
│   │   │   │   ├── ScoreCard.jsx
│   │   │   │   └── FeedbackDisplay.jsx
│   │   │   ├── Common/
│   │   │   │   ├── Navbar.jsx
│   │   │   │   ├── Sidebar.jsx
│   │   │   │   └── Loader.jsx
│   │   │   └── Reports/
│   │   │       └── ReportGenerator.jsx
│   │   │
│   │   ├── pages/
│   │   │   ├── HomePage.jsx
│   │   │   ├── DashboardPage.jsx
│   │   │   └── NotFoundPage.jsx
│   │   │
│   │   ├── services/
│   │   │   ├── api.js
│   │   │   ├── authService.js
│   │   │   └── projectService.js
│   │   │
│   │   ├── context/
│   │   │   ├── AuthContext.jsx
│   │   │   └── ProjectContext.jsx
│   │   │
│   │   ├── hooks/
│   │   │   ├── useAuth.js
│   │   │   └── useProject.js
│   │   │
│   │   ├── utils/
│   │   │   ├── constants.js
│   │   │   └── helpers.js
│   │   │
│   │   ├── App.jsx
│   │   ├── index.js
│   │   └── index.css
│   │
│   ├── package.json
│   └── tailwind.config.js
│
├── docker-compose.yml
├── .gitignore
└── README.md
```

3. Backend Development (Python/FastAPI)

3.1 Setting Up FastAPI Backend

Step 1: Install Dependencies

```
# 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
sentence-transformers==2.2.2
radon==6.0.1
pylint==3.0.2
openai==1.3.7
redis==5.0.1
celery==5.3.4
python-dotenv==1.0.0
pytest==7.4.3
httpx==0.25.2
```

Step 2: Main Application (app/main.py)

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.api import auth, projects, evaluations, users, reports
from app.database.connection import engine
from app.models import Base

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

app = FastAPI(
    title="AI Project Evaluation System",
    description="Automated 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=["*"]
)
```

```python
# Include routers
app.include_router(auth.router, prefix="/api/auth", tags=["auth"])
app.include_router(projects.router, prefix="/api/projects", tags=["projects"])
app.include_router(evaluations.router, prefix="/api/evaluations", tags=["evaluations"])
app.include_router(users.router, prefix="/api/users", tags=["users"])
app.include_router(reports.router, prefix="/api/reports", tags=["reports"])

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

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

3.2 Database Models

User Model (app/models/user.py)

```python
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)

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

Project Model (app/models/project.py)

```python
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime, Text, Enum
from sqlalchemy.orm import relationship
from app.database.connection import Base
from datetime import datetime
import enum
```

```
class ProjectStatus(str, enum.Enum):
    UPLOADED = "uploaded"
    PROCESSING = "processing"
    EVALUATED = "evaluated"
    PUBLISHED = "published"

class Project(Base):
    __tablename__ = "projects"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, nullable=False)
    description = Column(Text)
    student_id = Column(Integer, ForeignKey("users.id"))
    code_file_path = Column(String)
    doc_file_path = Column(String)
    programming_language = Column(String)
    status = Column(Enum(ProjectStatus), default=ProjectStatus.UPLOADED)
    uploaded_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    student = relationship("User", back_populates="projects")
    evaluation = relationship("Evaluation", back_populates="project", uselist=False)
```

Evaluation Model (app/models/evaluation.py)

```
from sqlalchemy import Column, Integer, Float, ForeignKey, DateTime, Text, Boolean
from sqlalchemy.orm import relationship
from app.database.connection import Base
from datetime import datetime

class Evaluation(Base):
    __tablename__ = "evaluations"

    id = Column(Integer, primary_key=True, index=True)
    project_id = Column(Integer, ForeignKey("projects.id"))
    code_quality_score = Column(Float)
    documentation_score = Column(Float)
    originality_score = Column(Float)
    final_score = Column(Float)
    ai_feedback = Column(Text)
    faculty_comments = Column(Text)
    is_finalized = Column(Boolean, default=False)
    evaluated_at = Column(DateTime, default=datetime.utcnow)
    finalized_at = Column(DateTime)
    finalized_by = Column(Integer, ForeignKey("users.id"))

    # Relationships
    project = relationship("Project", back_populates="evaluation")
```

4. AI/ML Engine Implementation

4.1 Code Quality Analyzer

The Code Quality Analyzer module uses static analysis tools to evaluate code quality, complexity, and maintainability.

```python
# 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
import os

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

    def analyze(self, file_path: str) -> dict:
        """Analyze code quality and return scores"""
        with open(file_path, 'r', encoding='utf-8') as f:
            code = f.read()

        complexity_score = self._analyze_complexity(code)
        maintainability_score = self._analyze_maintainability(code)
        quality_score = self._analyze_quality(file_path)

        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': {
                'cyclomatic_complexity': self._get_complexity_details(code),
                'lines_of_code': len(code.split('\n')),
                'functions_count': self._count_functions(code)
            }
        }

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

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

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

    def _analyze_quality(self, file_path: str) -> float:
        """Calculate quality score using Pylint"""
        try:
            pylint_stdout, pylint_stderr = lint.py_run(
                f'{file_path} --output-format=json',
                return_std=True
            )
            # Parse Pylint output and calculate score
            # This is simplified - you'd parse JSON output
            return 75.0
        except:
            return 50.0

    def _get_complexity_details(self, code: str) -> dict:
        """Get detailed complexity metrics"""
        try:
            results = radon_cc.cc_visit(code)
            return {
                'average': sum(r.complexity for r in results) / len(results) if results
else 0,
                'max': max((r.complexity for r in results), default=0)
            }
        except:
            return {'average': 0, 'max': 0}

    def _count_functions(self, code: str) -> int:
        """Count number of functions in code"""
        try:
            tree = ast.parse(code)
            return sum(1 for node in ast.walk(tree) if isinstance(node,
ast.FunctionDef))
        except:
            return 0
```

4.2 Documentation Evaluator

The Documentation Evaluator uses NLP techniques to assess the quality and completeness of project documentation.

```python
# 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', 'overview', 'installation',
            'usage', 'features', 'requirements'
        ]

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

        completeness_score = self._check_completeness(content)
        clarity_score = self._assess_clarity(content)
        structure_score = self._evaluate_structure(content)

        final_score = (
            completeness_score * 0.4 +
            clarity_score * 0.3 +
            structure_score * 0.3
        )

        return {
            'final_score': round(final_score, 2),
            'completeness': round(completeness_score, 2),
            'clarity': round(clarity_score, 2),
            'structure': round(structure_score, 2),
            'word_count': len(content.split()),
            'missing_sections': self._find_missing_sections(content)
        }

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

    def _assess_clarity(self, content: str) -> float:
        """Assess clarity based on readability metrics"""
        sentences = content.split('.')
        avg_sentence_length = sum(len(s.split()) for s in sentences) / len(sentences)

        # Optimal sentence length is 15-20 words
        if 15 <= avg_sentence_length <= 20:
            return 100.0
        elif avg_sentence_length < 10:
            return 60.0
        elif avg_sentence_length > 30:
            return 50.0
        else:
            return 80.0

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

        structure_elements = sum([has_headings, has_code_blocks, has_lists])
        return (structure_elements / 3) * 100

    def _find_missing_sections(self, content: str) -> list:
        """Find which required sections are missing"""
        content_lower = content.lower()
        return [
            section for section in self.required_sections
            if section not in content_lower
        ]
```

4.3 Plagiarism Detector

The Plagiarism Detector uses Sentence-BERT embeddings to compute semantic similarity between the submitted code and existing projects in the database.

```
# 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 threshold

    def detect(self, current_code: str, existing_projects: list) -> dict:
        """
        Detect plagiarism by comparing with existing projects

        Args:
            current_code: Code to check
            existing_projects: List of dicts with 'id' and 'code' keys

        Returns:
            Dictionary with originality score and similar projects
        """
        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 (highest first)
        similarities.sort(key=lambda x: x['similarity'], reverse=True)

        max_similarity = similarities[0]['similarity'] if similarities else 0
        originality_score = (1 - max_similarity) * 100
        # Flag if similarity exceeds threshold
        is_flagged = max_similarity > self.threshold

        return {
            'originality_score': round(originality_score, 2),
            'flagged': is_flagged,
            'max_similarity': round(max_similarity * 100, 2),
            'similar_projects': similarities[:5],  # Top 5 similar projects
            'risk_level': self._get_risk_level(max_similarity)
        }

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

4.4 AI Feedback Generator

The Feedback Generator uses Large Language Models to generate comprehensive, personalized feedback based on all evaluation metrics.

```
# 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'))

    def generate(self, evaluation_data: dict) -> str:
        """
        Generate comprehensive feedback using LLM

        Args:
            evaluation_data: Dictionary containing all evaluation scores

        Returns:
            Generated feedback text
        """
        prompt = self._create_prompt(evaluation_data)

        try:
            response = self.client.chat.completions.create(
                model="gpt-4",
                messages=[
                    {
                        "role": "system",
                        "content": "You are an experienced academic evaluator 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:
        """Create detailed prompt for LLM"""
        return f"""
        Generate comprehensive, constructive feedback for a student project with the following evaluation:

        Code Quality Score: {data['code_score']}/100
        - Complexity: {data['complexity']}/100
        - Maintainability: {data['maintainability']}/100

        Documentation Score: {data['doc_score']}/100
        - Completeness: {data['completeness']}/100
        - Clarity: {data['clarity']}/100

        Originality Score: {data['originality_score']}/100
        - Plagiarism Risk: {data.get('risk_level', 'UNKNOWN')}

        Project Title: {data['project_title']}

        Please provide:
        1. Overall assessment (2-3 sentences)
        2. Strengths (2-3 points)
        3. Areas for improvement (2-3 points)
        4. Specific recommendations (2-3 actionable items)

        Keep feedback constructive, specific, and encouraging.
        """

    def _generate_fallback_feedback(self, data: dict) -> str:
        """Generate basic feedback if LLM fails"""
        return f"""
        Project Evaluation Summary:

        Your project has achieved the following scores:
        - Code Quality: {data['code_score']}/100
        - Documentation: {data['doc_score']}/100
        - Originality: {data['originality_score']}/100
        Overall, your project demonstrates
{self._get_performance_level(data['final_score'])} performance.

        Please review the detailed metrics and consult with your instructor for
specific improvement areas.
        """

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

4.5 Grading Engine

The Grading Engine aggregates all scores and applies weighted calculations to determine the final grade.

```
# app/ai_engine/grading_engine.py

class GradingEngine:
    def __init__(self):
        # Configurable weights for different evaluation components
        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),
            'D': (50, 59),
            'F': (0, 49)
        }

    def calculate_final_score(self, scores: dict) -> dict:
        """
        Calculate final score and grade

        Args:
            scores: Dictionary with code_score, doc_score, originality_score

        Returns:
            Dictionary with final_score, letter_grade, and breakdown
        """
        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:
        """Convert numerical score to letter grade"""
        for grade, (min_score, max_score) in self.grade_scale.items():
            if min_score <= score <= max_score:
                return grade
        return 'F'
```

5. Frontend Development (React.js)

5.1 Setting Up React Application

```
# Create React app
npx create-react-app frontend
cd frontend

# Install dependencies
npm install react-router-dom axios
npm install @reduxjs/toolkit react-redux
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p

# Additional UI libraries (optional)
npm install react-icons
npm install recharts  # For charts/graphs
```

5.2 Authentication Context

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

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) {
      authService.getCurrentUser()
        .then(setUser)
        .catch(() => localStorage.removeItem('token'))
        .finally(() => setLoading(false));
    } else {
      setLoading(false);
    }
  }, []);

  const login = async (credentials) => {
    const response = await authService.login(credentials);
    localStorage.setItem('token', response.access_token);
    setUser(response.user);
    return response;
  };

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

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

5.3 API Service Layer

```
// src/services/api.js
import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://localhost:8000/api';

const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Request interceptor to add auth token
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Response interceptor for error handling
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export default api;
```

5.4 Project Upload Component

```
// src/components/Project/ProjectUpload.jsx
import { useState } from 'react';
import { projectService } from '../../services/projectService';
import { useNavigate } from 'react-router-dom';

const ProjectUpload = () => {
  const navigate = useNavigate();
  const [formData, setFormData] = useState({
    title: '',
    description: '',
    programming_language: 'python',
    codeFile: null,
    docFile: null
  });
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

  const handleFileChange = (e) => {
    const { name, files } = e.target;
    setFormData(prev => ({ ...prev, [name]: files[0] }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError(null);

    const data = new FormData();
    Object.keys(formData).forEach(key => {
      if (formData[key]) {
        data.append(key, formData[key]);
      }
    });

    try {
      const response = await projectService.uploadProject(data);
      navigate(`/projects/${response.id}`);
    } catch (err) {
      setError(err.response?.data?.detail || 'Failed to upload project');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="max-w-2xl mx-auto p-6">
      <h2 className="text-2xl font-bold mb-6">Upload New Project</h2>

      {error && (
        <div className="bg-red-100 border border-red-400 text-red-700 px-4 py-3 rounded
mb-4">
          {error}
        </div>
      )}

      <form onSubmit={handleSubmit} className="space-y-4">
        <div>
          <label className="block text-sm font-medium mb-1">Project Title</label>
          <input
            type="text"
            name="title"
            value={formData.title}
            onChange={handleChange}
            required
            className="w-full px-3 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-500"
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-1">Description</label>
          <textarea
            name="description"
            value={formData.description}
            onChange={handleChange}
            rows="4"
            className="w-full px-3 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-500"
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-1">Programming
Language</label>
          <select
            name="programming_language"
            value={formData.programming_language}
            onChange={handleChange}
            className="w-full px-3 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-500"
          >
            <option value="python">Python</option>
            <option value="javascript">JavaScript</option>
            <option value="java">Java</option>
            <option value="cpp">C++</option>
          </select>
        </div>

        <div>
          <label className="block text-sm font-medium mb-1">Code Files (.zip)</label>
          <input
            type="file"
            name="codeFile"
            onChange={handleFileChange}
            accept=".zip,.py,.js,.java"
            required
            className="w-full px-3 py-2 border rounded-lg"
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-1">Documentation
(.pdf/.md)</label>
          <input
            type="file"
            name="docFile"
            onChange={handleFileChange}
            accept=".pdf,.md,.txt"
            required
            className="w-full px-3 py-2 border rounded-lg"
          />
        </div>

        <button
          type="submit"
          disabled={loading}
          className="w-full bg-blue-600 text-white py-2 px-4 rounded-lg hover:bg-blue-
700 disabled:bg-gray-400"
        >
          {loading ? 'Uploading...' : 'Upload Project'}
        </button>
      </form>
    </div>
  );
};

export default ProjectUpload;
```

6. Complete Data Flow & API Integration

6.1 Project Submission Flow

The complete flow from project submission to evaluation:

19. 1. Student fills out project upload form with title, description, and files
20. 2. Frontend validates form data and creates FormData object
21. 3. POST request sent to /api/projects/upload with multipart/form-data
22. 4. FastAPI backend receives request, validates JWT token
23. 5. File validator checks file types, sizes, and content
24. 6. Files saved to disk storage with unique identifiers
25. 7. Project metadata stored in PostgreSQL database
26. 8. Celery task queued for asynchronous AI evaluation
27. 9. Success response returned to frontend with project_id
28. 10. Frontend redirects to project details page
29. 11. Real-time status updates via WebSocket or polling

6.2 AI Evaluation Pipeline

Parallel processing of evaluation components:

```
# app/tasks/evaluation_tasks.py
from celery import Celery
from app.ai_engine import (
    CodeAnalyzer,
    DocumentationEvaluator,
    PlagiarismDetector,
    FeedbackGenerator,
    GradingEngine
)
from app.database import SessionLocal
from app.models import Evaluation, Project

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

@celery_app.task
def evaluate_project(project_id: int):
    """Main evaluation task that orchestrates all AI components"""
    db = SessionLocal()

    try:
        project = db.query(Project).filter(Project.id == project_id).first()
        if not project:
            return

        # Update status
        project.status = 'processing'
        db.commit()
        # Initialize AI components
        code_analyzer = CodeAnalyzer()
        doc_evaluator = DocumentationEvaluator()
        plagiarism_detector = PlagiarismDetector()
        feedback_generator = FeedbackGenerator()
        grading_engine = GradingEngine()

        # Run evaluations
        code_results = code_analyzer.analyze(project.code_file_path)
        doc_results = doc_evaluator.evaluate(project.doc_file_path)

        # Get existing projects for plagiarism check
        existing_projects = get_existing_projects(db, project.id)
        plagiarism_results = plagiarism_detector.detect(
            read_code_file(project.code_file_path),
            existing_projects
        )

        # Calculate final grade
        final_results = grading_engine.calculate_final_score({
            'code_score': code_results['final_score'],
            'doc_score': doc_results['final_score'],
            'originality_score': plagiarism_results['originality_score']
        })

        # Generate AI feedback
        evaluation_data = {
            'project_title': project.title,
            'code_score': code_results['final_score'],
            'complexity': code_results['complexity'],
            'maintainability': code_results['maintainability'],
            'doc_score': doc_results['final_score'],
            'completeness': doc_results['completeness'],
            'clarity': doc_results['clarity'],
            'originality_score': plagiarism_results['originality_score'],
            'risk_level': plagiarism_results['risk_level'],
            'final_score': final_results['final_score']
        }

        ai_feedback = feedback_generator.generate(evaluation_data)

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

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

        # Send notification to faculty
        send_evaluation_notification(project.student_id, project_id)

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

    except Exception as e:
        project.status = 'error'
        db.commit()
        raise e
    finally:
        db.close()
```

7. Complete API Endpoints Reference

Below is the complete list of API endpoints with request/response examples:

Method	Endpoint	Description	Auth Required
POST	/api/auth/register	Register new user	No
POST	/api/auth/login	Login and get JWT token	No
GET	/api/auth/me	Get current user info	Yes
POST	/api/projects/upload	Upload new project	Yes (Student)
GET	/api/projects/my	Get user projects	Yes
GET	/api/projects/{id}	Get project details	Yes
GET	/api/evaluations/{id}	Get evaluation results	Yes
GET	/api/evaluations/pending	Get pending evaluations	Yes (Faculty)
PUT	/api/evaluations/{id}/finalize	Finalize evaluation	Yes (Faculty)
GET	/api/reports/generate/{id}	Generate PDF report	Yes
GET	/api/users	List all users	Yes (Admin)
DELETE	/api/projects/{id}	Delete project	Yes (Student/Admin)

7.1 Example API Requests

```
# Authentication - Register
POST /api/auth/register
{
  "email": "student@university.edu",
  "username": "john_doe",
  "password": "SecurePass123!",
  "full_name": "John Doe",
  "role": "student",
  "department": "Computer Science"
}

Response:
{
  "id": 1,
  "email": "student@university.edu",
  "username": "john_doe",
  "role": "student",
  "created_at": "2024-01-15T10:30:00Z"
}

# Authentication - Login
POST /api/auth/login
{
  "username": "john_doe",
  "password": "SecurePass123!"
}

Response:
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer",
  "user": {
    "id": 1,
    "email": "student@university.edu",
    "role": "student"
  }
}

# Project Upload (multipart/form-data)
POST /api/projects/upload
Content-Type: multipart/form-data

title: "Machine Learning Classifier"
description: "A project implementing various ML algorithms"
programming_language: "python"
codeFile: [binary data]
docFile: [binary data]

Response:
{
  "id": 42,
  "title": "Machine Learning Classifier",
  "status": "uploaded",
  "uploaded_at": "2024-01-15T11:00:00Z",
  "message": "Project uploaded successfully. Evaluation in progress."
}

# Get Evaluation Results
GET /api/evaluations/42
Authorization: Bearer <token>

Response:
{
  "id": 42,
  "project_id": 42,
  "code_quality_score": 85.5,
  "documentation_score": 78.0,
  "originality_score": 92.0,
  "final_score": 85.2,
  "letter_grade": "A-",
  "ai_feedback": "Your project demonstrates strong technical implementation...",
  "is_finalized": false,
  "evaluated_at": "2024-01-15T11:15:00Z"
}
```

8. Deployment & DevOps

8.1 Docker Configuration

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

services:
  backend:
    build: ./backend
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/evaldb
      - REDIS_URL=redis://redis:6379/0
      - OPENAI_API_KEY=${OPENAI_API_KEY}
    depends_on:
      - db
      - redis
    volumes:
      - ./backend:/app
      - uploads:/app/uploads

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

  db:
    image: postgres:15-alpine
    environment:
      - POSTGRES_DB=evaldb
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  celery_worker:
    build: ./backend
    command: celery -A app.tasks.celery_app worker --loglevel=info
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/evaldb
      - REDIS_URL=redis://redis:6379/0
    depends_on:
      - db
      - redis

volumes:
  postgres_data:
  uploads:
```

8.2 Production Deployment Steps

30. 1. Provision cloud server (AWS EC2, DigitalOcean, or GCP Compute Engine)
31. 2. Install Docker and Docker Compose on the server
32. 3. Set up domain name and point DNS to server IP
33. 4. Configure SSL/TLS certificates using Let's Encrypt
34. 5. Set up Nginx as reverse proxy with SSL termination
35. 6. Create .env files with production environment variables
36. 7. Set up PostgreSQL backups (automated daily backups)
37. 8. Configure monitoring (Prometheus + Grafana)
38. 9. Set up logging (ELK stack or CloudWatch)
39. 10. Deploy application using docker-compose up -d
40. 11. Set up CI/CD pipeline (GitHub Actions or GitLab CI)
41. 12. Configure firewall rules and security groups

8.3 Nginx Configuration

```
# /etc/nginx/sites-available/eval-system
server {
    listen 80;
    server_name yourdomain.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name yourdomain.com;

    ssl_certificate /etc/letsencrypt/live/yourdomain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/yourdomain.com/privkey.pem;

    # Frontend
    location / {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    # Backend API
    location /api {
        proxy_pass http://localhost:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        client_max_body_size 50M;
    }
}
```

9. Testing Strategy

9.1 Backend Testing (Pytest)

```python
# tests/test_code_analyzer.py
import pytest
from app.ai_engine.code_analyzer import CodeAnalyzer

@pytest.fixture
def code_analyzer():
    return CodeAnalyzer()

def test_code_analyzer_basic(code_analyzer, tmp_path):
    # Create temporary Python file
    test_file = tmp_path / "test.py"
    test_file.write_text("""
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b
""")

    result = code_analyzer.analyze(str(test_file))

    assert 'final_score' in result
    assert 0 <= result['final_score'] <= 100
    assert 'complexity' in result
    assert 'maintainability' in result

def test_plagiarism_detection():
    from app.ai_engine.plagiarism_detector import PlagiarismDetector

    detector = PlagiarismDetector()

    code1 = "def hello(): print('Hello World')"
    code2 = "def hello(): print('Hello World')"

    existing_projects = [{'id': 1, 'code': code2}]
    result = detector.detect(code1, existing_projects)

    assert result['originality_score'] < 50  # Should detect high similarity
    assert result['flagged'] == True
```

9.2 Frontend Testing (Jest + React Testing Library)

```javascript
// src/components/Project/__tests__/ProjectUpload.test.js
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import { BrowserRouter } from 'react-router-dom';
import ProjectUpload from '../ProjectUpload';
import * as projectService from '../../../services/projectService';
```

```
jest.mock('../../../services/projectService');

describe('ProjectUpload Component', () => {
  test('renders upload form', () => {
    render(
      <BrowserRouter>
        <ProjectUpload />
      </BrowserRouter>
    );

    expect(screen.getByText(/upload new project/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/project title/i)).toBeInTheDocument();
  });

  test('handles file upload', async () => {
    projectService.uploadProject.mockResolvedValue({ id: 1 });

    render(
      <BrowserRouter>
        <ProjectUpload />
      </BrowserRouter>
    );

    const titleInput = screen.getByLabelText(/project title/i);
    const codeFileInput = screen.getByLabelText(/code files/i);

    fireEvent.change(titleInput, { target: { value: 'Test Project' } });

    const file = new File(['code'], 'test.py', { type: 'text/plain' });
    fireEvent.change(codeFileInput, { target: { files: [file] } });

    const submitButton = screen.getByText(/upload project/i);
    fireEvent.click(submitButton);

    await waitFor(() => {
      expect(projectService.uploadProject).toHaveBeenCalled();
    });
  });
});
```

10. Implementation Timeline

Recommended timeline for implementing the complete system:

Phase	Duration	Tasks
Phase 1: Setup & Planning	Week 1	Project structure setup, database design, technology stack finalization
Phase 2: Backend Core	Weeks 2-3	FastAPI setup, database models, authentication, file handling
Phase 3: AI Engine	Weeks 4-5	Code analyzer, doc evaluator, plagiarism detector, feedback generator
Phase 4: Frontend Core	Weeks 6-7	React setup, authentication UI, project upload, dashboard
Phase 5: Integration	Week 8	API integration, end-to-end testing, bug fixes
Phase 6: Advanced Features	Week 9	Reports, notifications, analytics dashboard
Phase 7: Testing & QA	Week 10	Comprehensive testing, performance optimization
Phase 8: Deployment	Week 11	Production deployment, monitoring setup, documentation
Phase 9: Polish & Launch	Week 12	Final testing, user training, soft launch

10.1 Key Milestones

- Week 3: Backend API functional with basic CRUD operations
- Week 5: AI evaluation engine completing basic analysis
- Week 7: Frontend with complete user workflows
- Week 8: Full integration with working end-to-end flow
- Week 10: Production-ready system with all features
- Week 12: Deployed system with user documentation

11. Best Practices & Tips

11.1 Code Quality

- Follow PEP 8 style guide for Python code
- Use ESLint and Prettier for JavaScript/React code
- Write comprehensive docstrings and comments
- Implement proper error handling and logging
- Use type hints in Python (mypy)
- Keep functions small and focused (Single Responsibility Principle)
- Write unit tests as you develop (TDD approach recommended)

11.2 Security Considerations

- Never commit API keys or sensitive credentials to Git
- Use environment variables for all configuration
- Implement rate limiting on all API endpoints
- Validate and sanitize all user inputs
- Use parameterized queries to prevent SQL injection
- Implement proper CORS policies
- Keep dependencies updated and scan for vulnerabilities
- Use HTTPS in production
- Implement proper session management and token expiration

11.3 Performance Optimization

- Use database indexing on frequently queried fields
- Implement caching with Redis for expensive operations
- Use Celery for asynchronous processing of AI evaluations
- Optimize AI model loading (load once, reuse)
- Implement pagination for large data sets
- Use lazy loading for React components
- Compress file uploads before transmission
- Monitor and optimize database queries
- Use CDN for static assets in production

Conclusion

This comprehensive implementation guide provides a complete roadmap for building the AI Smart Academic Project Evaluation System. By following the structured approach outlined in this document, you can create a production-ready full-stack application that leverages modern AI/ML technologies to automate and improve the academic project evaluation process.

Remember to start with a solid foundation, implement features iteratively, test thoroughly, and maintain clean, well-documented code throughout the development process. Good luck with your capstone project!