# K12 Answer Sheet Evaluator - Project Documentation

## 1. Core Planning Documents

### 1.1 Project Charter

- **Purpose:** To automate and streamline the evaluation of K12 handwritten answer sheets using OCR and AI, reducing educator workload and providing rapid, detailed feedback to students.
- **Objectives:**
  - Achieve an AI grading accuracy comparable to human teachers.
  - Reduce evaluation time per paper from minutes to seconds.
  - Provide a robust RAG (Retrieval-Augmented Generation) system to anchor AI evaluations to specific textbook curricula to prevent hallucination.
  - **Hackathon Focus:** Move beyond simple "marks" by explicitly telling students **what is wrong**, **why it is wrong**, **what concepts are missing**, and exactly **how to improve** using the `google-genai` evaluator prompt.
- **Scope:** The system covers teacher assignment creation, student answer submission (image/PDF uploads), AI-driven evaluation using Google Cloud Vision and Gemini AI, report card generation, and real-time notifications.
- **Stakeholders:**
  - **Teachers/Educators:** End-users creating papers and reviewing AI grades.
  - **Students:** End-users submitting answers and reviewing performance.
  - **School Administrators:** Overseeing system usage and metrics.
- **High-Level Risks:**
  - *Risk:* Inaccurate OCR extraction due to poor student handwriting.
  - *Risk:* AI hallucinations or incorrect grading penalizing students.
  - *Mitigation:* Mandatory teacher review overrides; robust RAG context injection.

### 1.2 Requirements Specification

- **Functional:**
  - Teachers can upload textbooks to build a vector knowledge base.
  - Students can submit images of physical answer sheets.
  - System automatically detects questions, handwriting, and diagrams.
  - System matches answers against the answer key and textbook data.
  - System generates PDF report cards and Leaderboards.
- **Non-Functional:**
  - High availability, real-time sync (polling/websockets), low latency for the UI.
  - Secure JWT authentication and role-based access control (RBAC).

---

## 2. Technical Specifications

### 2.1 System Architecture

- **Client-Server Model:** Frontend built in React communicating with a Python FastAPI backend via REST APIs.
- **Asynchronous Processing:** Heavy AI/OCR tasks are offloaded to FastAPI `BackgroundTasks` to avoid blocking the HTTP request thread.

- **RAG Architecture:** Textbooks are parsed ( `PyMuPDF` ), chunked, embedded ( `sentence-transformers` ), and stored in Qdrant. Context is retrieved dynamically during answer evaluation.

### 2.2 Technology Stack & Dependencies

- **Backend:** Python 3.10+, FastAPI, SQLAlchemy, PostgreSQL, Alembic.
- **AI/ML:** Google Cloud Vision, Gemini API ( `google-genai` ), Qdrant, LangChain.
- **Frontend:** React 19, Tailwind CSS, Vite, React Router v7, Axios.

### 2.3 Data Design & Database Schema Operations

- **Users:** Stores `id` , `email` , `role` (Teacher/Student), `full_name` , `grade` .
- **Textbooks:** Stores metadata and file paths for RAG vector embeddings.
- **QuestionPapers / Questions:** Relational storage for paper structures, answer keys, and max marks.
- **AnswerSubmissions / Evaluations:** Tracks student upload statuses, total marks, and granular per-question evaluations.
- **Notifications:** polymorphic target tracking (unread, read, clears).

### 2.4 Code Structure Overview

```
k12-answer-evaluator/
├── backend/
│   ├── app/
│   │   ├── api/               # FastAPI Controllers (routers)
│   │   ├── core/              # App config, DB session, security logic
│   │   ├── models/            # SQLAlchemy DB schemas
│   │   ├── schemas/           # Pydantic validation schemas
│   │   ├── services/          # Heavy lifting (OCR, Evaluator AI, RAG ingestion)
│   │   └── main.py            # FastAPI App definition
├── frontend/
│   ├── src/
│   │   ├── components/        # React Views (Auth, Teacher UI, Student UI)
│   │   ├── context/           # Global States (Auth, Theme, Notifications)
│   │   ├── services/          # API handlers (Axios interface)
```

---

## 3. Usage and Deployment Guides

### 3.1 Setup / Installation Steps

1. **Clone the Repository:**

```
git clone https://github.com/your-org/project-k12.git
cd project-k12
```

2. **Backend Setup:**

```
cd k12-answer-evaluator/backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

3. **Environment Variables ( `backend/.env` ):** Ensure you specify `DATABASE_URL` , `GEMINI_API_KEY` , `GOOGLE_APPLICATION_CREDENTIALS` , and `SECRET_KEY` .
4. **Database Migrations:**

```
alembic upgrade head
```

5. **Frontend Setup:**

```
cd ../../frontend
npm install
```

### 3.2 Running the Application

- **Start Backend:** `uvicorn app.main:app --reload` (Runs on port 8000)
- **Start Frontend:** `npm run dev` (Runs on port 5173/5174)

### 3.3 Quick Start Tutorial & End-User Workflow

1. **Teacher Setup:** Sign up as a Teacher, navigate to "Add Textbook" and upload a reference PDF. Then, create a new Question Paper.
2. **Student Action:** Sign up as a Student. Go to Dashboard -> Profiles to specify your Grade. Then navigate to Assessments and upload an Answer Sheet image.
3. **Completion:** The teacher logs in, reviews the auto-graded sheet, makes overrides if necessary. The student downloads the PDF Report.

### 3.4 Troubleshooting FAQs

- **Q:** Why is the AI taking 20+ seconds to evaluate?
  - **A:** Network latency with Gemini API. Heavy OCR extractions inherently take time. It runs in the background.
- **Q:** Why is Enum data corrupting operations?
  - **A:** Ensure PostgreSQL DB Enum cases match Python schemas exactly (we force everything to lowercase).

---

## 4. API Documentation

*Full reference of REST endpoints:*

- `/api/auth/register` (POST) - Account creation
- `/api/auth/login` (POST) - Token retrieval
- `/api/auth/profile` (PATCH) - Update user data (Name, Grade)
- `/api/teacher/papers` (GET, POST, PUT, DELETE) - Paper CRUD
- `/api/teacher/papers/from-image` (POST) - OCR/AI Paper generation
- `/api/teacher/extract-questions` (POST) - Test OCR parsing
- `/api/student/papers` (GET) - View assigned papers
- `/api/student/submit/{paper_id}` (POST) - File upload & eval trigger
- `/api/v2/notifications` (GET, PATCH, DELETE) - Real-time alerts
- `/api/v2/papers/{id}/leaderboard` (GET) - Rank listings
- `/api/v2/submissions/{id}/report` (GET) - PDF stream

---

## 5. Maintenance Essentials

- **Contribution Guidelines:** All contributors must branch from `main`, use standard commit messages ( `feat:` , `fix:` , `docs:` ), and submit a Pull Request.
- **License:** MIT License.
- **Risk Register:** Kept internally by the project manager.

---

## 6. Project README Overview

# K12 Answer Evaluator


## Overview

The **K12 Answer Evaluator** is a cutting-edge educational platform designed to automate the evaluation of handwritten student answer sheets. It leverages Google Cloud Vision for OCR, Gemini AI for intelligent grading against an answer key, and an advanced Retrieval-Augmented Generation (RAG) system to query textbook content for accurate feedback.

It provides two distinct portals:

- **Teacher Portal:** Central hub to extract, create, and assign question papers. Gives teachers final authority to override AI evaluations.
- **Student Portal:** A unified, glassmorphism-themed UI where students can upload handwritten answers, monitor tracking indicators, analyze subject-wise trends, and download detailed PDF report cards.

## Hackathon Criteria Fulfillment

This project was built to directly address the K12 AI Evaluation Hackathon problem statement.

- **Target:** CBSE/Matriculation K12 education (configured via `class_level` and `subject` ).
- **What is wrong & Why it is wrong:** The AI parses `errors` into granular `[what, why, impact]` breakdowns.
- **Missing Concepts & Improvement:** Outputs `missing_concepts` and `improvement_guidance` to tell students exactly how to improve.
- **Correct Expected Answer:** Outputs `correct_answer_should_include` compared against the student's submission.
- **Working Flow:** End-to-end integration (Upload -> OCR -> Vector DB RAG -> Gemini LLM Evaluation -> Score/Feedback Generation -> Student Dashboard).

---

## 7. Changelog

All notable changes to this project will be documented in this section. The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

### [2.0.0] - 2026-02-21

**Added**

- **Core AI Evaluation**: Completely revamped the `evaluate_submission` function to utilize Google Gemini via `google-genai` and `langchain` components instead of OpenAI.
- **OCR System Integration**: Integrated Google Cloud Vision to robustly parse handwritten text, checkboxes, diagrams, and mathematical symbols directly from image uploads.
- **Knowledge Base (RAG)**: Developed a textbook processing pipeline uploading documents into Qdrant for context-aware grading.
- **Student Performance Dashboard**: Added dynamic progress tracking, bar charts, and historical submission timelines.
- **Interactive Reports & Leaderboards**: Added PDF Report Card streaming ( `reportlab` ) and class ranking leaderboards per paper.
- **Notification System**: Added real-time polling to inform students of evaluation completions and new assignments.
- **Student Profile Management**: Added the ability for students to specify their `grade` level across the frontend UI and database.

**Fixed**

- **Database Enum Errors**: Resolved critical `DataError` exceptions related to PostgreSQL Enum capitalization by syncing SQLAlchemy definitions strictly to lowercase mappings.
- **Context Generation Chunking**: Fixed file ingestion bugs with unstructured text that previously triggered database transaction rollbacks.
- **Frontend State Synching**: Removed orphaned React hooks that blocked evaluation state refreshing when navigating to the results viewer.

---

# 8. Test Plan

## 8.1 Quality Assurance Objective

To ensure that the application functions seamlessly across its core user personas (student and teacher). The primary objective is to guarantee the robustness of the AI Evaluation process and prevent grade falsification or authentication breaches.

## 8.2 Test Strategies

### 8.2.1 Backend Unit & Integration Tests (Pytest)

- Validating the parsing integrity of our custom OCR pipeline.
- Ensuring `google_genai` prompts safely parse structured JSON without crashing under abnormal token inputs.
- Ensuring secure multi-role JWT route protection (Students cannot access Teacher Endpoints, vice versa).

### 8.2.2 Frontend Functional Testing

- Route protection guards are operating correctly ( `<ProtectedRoute>` component).
- Form validation effectively handles edge cases (e.g., uploading malformed images, missing text fields).
- Verifying the React State lifecycle for real-time polling updates when papers move to "Evaluated" state.

### 8.2.3 Evaluation Quality Testing

Testing the AI output against a benchmark suite of manual human-graded answer sheets to determine scoring drift. Ensure the `get_rag_context` effectively halts hallucinated grades by grounding answers to the supplied textbook embeddings.

### 8.3 Core Test Scenarios (Manual Regression)

**Authentication & Roles**

- [ ] User can register a Teacher account.
- [ ] User can register a Student account.
- [ ] User can log in with valid credentials.
- [ ] API rejects requests missing a JWT Bearer header.

**Teacher Workflow**

- [ ] Create a Question Paper manually.
- [ ] Extract question paper from an image/PDF (OCR Vision test).
- [ ] Upload a reference textbook (PDF RAG test).
- [ ] Override AI-given marks on a student submission.

**Student Workflow**

- [ ] View assigned / available papers in the dashboard.
- [ ] "Profile missing grade" badge redirects to the `student/profile` route properly.
- [ ] Uploading 1-3 images successfully initiates a background task without hitting upload timeouts.
- [ ] View granular line-by-line feedback.
- [ ] Download a PDF Report Card that accurately totals the marks.