# Software Requirements Specification for AI-Assisted E-Learning Platform

## 1. Introduction

This Software Requirements Specification (SRS) describes the requirements for a **MERN (MongoDB, Express, React, Node)** based E-Learning platform enhanced with **Google Gemini** generative AI integration. The platform is intended to support students, teachers, parents, and administrators by providing automated grading, personalized feedback, adaptive learning recommendations, and analytics. It leverages modern web technologies (the MERN stack) and Google’s Gemini AI to deliver a smarter and more personalized education experience. The document follows the IEEE SRS format and is intended for project stakeholders including developers, project managers, and client representatives.

### 1.1 Purpose

The purpose of this SRS is to fully capture the functional and non-functional requirements of the AI-Assisted E-Learning Platform. It will serve as a contract between the development team and the stakeholders, ensuring that the delivered system meets the agreed expectations. This document will:

* Clearly define the platform’s features such as AI-driven grading, feedback generation, adaptive learning suggestions, and multi-portal access for different user roles.
* Specify interface requirements for integrating the external **Google Gemini API** for generative AI services.
* Detail the system’s constraints, quality attributes (security, scalability, reusability, etc.), and other factors critical to deployment for clients.
* Provide use case analysis and high-level system architecture to guide design and implementation.

### 1.2 Scope

The system to be developed is an **AI-Assisted E-Learning Platform** that will be used by educational institutions or corporate training programs to facilitate assignments, grading, and personalized learning. Key capabilities in scope include:

* **Automated Assignment Grading:** The platform will accept student assignment submissions (text-based, and potentially images for handwritten work) and automatically grade them using the Google **Gemini** AI model. The AI will generate a grade or score as well as detailed feedback on the submission.
* **Personalized Feedback and Improvement Suggestions:** For each graded submission, the system will provide constructive comments and suggestions for improvement, phrased in an encouraging manner (simulating a teacher’s personalized feedback). This feedback is generated by the AI in alignment with provided rubrics or performance criteria.
* **Adaptive Learning Path Recommendations:** Based on a student’s performance history, the system will recommend the next topic or lesson for the student to focus on. For example, if a student struggles with fractions, it may suggest reviewing decimals or related foundational topics next. This ensures an adaptive learning path tailored to each student.
* **Multi-Portal User Interface:** Different front-end portals will be available for **Students**, **Teachers**, **Parents**, and **Administrators**. Each portal provides role-specific functionalities (e.g., students submit work and view feedback, teachers create assignments and review AI gradings, parents view their child’s progress, admins manage the system and view overall analytics).
* **Leaderboard and Gamification:** The platform will maintain a leaderboard or similar gamified element to display top performers or completion streaks, fostering a motivating competitive environment among students.
* **Administrative Analytics and Reporting:** The system will compile analytics such as class performance summaries, improvement trends, and usage statistics, accessible to administrators (and in a limited way to teachers and parents). This may include AI-generated summaries of class progress or areas where many students struggle.
* **Integration with External Services:** The core AI integration is with Google’s Gemini API for generative content (for grading, feedback, etc.). Additionally, for hosting and storage, the system will use cloud services (Netlify or similar for front-end hosting, Render or similar for back-end deployment, and Cloudinary or equivalent for storing uploaded files/images). These integrations and any associated constraints are within scope.

Out of scope (for this version) are certain optional or future enhancements, such as voice feedback (text-to-speech motivation), grading of completely arbitrary content (beyond the structured assignments), or extensive Learning Management System (LMS) features beyond assignments and basic course content. However, some of these are noted in Appendix C as potential future requirements.

### 1.3 Definitions, Acronyms, and Abbreviations

* **AI:** Artificial Intelligence.
* **LLM:** Large Language Model. In this context, refers to advanced AI models like Google Gemini capable of understanding/generating text (and other modalities in some cases).
* **Gemini API:** Google’s Generative Language API, specifically the *Gemini Pro* model used for content generation and analysis[[1]](https://ai.google.dev/competition/projects/grademate#:~:text=GradeMate%20is%20an%20advanced%20app,performance%20and%20improve%20their%20skills). This API is part of Google’s generative AI services, used here for grading and feedback.
* **MERN:** An acronym for the technology stack MongoDB, Express.js, React.js, Node.js.
* **MongoDB:** A NoSQL document-oriented database used for storing application data (users, assignments, logs, etc.).
* **Express.js:** A web application framework for Node.js, used to build the back-end API.
* **React.js:** A front-end JavaScript library for building user interfaces (used for the client-side portals).
* **Node.js:** A JavaScript runtime for executing server-side code (runs the Express server).
* **Portal:** In this document, refers to a role-specific front-end interface of the system (Student Portal, Teacher Portal, Parent Portal, Admin Dashboard).
* **Assignment:** A task or exam (could be an essay, quiz, etc.) that a student submits for grading.
* **AI Grading:** Automated evaluation of a student’s submission by the AI system, producing a grade and feedback.
* **Adaptive Learning:** The system’s capability to adjust the learning content or suggestions based on the learner’s past performance.
* **SRS:** Software Requirements Specification.
* **API:** Application Programming Interface, how different software components communicate. Specifically, the Gemini API is a RESTful service used by our system’s backend.
* **OCR:** Optical Character Recognition, a technology to convert scanned images (e.g., of handwritten text) into machine-readable text. (OCR is mentioned as a possible addition for reading handwritten submissions via Gemini Vision API in future enhancements.)

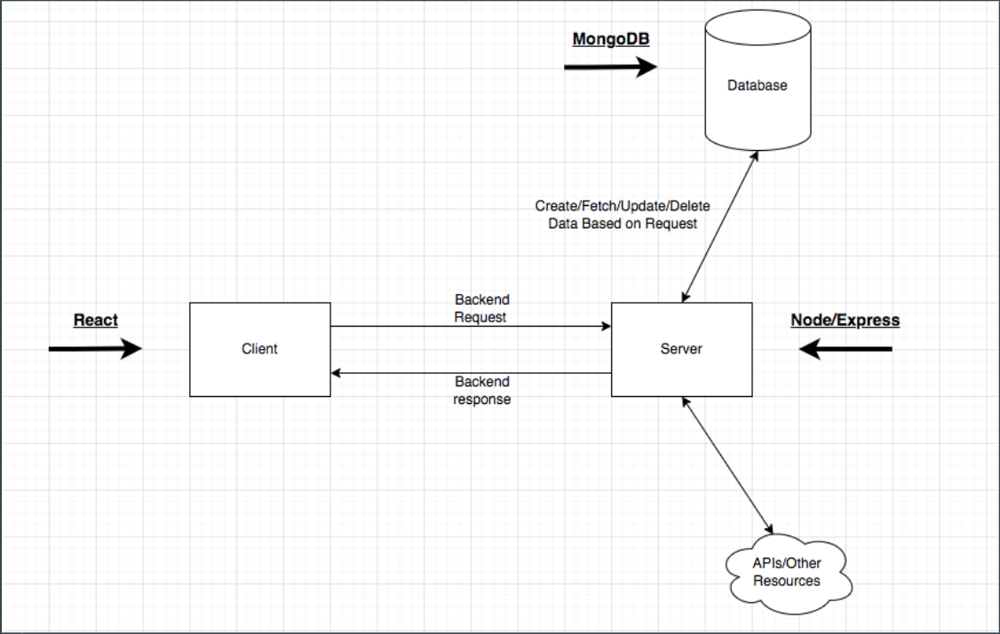
### 1.4 Overview

The rest of this SRS is organized as follows: Section 2 gives an overall description of the product, including its context in a larger system, major functions, user characteristics, and constraints. Section 3 details external interface requirements, describing user interfaces and how the system interacts with hardware, other software, and communication protocols. Section 4 delineates the system’s features in detail, each broken into descriptions, stimuli/response (use case scenarios), and specific functional requirements. Section 5 covers non-functional requirements and qualities such as performance, security, scalability, and maintainability. Section 6 notes any other requirements (such as legal or regulatory considerations). The document concludes with Appendices that provide a glossary of terms, and analysis models (like use case and UML diagrams) to support the requirements.

## 2. Overall Description

### 2.1 Product Perspective

The AI-Assisted E-Learning Platform is an advanced web-based system that builds upon a traditional Learning Management System (LMS) by integrating artificial intelligence for enhanced functionality. It can be viewed as a **self-contained product** that also fits into a larger ecosystem of educational technology tools. This product is **new** (not a direct replacement of an existing system), but it is inspired by conventional LMS and grading systems, augmented with AI capabilities. It follows a multi-tier architecture typical of web applications with an AI layer added:

  
**Figure 1: High-Level System Architecture.** The platform uses a client-server model. The **React** front-end (client) interacts with a **Node/Express** back-end (server) through RESTful API calls. The server communicates with a **MongoDB** database for data storage (users, assignments, results) and with external AI services via HTTP requests. The Google **Gemini** AI service is an external component (as depicted by the “APIs/Other Resources” cloud) that the server calls for grading and feedback generation. The entire system is typically hosted on cloud platforms (e.g., front-end on Netlify, back-end on Render) and uses cloud storage (like Cloudinary) for any uploaded files.

From a **product family** perspective, this platform could be seen as part of a family of e-learning solutions or intelligent tutoring systems. If considering integration, it could interface with other systems such as a school’s existing student information system (for user data) or content repositories, but those are beyond the current scope. The context diagram above shows that the **Gemini AI API** is a critical external interface – the success of AI features depends on it. The platform’s requirements are influenced by the constraints and capabilities of this AI service (e.g., network calls, API key usage, model limits).

**Key components and their interactions**: - **Front-end (Client)** – Runs in the user’s web browser. For example, a student uses the React-based **Student Portal** to submit an assignment or view feedback. The portal sends requests to the backend (e.g., submit assignment data). - **Back-end (Server)** – The Node.js/Express application that exposes REST API endpoints (e.g., POST /api/assignments/submit, GET /api/assignments/{id}/grade). It processes incoming requests, applies business logic, and interacts with the database and external services. Notably, when grading is required, the back-end formulates a prompt and calls the Gemini API. - **Database** – MongoDB stores persistent data: user accounts (with roles and profile info), assignments and their contents (questions, model answers, etc.), student submissions, grades and feedback, and logs of AI interactions (for auditing or improvement). - **AI Service (Gemini)** – An external cloud service accessed via API. For example, when a student submission comes in, the back-end will send the submission text along with grading instructions to the Gemini model, which returns the evaluation (score and feedback). This integration is encapsulated in a service module (e.g., geminiService.js in the code) that handles constructing API calls and parsing responses. - **Hosting & Deployment** – The architecture allows the front-end to be decoupled and served statically (Netlify ), while the back-end runs on a server or platform (like Render or any Node-friendly host). File uploads (like images or PDFs for assignments) may be stored on a service such as Cloudinary or AWS S3; our design assumes such integration but specifics are a deployment detail.

This system is **intended to operate standalone** for a client (e.g., a school or company) – it’s not a module inside a larger software, but it can exchange data with others if needed via APIs. Interfaces between components are clearly defined: e.g., the front-end and back-end speak HTTP+JSON, the back-end and AI speak via Google’s REST API with JSON payloads, and the back-end and database use MongoDB drivers.

In summary, the product’s perspective is that of a modern web application enhanced with an AI layer. It draws on known patterns (MERN stack architecture) and novel integration (Gemini AI) to achieve its goals. The **context diagram** (Figure 1) illustrated major subsystems and external interfaces, showing that our system sits between users (multiple roles) and an AI cloud service, orchestrating interactions among them.

### 2.2 Product Functions

At a high level, the platform provides the following major functional capabilities (organized by feature sets):

* **Assignment Management:** Teachers can create and publish assignments or quizzes. Students can submit responses to these assignments through their portal. The system stores assignment details and submissions.
* **Automated Grading (AI Grading):** For submitted assignments (especially essay or open-ended responses), the system automatically generates grades and evaluation using the AI engine. This includes:
* Scoring the submission (e.g., numeric grade or percentage).
* Generating text feedback explaining the score, highlighting good points and areas to improve.
* **Feedback and Guidance:** Beyond grading, the system provides constructive feedback messages. This may include pointing out errors, giving hints for improvement, and encouraging the student. It can also generate **personalized study tips** (e.g., “Review Chapter 3 on fractions to strengthen this skill”).
* **Adaptive Learning Suggestions:** The platform analyzes a student’s performance data (grades over time, which topics they excel or struggle in) and uses AI to recommend what the student should study next. For example, after a math quiz, it might suggest “Practice decimal addition next, since you scored low on fraction addition.” This helps create a tailored learning path for each student.
* **Leaderboard & Gamification:** The system tracks points or scores (for example, based on quiz performance or task completion) and displays a leaderboard of top students (could be for a class or overall, depending on settings). This fosters a competitive yet fun environment. It may also award badges or achievements (e.g., “Completed 5 assignments in a row on time”).
* **Progress Analytics and Reports:** For teachers and admins, the platform provides analytical dashboards. This includes class averages, distribution of scores, identification of commonly missed questions or topics, and tracking of improvement over time. Parents have access to a simplified report of their own child’s progress (e.g., recent scores, overall performance trend, any AI recommendations).
* **User Management and Roles:** Administrators can manage user accounts and roles (creating teacher, student accounts, etc.). The system supports authentication (login) and role-based access control, ensuring each portal only allows appropriate actions. Basic profile management (updating password, profile info) is provided.
* **Content Creation Aids (Teacher Assistant):** *(Optional/advanced feature)* The system can assist teachers in creating content. For example, a teacher can prompt the AI to “generate 5 quiz questions on fractions for Grade 5,” and the AI will output suggested questions with answers. This helps in rapid quiz or worksheet generation. This feature is considered lower priority and can be enabled as needed.
* **Administrative Tools:** Admin users can configure certain system settings (like AI thresholds, curriculum mappings for recommendations, etc.), and oversee system usage. They can also trigger or view logs of AI interactions for auditing (to ensure the AI’s feedback is appropriate and no harmful content is produced).

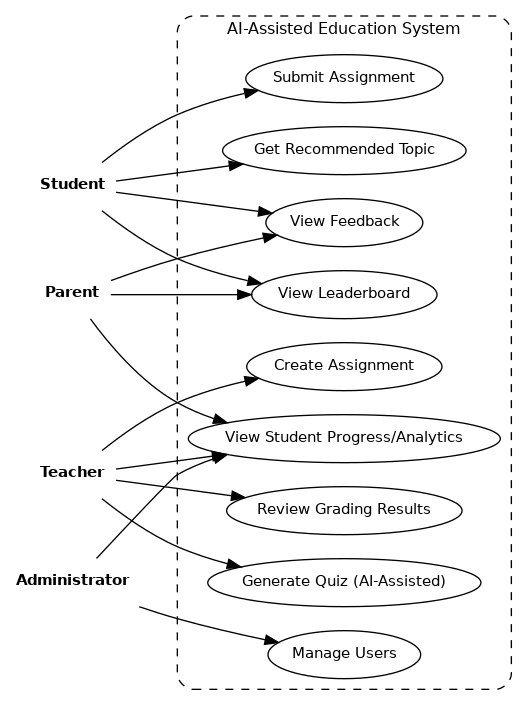
The above functions can also be grouped by user role for clarity (the primary use cases of each actor in the system):

* **Student Functions:** Submit assignments/quizzes, view grades and AI feedback, get recommended next topics, view personal progress and class leaderboard, manage own profile.
* **Teacher Functions:** Create assignments/quizzes, view student submissions and AI-assigned grades, override or adjust grades if needed, review AI feedback given to students, generate additional feedback or hints, view class analytics, optionally use AI to help create new content (questions, exercises).
* **Parent Functions:** View their child’s grades and feedback, see overall progress and perhaps compare to class average, receive AI-generated tips on how to support their child’s learning (e.g., “Your child could benefit from practicing reading 15 minutes daily” – this could be a possible feature).
* **Administrator Functions:** Manage all users and data, configure system (e.g., academic terms, subjects, AI API keys, etc.), view aggregate analytics across the platform, ensure security and data privacy compliance, and manage content repositories.

The **major system features** in Section 4 will break down these functions into detailed requirements. Each feature corresponds to one or more of the above functional areas (e.g., AI Grading, Adaptive Recommendation, etc. will be detailed with specific requirements). For clarity, the system features will also reference the **use case scenarios** involved.

**Key Use Cases Overview:** The primary interactions in the system can be summarized via typical use cases: - A student submits an assignment answer and immediately receives an AI-generated grade with feedback. - A student reviews the feedback and then requests a recommended study topic or additional practice generated by the AI. - A teacher creates a new assignment on the portal (possibly using AI suggestions for questions), which is then available for students. - A teacher views an AI-generated grading report for a submitted assignment (e.g., all student submissions graded, with an overview of how the class did and common mistakes identified). - A parent logs in to see their child’s latest results and reads the feedback the child got, as well as an overall progress report. - An administrator adds a new teacher account, updates system-wide settings (like enabling/disabling the AI content generation feature for teachers), and checks the security logs or system metrics.

We illustrate the relationships between users and these functions in a UML use case diagram (see Figure 2). This diagram shows which actor performs which actions in the system.

  
**Figure 2: Use Case Diagram for the AI-Assisted Education System.** Each actor (Student, Teacher, Parent, Administrator) has specific use cases they participate in. For example, Students can *Submit Assignment*, *View Feedback*, *Get Recommended Topic*, etc., while Teachers can *Create Assignment*, *Review Grading Results* (from AI), *Generate Quiz (AI-Assisted)*, etc. Parents primarily *View Feedback*, *View Leaderboard* and *View Student Progress/Analytics* for their child. The Administrator oversees management (*Manage Users*) and can also view high-level analytics.

*(The use case diagram above provides a high-level visualization of system functionality. Detailed functional requirements for each of these actions are given in Section 4.)*

### 2.3 User Classes and Characteristics

The system will be used by several distinct user classes, each with different needs and technical skill levels:

* **Students:** The primary end-users who are learners (e.g., K-12 students, university students, or trainees in a corporate scenario). Characteristics:
* Typically have basic to moderate computer skills (depending on age group). The UI for students should be very intuitive and engaging.
* They use the system frequently (possibly daily or weekly) to submit homework, take quizzes, and review feedback.
* Their main concern is ease of use and clarity of feedback. Many students may not be very technical, so the system should guide them clearly (especially younger students).
* Security/Privileges: Students should only access their own data (their assignments, their scores) and general class info like leaderboard. They should not see other students’ detailed data (except possibly comparative ranks if leaderboard enabled).
* **Teachers/Instructors:** Users who manage courses, assignments, and oversee student progress.
* Generally tech-savvy in using educational tools, but not necessarily programmers. The interface for teachers can assume moderate proficiency.
* Frequency of use: daily or weekly, whenever managing class or grading.
* Teachers need more advanced features (content creation tools, analytics) but these should be presented in a user-friendly manner. They appreciate time-saving automation (hence the AI grading is a major benefit).
* Privileges: A teacher should have access to all data of the students in their class(es), including viewing all submissions and AI feedback. They can override grades if needed. They cannot see data for classes they do not teach (unless also an admin).
* **Parents/Guardians:** Users who monitor a student’s progress (primarily in K-12 or maybe training supervisors in corporate context).
* Likely infrequent users, logging in perhaps weekly or when notified of a new report. The interface must be very straightforward, focusing on results and insights without requiring technical skill.
* They are mostly interested in summaries rather than raw details – e.g., overall grade trends, feedback highlights, and how they can help their child.
* Privileges: Parents can only view information pertaining to their own child(ren). No editing capabilities; mostly read-only access to reports and maybe message features.
* **Administrators:** Users responsible for maintaining the system (could be an IT admin at a school or the software’s system admin).
* They are technically proficient. They might be responsible for onboarding new users, maintaining system configurations, and ensuring data security.
* Frequency of use: a few times a week or as needed for administrative tasks.
* Privileges: Full access to all system functions and data. Admin can create or remove user accounts, assign roles (teacher, student, etc.), configure global settings (like the AI API keys, system preferences), and view any report. They can also manage content libraries if applicable and oversee system health.

Each user class has unique requirements which have been considered in the functional design: - For example, **Students** require a fun, simple UI and immediate feedback (hence the AI’s instant grading is valuable). - **Teachers** need reliability and ability to intervene in the AI process if needed (e.g., edit a feedback or adjust a grade, since AI might not always be perfect). - **Parents** need clarity and reassurance from the feedback (the system’s tone for parents should be reassuring and constructive about their child). - **Admins** need security and control (they care about data privacy, making sure the AI doesn’t leak data outside, etc.).

These considerations will be reflected in the requirements (e.g., requirements ensuring role-based access, editing rights for teachers, etc., see Section 4 and 5.3 on security).

### 2.4 Operating Environment

The system is a **web application** and will primarily operate in the following environments:

* **Client-Side:** Any modern web browser (Chrome, Firefox, Safari, Edge) on a desktop or laptop computer. The front-end React application will be optimized for desktop use (for teachers and admins especially) and responsive for mobile or tablet use (students and parents may use tablets or phones). The UI should follow modern web standards (HTML5, CSS3, ECMAScript 2020+). There may be future dedicated mobile apps, but currently the web responsive design is expected to suffice.
* **Server-Side:** A Node.js runtime environment (version 16.x or above, or later stable LTS version at time of deployment) running on a server. This could be a cloud server or platform-as-a-service (like Render.com or Heroku). The server will run on a standard OS (Linux-based preferred for deployment). It must support Node, have internet connectivity to reach the Gemini API, and connect to the database.
* **Database:** MongoDB 4.x or above, either self-hosted (on a VM or container) or a cloud MongoDB service (like MongoDB Atlas). The environment should allow network access between the Node server and the MongoDB instance. Storage needs: depends on number of users and assignments, but likely moderate (text data and logs, some media for assignments).
* **External AI Service:** The Google Generative Language API (Gemini model) is cloud-hosted by Google. Our system connects to it over the internet via HTTPS. So the operating environment for AI is essentially Google’s servers; our system just needs the ability to make requests (outbound internet calls). It requires an API key provided by Google, which must be kept secure on our server (likely stored in environment variables or a secure config file).
* **Hosting & Deployment:** The front-end will be built (bundle of HTML/CSS/JS) and can be hosted on static site hosts like Netlify or Vercel. The Node/Express backend can be hosted on services like Render.com, AWS Elastic Beanstalk, Azure, or similar. Cloudinary (or an equivalent) may be used for storing uploaded images (if any, like student upload of a photo of homework). These hosting environments impose certain constraints (for example, Netlify has a build environment for React, Render has 512MB default instance memory etc.), which the app should operate within.
* **Hardware considerations:** There is no special hardware required on the client beyond a device capable of running a modern browser and (optionally) a scanner or camera if uploading handwritten work. On the server side, typical cloud server hardware (1-2 vCPUs, 1-2 GB RAM to start, scalable as user load grows) is expected. The system should be able to scale horizontally (multiple server instances behind a load balancer) as user count increases.

In summary, the system operates in a typical web/cloud environment: user-facing on web browsers, server running in cloud, and heavy reliance on internet connectivity for the AI service. It needs to **peacefully coexist** with standard network infrastructure (firewalls allowing outgoing HTTPS to Google APIs, etc.) and possibly with organizational IT policies (for example, schools might require the system to run on premises – if so, that could be accommodated with on-prem servers and local installation of Node and Mongo, but primary assumption is cloud deployment).

### 2.5 Design and Implementation Constraints

Several constraints influence the development and implementation of the platform:

* **Technology Stack Choices:** The system **must use the MERN stack** (this is a given constraint, perhaps due to team expertise or existing code):
* The front-end is constrained to use **React.js**. This means the UI will be a single-page application, and we will use JavaScript/TypeScript and possibly frameworks like Redux for state management.
* The back-end is constrained to **Node.js with Express**. All server logic will be in JavaScript (or TypeScript) on Node. We will follow an MVC/service architecture separating routes, controllers, services as per standard practice.
* **MongoDB** must be used for data persistence. This implies data will be stored in JSON-like documents and may require careful design for relationships (since it’s NoSQL). It also means certain transactional constraints of SQL are not available, though we gain flexibility and scalability.
* **Gemini API Integration:** The use of **Google’s Gemini API** imposes constraints:
* We must adhere to Google’s API usage policies (rate limits, content guidelines). For instance, we need to ensure we don’t exceed the allowed number of requests per minute. We may need to queue or throttle grading requests if an entire class submits simultaneously.
* Network connectivity and latency: Each grading request requires an external API call, which might take some seconds to return (depending on AI model speed and internet latency). The system design must handle this asynchronously (so as not to block the server or user interface). Possibly use async job queues or loading indicators on UI.
* **Prompt formatting**: The AI’s output quality depends on how we prompt it. The development has a constraint to craft effective prompts (like including the assignment question, model answer or rubric, and student answer in a structured format) to get reliable grading[[5]](https://medium.com/@aarathisree.1535/edu-evaluator-how-i-built-an-ai-that-grades-handwritten-answer-sheets-using-flask-and-gemini-ee8665d61240#:~:text=Sample%20Gemini%20Prompt). This might not affect end-user requirement directly, but it’s a design constraint for developers: maintaining prompt templates in code (as seen in geminiService.js example).
* **Security/Privacy Regulations:** If used in a school setting, there are likely privacy laws (e.g., FERPA in the US, or GDPR if in Europe for student data). Implementation must ensure:
* Student data (personal info, grades, submissions) is stored securely (encrypted in transit, protected at rest).
* The AI service (Gemini) should not receive any more personal data than necessary. Ideally, we send only the content of the assignment answer and maybe an anonymized student ID for context, but not names or sensitive info. This might constrain how we use the API (to avoid sending personal identifiers to a third-party service).
* We may need to allow data deletion (parents might request deletion of their child’s data, etc.). Using MongoDB, we must design schemas to easily remove or anonymize data if needed.
* **Corporate/Organizational Policies:** The client (company or school) might have specific constraints like:
* Use of only approved cloud services (perhaps they insist on Azure instead of Google for AI, etc.). However, since Gemini is from Google, if an organization disallows that, it would be a major constraint. We assume our client approves the use of Google’s service for this project.
* Coding standards or frameworks: e.g., if the dev team has a standard logging library or error handling framework, we must incorporate those.
* Accessibility standards: The UI might need to follow WCAG guidelines for accessibility if it’s a public education tool.
* **Hardware Limitations:** On the server, if initially using a minimal hosting tier, we might have memory limitations. For example, processing a very large text through the AI might be constrained by memory/time. We have to ensure that the system can handle typical lengths of assignments (say essays of a few thousand words) but extremely large inputs might need to be truncated or disallowed.
* **Parallel Operations:** The system should support multiple users performing actions concurrently (multiple students submitting at once). We must design for concurrency (Node is single-threaded but can handle many requests via async IO; if heavy CPU work emerges, might need clustering).
* **Development Timeline and Resources:** If this is a constraint, perhaps we only have a certain amount of time to implement. As a result, some features are optional or deferred (like the Teacher Assistant feature is optional). We prioritize core features first (grading, feedback, portals) to meet deadlines for client demos.
* **Programming Standards:** The organization may enforce certain conventions (for example, code must be in TypeScript, or must pass ESLint rules, etc.). Also, using certain versions of libraries (maybe React 18+, Node 18+, etc.). All these set boundaries on our implementation approach.
* **Integration with Legacy Data:** If this system replaces a manual or older process, there might be an import of existing data (past grades, assignments). We might constrain design to include migration scripts or at least structure data flexibly to accommodate manual backfill.
* **Deployment Constraints:** Using Netlify/Render means:
* The frontend must be a static build (no server-side rendering unless using another approach).
* The backend on Render might sleep when idle (if using free tier), causing a delay on first request – stakeholders should be aware or we use paid tier to avoid that.
* Cloudinary usage means we must limit file sizes to their free tier limits (for example, max 10 MB images or such) unless the client has a subscription.

In summary, these constraints ensure the system is built with the chosen stack and within external limits (AI API, legal, hosting). They are not negotiable and the design must work around them (e.g., building robust error handling for AI calls that might fail due to rate limit or network issues).

### 2.6 User Documentation

The following documentation will be provided with the software for end-users and administrators:

* **User Manual (Student & Parent):** A simple guide (possibly web-based or PDF) explaining how students can log in, submit assignments, view feedback, interpret the AI feedback and suggestions, and how parents can log in to view progress. It will include screenshots of the Student Portal and Parent Portal, pointing out key elements like where to see grades or comments.
* **Teacher Manual:** Documentation focused on instructors, explaining how to create assignments, how the AI grading works (e.g., that it’s automatic upon submission, but teachers can edit the grades), how to view class analytics, and how to use the AI assistance features (like generating quiz questions). This may also contain tips on writing good prompts for the AI if they use the content generation tool.
* **Admin Guide:** Documentation for system administrators covering user management (creating accounts, resetting passwords), configuration of AI integration (how to input the API key, adjust AI settings if any), and steps to troubleshoot common issues (e.g., “What to do if the AI service is down” or checking logs).
* **Online Help and Tooltips:** The application will have built-in help features, such as tooltips or info icons next to complex features (for example, an info icon next to an AI-generated grade explaining “This grade was generated by an AI based on the rubric. You can adjust it if necessary.”). There may also be an FAQ page within the app for common questions about the AI (“How does the AI decide my score?”, “Why did my feedback say to review a topic?” etc.).
* **Installation/Deployment Guide:** (For the client’s IT/DevOps) If the system is delivered for on-premise deployment, documentation on how to set up the environment, install Node, the database, configure environment variables (like the Gemini API key), and deploy the front-end will be provided. If it’s a cloud-hosted SaaS, this may not be needed by client, but internal dev team will maintain it.
* **API Documentation (if applicable):** If the system exposes any APIs (for example, maybe an API for integrating grades into another system), those endpoints will be documented. Currently, this platform is primarily self-contained, so this might just be internal documentation.

### 2.7 Assumptions and Dependencies

 **Internet Required:**  
The system depends on a stable internet connection because AI grading calls use the external Gemini API. If offline, users won’t get instant feedback.

 **Typed Submissions Only (for Now):**  
Students are expected to submit typed text. Support for scanned or handwritten work using OCR (e.g., Gemini Vision) may come later.

 **Gemini API Dependency:**  
The system relies on Google Gemini being available, stable, and affordable. If Gemini changes or fails, prompts may be adjusted or another AI may replace it.

 **Moderate Initial User Scale:**  
The first version targets a single school or a few hundred users. Scalability will be added later as usage grows.

 **Human Oversight:**  
Teachers supervise AI grading and can review or correct it. AI acts as an assistant, not a final authority in grading.

 **User Consent for AI Use:**  
All users (or guardians) are assumed to consent to AI analyzing their work. This must be clearly stated in user agreements for ethical compliance.

**Dependencies:** - **Google Gemini API service**: The system relies on several key dependencies. Its core component is the **Google Gemini API** for AI grading, meaning any changes or downtime in the service require updates. It also depends on **third-party packages** (e.g., Express, node-fetch, Axios) and **cloud services** like Netlify, Render, and Cloudinary for deployment—any issues with these could affect performance or availability. The app assumes users have **modern browsers** supporting ES6, and if notifications are added later, **email/SMS services** may be required. Overall, system reliability depends on stable AI and cloud infrastructure, with fallback mechanisms needed to handle outages.3. External Interface Requirements

This section describes the interactions between the software product and its users, hardware, other software systems, and communication protocols.

### 3.1 User Interfaces

The system provides web-based interfaces for four main roles — **students, teachers, parents, and administrators** — all designed to be responsive, consistent, and user-friendly.

* **Student Portal:**  
  Dashboard for upcoming assignments, feedback, and leaderboard; assignment submission form; feedback with comments and icons; progress and recommended topics; simple profile page.  
  *Tone:* Friendly, gamified, error messages in plain language.
* **Teacher Portal:**  
  Dashboard with class stats, assignment creation with optional AI question generator, submission review (approve/edit AI grades), analytics with charts, and notifications.  
  *Navigation:* Sidebar with clear sections.
* **Parent Portal:**  
  Overview of child’s performance, detailed feedback reports, comparison to class averages, and simple improvement suggestions.  
  *Design:* Minimal, mobile-friendly, read-only.
* **Admin Portal:**  
  Tools for user management, configuration (e.g., API keys, school logo), and system monitoring.  
  *Design:* Simple, secure, and technical.
* **General UI Standards:**  
  Responsive layouts, consistent color scheme and icons, accessibility features (contrast, font size, alt text), and clear navigation (header, breadcrumbs, help icon).

**3.2 Hardware Interfaces**

The system runs on **standard computing hardware** with no special devices required.

* **Client Devices:** Works on desktops, laptops, tablets, and smartphones via modern browsers.
* **Optional Devices:** Cameras/scanners may be used for uploads but not directly controlled by the app.
* **Server/Database:** Hosted on cloud infrastructure with standard CPU, memory, and network requirements.
* **Summary:** No special hardware; just ensure browser and network compatibility.

**3.3 Software Interfaces**

The software interacts with **MongoDB**, **Google Gemini API**, and the internal **Node.js/React** system.

* **Database (MongoDB):** Stores users, assignments, submissions, and grades; uses Mongoose or official drivers via secure connection URIs.
* **Gemini API:** Communicates over HTTPS; JSON requests/responses for AI grading; handles errors gracefully; encapsulated in a service module.
* **Front-End ↔ Back-End:** RESTful JSON APIs with endpoints for submitting assignments, fetching grades, and authentication via JWT/session.
* **Other APIs:** Future OCR or email integrations (e.g., SendGrid).
* **Summary:** Main workflow — React frontend → Node backend → MongoDB + Gemini API.

**3.4 Communication Interfaces**

Built entirely on **secure web standards** for reliable data exchange.

* **Protocols:** HTTPS (TLS 1.2/1.3) for all communications.
* **Data Format:** JSON for all requests and responses.
* **Error Handling:** Graceful messages for connectivity or API errors.
* **Security:** JWT/session authentication, encrypted data, and hidden API keys (server-side only).
* **Ports:** 443 (HTTPS), MongoDB 27017 internally.
* **Optional:** WebSockets or polling for real-time features.

**Summary:**  
All communications are encrypted, reliable, and standardized (REST + JSON), ensuring secure transfer of student data and AI results.

## 4. System Features

This section details the functional requirements organized by major system features. Each feature is described along with its priority and the typical sequence of interactions (stimulus/response), followed by enumerated specific requirements.

### 4.1 Feature: AI-Powered Assignment Grading

**Description and Priority:** This feature enables automatic grading of student assignments using the Gemini AI. It is a **High Priority** feature, as it is a core selling point of the platform (smart grading). Benefit: significantly reduces teacher workload and provides instant feedback to students. There is medium risk if the AI mis-grades, but mitigation is by teacher oversight. This feature is foundational to the adaptive learning that follows.

**Stimulus/Response Sequences:** 1. **Student Submission Trigger:** A student completes an assignment (for example, an essay question or open-ended quiz) on the Student Portal and clicks “Submit”. - *Stimulus:* The student’s action of submitting their work. - *Response:* The system receives the submission and acknowledges receipt (e.g., “Submission received, grading in progress…” message to student). 2. **AI Grading Process:** The back-end takes the submission content and formulates a prompt for the Gemini API (including any rubric or desired format for answer). The system calls the Gemini API with this prompt. - *Stimulus:* The system (back-end) sends a request to the AI service with the student’s answer. - *Response:* The AI service returns a grade and feedback content. For example, the AI might respond with text: “Grade: 8/10. Feedback: You covered the main points but missed...”. - The system parses this response. If the response is just prose, the system might extract a numeric grade by pattern or ensure the format. 3. **Return Results to Student:** Once grading is done (usually within a few seconds), the front-end either receives the result via the original request response or by polling. - *Stimulus:* The back-end triggers a completion, sending data to front-end. - *Response:* The student’s UI displays their **score** (e.g., 80%) and **feedback comments**. The feedback could be a few sentences or bullet points. The student can then read and potentially ask for clarifications (not in scope, but maybe they could click a “Why did I get this grade?” button if implemented). 4. **Teacher Review (if applicable):** Concurrently or later, the teacher can view the AI-assigned grades for the assignment. - *Stimulus:* Teacher visits the submission review page. - *Response:* The system shows each student’s grade and the AI feedback. The teacher may optionally click to **override** a grade if they disagree (e.g., change 8/10 to 9/10) and add a manual comment. This override triggers an update stored in the database (and could notify the student of the adjustment).

Edge cases: - If the AI service is unreachable or returns an error, the system might either queue the grading or inform the student that grading will be delayed. Possibly, it falls back to “Awaiting grade” status. - If a submission is in a format the AI can't handle (like an unsupported language or just gibberish), the system should handle gracefully, perhaps asking teacher to grade manually. - If a student submits after due date (depending on rules), it still grades but maybe flags as late.

**Functional Requirements:** (Each requirement uniquely identified for this feature, labeled as FG- for "grading")

* **REQ-FG-1:** The system **shall** accept a student’s assignment submission (text input or file upload) and store the submission data associated with that student and assignment.
* **REQ-FG-2:** Upon receiving a submission, the system **shall** automatically initiate an AI grading request. This involves constructing a prompt that includes the necessary context (problem description and the student’s answer). *Rationale:* To ensure the AI has full context for fair grading.
* **REQ-FG-3:** The system **shall** connect to the Google Gemini API with the prompt and **shall** retrieve an evaluation result. The result is expected to contain at least a score or grade and qualitative feedback[[4]](https://slejournal.springeropen.com/articles/10.1186/s40561-024-00310-z#:~:text=Gemini%20uses%20its%20advanced%20understanding,This%20technology%20is%20designed%20to).
* **REQ-FG-4:** The system **shall** parse the AI’s response to extract the grade (numerical or letter) and feedback comments. For example, if the AI response text contains a clearly identifiable score (like “7/10”), the system will capture that as the grade value. If the format is not as expected, the system should handle gracefully (maybe default to teacher grading needed).
* **REQ-FG-5:** The system **shall** present the grade and feedback to the student in the Student Portal UI immediately after it is available. The feedback **shall** be presented in a readable format (e.g., paragraphs or bullet points). Any important remarks in the feedback (like “plagiarism suspected” or “excellent work”) should be clearly visible (possibly highlighted).
* **REQ-FG-6:** The student **shall** be able to view not just the score but also the detailed feedback. The system should ensure both are visible together so the student can understand the reasoning behind the grade.
* **REQ-FG-7:** The system **shall** save the AI-generated grade and feedback in the database associated with the submission. This ensures that teachers and parents can view it later and that it is not lost if the page refreshes.
* **REQ-FG-8:** Teachers **shall** have the ability to review AI-graded submissions. The system **shall** provide an interface (in Teacher Portal) listing each student’s result for the assignment, including the AI’s assigned grade and feedback.
* **REQ-FG-9:** The system **shall** allow the teacher to override the AI-generated grade and/or append additional feedback. If a teacher changes a grade or feedback:
* It **shall** update the stored record of that grade.
* It **shall** update what the student sees (e.g., if the student checks later, they see the revised grade and teacher’s comments).
* It **shall** mark that grade as “adjusted by teacher” so it’s traceable.
* **REQ-FG-10:** If the AI fails to return a result (due to error or timeout), the system **shall** notify the student that grading is delayed and **shall** queue the submission for later processing. Additionally, an alert could be shown to the teacher on the dashboard (like “5 submissions require manual grading due to AI issues”).
* **REQ-FG-11:** The system **shall** enforce any grading rubric if provided. *(This might be a future extension: e.g., teacher uploads a rubric and the AI uses it.)* For now, the assumption is AI does holistic grading. If rubric-based grading is implemented, the AI prompt will include rubric criteria and the feedback may score each criterion.
* **REQ-FG-12:** The system **should** attempt to filter or moderate the AI’s feedback for appropriateness. The AI by default avoids harmful content, but if any profane or inappropriate text were returned (unlikely from Gemini due to safety measures), the system should sanitize it or hide it and flag for teacher review.
* **REQ-FG-13:** The grading process **shall** typically be completed within a reasonable time (target: under 10 seconds for a standard essay length). If it exceeds a timeout (say 30 seconds), the system should abort and notify user of delay (as per REQ-FG-10). *Rationale:* User experience – students shouldn’t wait indefinitely.
* **REQ-FG-14:** Only authorized triggers can call the AI grading. The system **shall not** expose the grading API directly to the client to avoid misuse; it must go through server verification that this is a legitimate submission from an authenticated student for a valid assignment.
* **REQ-FG-15:** The system **shall** log the AI grading transaction (student, assignment, timestamp, and the raw AI response or at least the prompt & result) for auditing. *Rationale:* This helps debug any issues or review AI performance later (and is useful if explaining grades to stakeholders).

**REQ-FG-1 to 15** ensure that the automated grading feature functions reliably, securely, and with oversight. This feature ties into other features: the output of grading is used in feedback display (Feature 4.2) and adaptive learning (Feature 4.3). It also intersects with non-functional requirements like performance (the speed of grading) and security (not exposing AI key, etc., see Section 5).

### 4.2 Feature: AI-Generated Feedback and Guidance

**Description and Priority:** This feature provides **personalized feedback** to students beyond just a grade. It’s essentially part of the grading output but is considered a separate highlight because of its educational importance. Priority is **High**, as it directly influences the learning outcome by telling students how to improve. It also includes encouraging tone and possibly guidance on what to study next (though specific next-topic suggestion is in adaptive learning feature). The feedback generator should mimic a supportive teacher’s comments.

**Stimulus/Response Sequences:** 1. After an assignment is graded (automatically via AI or even manually by teacher), the system prepares feedback: - *Stimulus:* A grading event is completed (trigger can be the AI returning a result). - *Response:* The system extracts or composes feedback. In AI’s case, the feedback text is part of the response (e.g., “Feedback: ...”). If a teacher overrides or adds notes, that becomes part of feedback. 2. The student views the feedback: - *Stimulus:* Student opens the results page for their submission (could be immediately post-submit or later via a “My Grades” section). - *Response:* The system displays the feedback message(s) clearly. For instance: “*Great introduction and good understanding of the concept. To improve, you should provide more examples in the second section.*” along with maybe an encouragement like “Keep up the good work!” (the AI can be instructed to include an encouraging remark). 3. (Optional) If the student doesn’t understand the feedback, they might reach out via a comment or request (not originally specified, but some systems allow a dialogue). We won’t implement a full Q&A with AI here, but a student could ask teacher in class or so. 4. Encouragement and positivity: - The system may append some general praise or motivation if the score is low (the AI can be prompted to be encouraging). If score is high, congratulate. - *Stimulus:* Low score scenario (e.g., < 60%). - *Response:* Feedback might include: “Don’t be discouraged. Focus on the areas mentioned and you’ll do better next time.” 5. Logging or viewing by others: - Teachers or parents viewing the student’s feedback: - *Stimulus:* Teacher or parent opens student’s submission details. - *Response:* They see the same feedback text that the student saw. This transparency helps everyone be on the same page regarding the student’s performance.

**Functional Requirements:** (Label FR for feedback, to differentiate from FG above)

* **REQ-FR-1:** The system **shall** provide textual feedback alongside a grade for each graded assignment. This feedback should be in natural language, as if written by a teacher, highlighting strengths and areas for improvement[[9]](https://ai.google.dev/competition/projects/grademate#:~:text=from%20curated%20options,performance%20and%20improve%20their%20skills).
* **REQ-FR-2:** The tone of the feedback **shall** be constructive and supportive. The AI (Gemini) should be prompted or configured to avoid harsh language. It should aim to “encourage and guide” as per educational best practices (e.g., use phrases like “I suggest you try...” rather than “This is wrong”).
* **REQ-FR-3:** Feedback **shall** include specific details from the student’s submission whenever possible. For example, referencing a part of their answer: “In your second paragraph, you mentioned XYZ, which is a good point, however, you overlooked...”. *Rationale:* Specific feedback is more helpful than generic statements.
* **REQ-FR-4:** The system **shall** allow teachers to edit or add to the feedback given to a student. If a teacher feels the AI feedback missed something or was slightly off, they can append “Teacher’s note: ...” or modify it. The original AI feedback should remain visible or marked so that it’s clear what was AI vs teacher (for trust and analysis).
* **REQ-FR-5:** If the AI cannot generate meaningful feedback (for instance, student answer is blank or irrelevant), the system **shall** produce a default feedback or a gentle message, such as “No answer was provided, so no feedback available” or “Please attempt the assignment to receive feedback.”
* **REQ-FR-6:** The feedback content **shall** be stored in the system associated with the submission, so it can be retrieved later by student, teacher, or parent. (Similar to REQ-FG-7 but focusing on feedback text).
* **REQ-FR-7:** The system **should** tailor feedback length according to the complexity of the assignment. For a short answer, a couple of sentences might suffice; for a long essay, a few paragraphs might be needed. We can instruct AI accordingly or trim if needed. There is no hard requirement on length, but it should not be so long that a student won't read it, nor so short that it’s uninformative.
* **REQ-FR-8:** The system **may** include positive reinforcement in feedback. For example, always find at least one thing the student did well and mention it (this can be part of the AI prompt structure).
* **REQ-FR-9:** The feedback generation process **shall** align with the grading criteria. If an assignment has specific rubric points (e.g., clarity, accuracy, creativity), feedback should touch on those. (This is advanced, but if rubric integration exists, AI should comment per criterion).
* **REQ-FR-10:** Parents viewing the feedback **shall** see it in a possibly simplified summary if needed. If the feedback is very technical or long, we might consider a summary for parents or a tooltip explaining jargon. For now, the same feedback is shown, assuming it’s understandable.
* **REQ-FR-11:** The system **shall not** include any sensitive or personally identifying information in AI prompts or feedback beyond what is necessary. For instance, do not prompt AI with “This answer is by John Doe, age 13”, only the content. Similarly, feedback should not include references like “As a 13-year-old, you should...”, unless relevant. This is to ensure privacy and avoid biases.
* **REQ-FR-12:** If multiple languages are supported in assignments (not stated but imagine an English class vs Spanish class), the system **should** generate feedback in the language of instruction. If a student writes an essay in Spanish and the assignment is marked as Spanish, the feedback should ideally be in Spanish. (This could be achieved by telling AI the language context).
* **REQ-FR-13:** The system **should** store an internal representation of the feedback quality or flags. For example, if the AI used certain key phrases or if the student consistently gets the same feedback, an admin/teacher might want to know. (This is more of an analytical requirement – e.g., track feedback patterns).
* **REQ-FR-14:** In case of low score, the system **shall** either as part of this feature or next (adaptive learning) provide guidance. Possibly: “Review Chapter 2” could appear as part of feedback or right after it. This overlaps with Feature 4.3 but is sometimes combined: the AI might directly say, “I recommend practicing X to improve.”
* **REQ-FR-15:** The feedback mechanism **shall** comply with any educational standards for feedback if provided by client (for example, some schools have policy that feedback must be phrased in a certain positive manner). If so, the AI prompt can be fine-tuned to meet those guidelines.

The above requirements ensure that every grade is accompanied by meaningful comments, fulfilling the platform’s goal of **educative feedback, not just scores**. By implementing these, the system leverages research that shows personalized feedback helps students learn better[[10]](https://slejournal.springeropen.com/articles/10.1186/s40561-024-00310-z#:~:text=etc,natural%20responses%2C%20and%20relevant%20examples).

### 4.3 Feature: Adaptive Learning Recommendations

**Description and Priority:** This feature uses student performance data and AI to suggest what topic or content the student should focus on next. It’s essentially an AI-driven tutor guiding the student’s learning path. Priority is **Medium-High** – it adds significant value (personalization), but after ensuring grading and feedback are solid. This can differentiate the platform by providing an individualized learning experience. It’s closely tied to the AI analysis of performance.

**Stimulus/Response Sequences:** 1. **After Grading Event:** Once a student’s assignment is graded, the system triggers the logic for recommendations. - *Stimulus:* A new grade/feedback is finalized for a student. - *Response:* The system compiles the student’s recent performance record (could be just this assignment or a broader history) and sends a query to the AI or uses internal logic to determine the next topic. For example, “Student scored low in fractions, medium in decimals, high in multiplication. Next recommended topic: division (related to fractions).” - If using the Gemini API for this, it might prompt: “Given the student’s weaknesses in X, suggest a next topic to study.” 2. **Student Views Recommendation:** The next time (or immediately) the student is shown a suggestion: - *Stimulus:* Student views the feedback page or a “Learning Path” page. - *Response:* The UI displays a recommendation, e.g., “Based on your performance, we recommend you review **Decimals** before moving on. [Start lesson]”. The bracket could be a link to materials if available. - If multiple suggestions, maybe a small list (but likely one main suggestion to keep focus). 3. **Teacher/Parent Notification:** - *Stimulus:* Teacher or parent checks progress. - *Response:* They might see that “the system suggests Alice should focus on Decimals next” so that they are aware and can encourage the student on that topic. 4. **Follow-Up Adaptation:** If the student heeds the advice and improves in that area, subsequent recommendations change. - The system should update recommendations as new data comes in. So it’s an ongoing cycle: performance -> recommendation -> new learning -> performance ... - If a student continuously struggles in a certain area, the system might repeat similar recommendations or escalate (like suggesting a remedial module).

**Functional Requirements:** (Label AR for adaptive recommendation)

* **REQ-AR-1:** The system **shall** analyze a student’s performance data to identify strengths and weaknesses. This data may include recent assignment scores, specific topics those assignments covered, and perhaps time taken or attempts (if we track that). Initially, we can simplify: use the latest assignment’s topic or the last few scores.
* **REQ-AR-2:** Based on the analysis, the system **shall** determine a recommended next topic or learning activity for the student. This recommendation could be generated by the AI. For example, sending a summary of performance to Gemini with a prompt like “Suggest next learning topic for a student who scored 60% in fractions, 90% in multiplication” and get a suggestion. Or using rules, e.g., if fraction score < 70% then recommend fractions review.
* **REQ-AR-3:** The recommendation **shall** be specific and actionable. Instead of a vague “study more math,” it should say something like “Practice [Topic X]” or “Review [Concept Y] chapter” or “Try the [Topic Z] quiz next.” If possible, tie into actual content (if our system has links to lessons or external resources).
* **REQ-AR-4:** The system **shall** display the recommendation to the student in the Student Portal in a prominent way after each graded assignment. This might be on the feedback page (e.g., an italic note “Recommended Next: ...”) or on the dashboard (“Your personalized suggestion: ...”).
* **REQ-AR-5:** The system **shall** update the recommendation when new data is available. So, if a student completes another assignment or an extra practice, the suggestion might change. The student’s dashboard should always show the latest recommendation based on current data.
* **REQ-AR-6:** Teachers **shall** be able to view and optionally adjust the recommendations. A teacher might know something the system doesn’t (like upcoming curriculum topics) and might override: e.g., “System suggested Decimals, but I (teacher) think they should focus on Percentages, so I change it.” If teacher changes, system should show that instead and possibly learn from it (learning component, future scope).
* **REQ-AR-7:** The logic for recommendations **should** consider curriculum structure. For example, if topics have a prerequisite structure (like you typically learn fractions before decimals), the system should recommend logically following topics. This might be encoded as knowledge (graph of topics) or indirectly via AI which might know common educational sequences[[11]](https://slejournal.springeropen.com/articles/10.1186/s40561-024-00310-z#:~:text=and%20written%20assignments%20and%20tasks,friendly%20way).
* **REQ-AR-8:** The recommendation feature **shall** be implemented in a way that it does not confuse or overwhelm the student. Only one primary recommendation at a time should be highlighted. If multiple areas need improvement, either pick the most important or sequence them (maybe “Next: topic A, then topic B”).
* **REQ-AR-9:** The system **shall** store the recommendation given (and time) for tracking whether the student followed it (if the system has means to track, like if recommended X and student later does an assignment in X). This can be used to evaluate the adaptation feature’s effect.
* **REQ-AR-10:** If no clear recommendation can be made (say a student is acing everything or the data is insufficient), the system **shall** either suggest a challenge topic or simply encourage further practice without a specific new topic (“Keep up the great work! Perhaps try an advanced problem set in this topic to challenge yourself.”).
* **REQ-AR-11:** The adaptive system **should** make use of the AI’s knowledge base when needed. For example, if a unique pattern occurs, the AI can analyze text of answers to see misconceptions (like common wrong concept used) and base a recommendation on that (e.g., “They consistently used a wrong formula, recommend reviewing formula usage.”). This is advanced but a goal.
* **REQ-AR-12:** The system’s recommendations **shall** align with the content that the platform or school can offer. If the platform doesn’t have content for an exotic suggestion, better to suggest something available. This might mean having a list of known topics or linking with an existing syllabus.
* **REQ-AR-13:** Parents should be informed of recommendations in their view (e.g., “Suggested Next Topic: Decimals”). This helps them help their child.
* **REQ-AR-14:** The computation for recommendation **shall** happen server-side (likely using the AI service or internal logic) to maintain consistency and not expose algorithms.
* **REQ-AR-15:** **Priority/Component ratings:** If following some IEEE suggestions, we could note that this feature’s benefit is high, but penalty of not having it is not critical (the system still works without it). Cost is moderate (AI calls for it, complexity), risk is moderate (if recommendations are poor, might be ignored). So it’s prioritized after core grading.

This feature aims to implement a simplified version of personalized learning, which is an active area of EdTech. The requirements above ensure that the system not only evaluates but also guides, making learning paths more tailored.

### 4.4 Feature: Multi-Portal Access & Role Management

**Description and Priority:** This feature encapsulates the system’s ability to provide different interfaces and permissions based on user roles (Student, Teacher, Parent, Admin). It includes user authentication, authorization, and the functionalities unique to each role. Priority is **High**, since without proper role management the system cannot function correctly (students shouldn’t see others’ data, etc.). It’s foundational to system security and usability.

**Stimulus/Response Sequences:** 1. **User Login:** - *Stimulus:* On accessing the site, a user attempts to log in by providing credentials. - *Response:* The system verifies credentials (checking against MongoDB user records). If valid, it creates a session or JWT and redirects the user to the appropriate portal homepage. The UI and available menu options are dynamically adjusted based on role (e.g., a teacher sees “Create Assignment” option, a student does not). - If invalid, an error message appears (“Invalid username or password”). 2. **Navigation within Portal:** - *Stimulus:* A user navigates to a section (like a teacher clicking “Assignments”). - *Response:* The system checks the user’s session and role to ensure access. If allowed, it serves the requested page data. If not (e.g., a student trying to access an admin page by URL), it denies access (could show 403 Forbidden or redirect to home). 3. **Role-Specific Actions:** - Student submitting assignment (we covered in grading feature). - Teacher creating assignment: - *Stimulus:* Teacher fills out assignment form and saves. - *Response:* System saves assignment to DB, visible to students of that class. - Admin creating a user: - *Stimulus:* Admin enters new user details. - *Response:* System saves new user, possibly sends them an activation link or sets a default password. - Parent viewing child’s report: - *Stimulus:* Parent selects their child (if parent account tied to multiple kids) or goes to progress page. - *Response:* System fetches that student’s records and displays. If parent somehow tries to access a student that’s not their child, system blocks it. 4. **Logout:** - *Stimulus:* User clicks logout. - *Response:* Session is terminated (token invalidated) and user sees login screen.

**Functional Requirements:** (Label UM for User Management / multi-portal)

* **REQ-UM-1:** The system **shall** implement secure authentication for users. Users must log in with a username/email and password (or other auth method). Passwords shall be stored hashed in the database.
* **REQ-UM-2:** The system **shall** support at least the four roles: Student, Teacher, Parent, Administrator. Each user account has one (or more, but typically one) of these roles associated.
* **REQ-UM-3:** Based on the user’s role, the system **shall** present an appropriate user interface (portal). This includes different navigation menus and homepage/dashboard content as described in Section 3.1.
* **REQ-UM-4:** The system **shall** enforce authorization rules for all actions:
* Students can only access their own data (their submissions, their grades).
* Teachers can access data of students in their classes (we assume some mapping of teacher to student group or class).
* Parents can access data of their linked student(s) only.
* Admins can access all data.
* No user should be able to perform actions outside their permission (e.g., a student should not be able to create an assignment or view admin pages).
* **REQ-UM-5:** The system **shall** allow administrators to create, read, update, and delete (CRUD) user accounts and assign roles. This may involve an admin interface to manage users.
* **REQ-UM-6:** The system **should** allow teachers to create classes or groups and enroll students (or assign students to classes), unless class roster is pre-managed by admin. How students and teachers are linked should be definable.
* **REQ-UM-7:** Each portal’s functionality has specific requirements:
* **Student Portal Requirements:**
  + **REQ-UM-7.1:** Students shall be able to view a list of assignments (with status: pending, submitted, graded).
  + **REQ-UM-7.2:** Students shall be able to submit answers to open assignments.
  + **REQ-UM-7.3:** Students shall be able to view their past results, feedback, and recommendations.
  + **REQ-UM-7.4:** Students shall see the leaderboard relevant to them (maybe class-wide).
* **Teacher Portal Requirements:**
  + **REQ-UM-7.5:** Teachers shall be able to create new assignments specifying title, description, due date, and associated class or group of students.
  + **REQ-UM-7.6:** Teachers shall be able to view submissions for each assignment and the AI grades/feedback.
  + **REQ-UM-7.7:** Teachers shall be able to modify grades and feedback as needed (as per grading feature).
  + **REQ-UM-7.8:** Teachers shall be able to view analytics for their classes (average, distribution of scores, etc.).
  + **REQ-UM-7.9:** Teachers (optionally) shall be able to generate content using AI (like quiz questions, as in optional feature).
* **Parent Portal Requirements:**
  + **REQ-UM-7.10:** Parents shall be able to view summary of their child’s performance and latest feedback.
  + **REQ-UM-7.11:** Parents shall be able to see upcoming assignments deadlines (so they can remind their child).
  + **REQ-UM-7.12:** Parents should not see details of other students or any teacher-only data.
  + **REQ-UM-7.13:** If messaging is available, parents could message teacher (not explicitly required now, but some systems have parent-teacher comms).
* **Admin Dashboard Requirements:**
  + **REQ-UM-7.14:** Admins shall be able to manage all users (create/edit/remove accounts, reset passwords).
  + **REQ-UM-7.15:** Admins shall be able to configure system settings, such as the AI integration (e.g., update API keys), feature toggles (like turn on/off leaderboard or content suggestions).
  + **REQ-UM-7.16:** Admins shall be able to view overall usage metrics (number of submissions, AI usage count, etc., possibly for billing or insight).
* **REQ-UM-8:** The system **shall** maintain an audit log of critical actions, especially admin actions (creating users, etc.), and perhaps teacher overrides of grades, for accountability.
* **REQ-UM-9:** The system **should** provide a mechanism for password recovery (since this is user-facing): e.g., “Forgot password” which emails a reset link. This would require integration with email as mentioned. If not in first release, an admin can reset manually.
* **REQ-UM-10:** The system **shall** support concurrency of different roles. For example, a teacher can be grading while students are submitting other assignments, etc., without data conflicts. (This is design-level, but implies our database design allows it).
* **REQ-UM-11:** The interface differences per role must not rely solely on front-end hiding elements. The back-end **shall** enforce role checks on each request to secure data. For instance, even if a student somehow calls an admin API, the server will reject it if their token is not admin.
* **REQ-UM-12:** If a single person has multiple roles (like a teacher who is also a parent of a student in the system), the system should handle that gracefully. Possibly have a way to switch context or have separate accounts. This might be an edge case to consider in design.
* **REQ-UM-13:** The system **shall** have a user-friendly profile section in each portal to allow changing password, updating email, etc., which is a common requirement.
* **REQ-UM-14:** The system **may** integrate with external SSO (Single Sign-On) systems if needed by client (for example, Google login for G Suite schools). Not required now, but design should not preclude adding OAuth providers.
* **REQ-UM-15:** **Performance for multi-user**: The system shall support at least X concurrent users performing actions (the value depends on client’s needs, e.g., 100 concurrent students submitting). This is non-functional, but we ensure the design and role management do not bottleneck.

By fulfilling these requirements, the platform will properly handle different user types, ensuring each gets the right information and functionality. It also ensures security of data by partitioning permissions.

### 4.5 Feature: Leaderboard and Gamification

**Description and Priority:** This feature introduces a competitive gamification element by maintaining a leaderboard of student performance. This is a **Medium Priority** feature. While not core to functionality, it enhances student engagement and is often requested by clients to motivate learners. If time is constrained, it could be slightly lower priority than core learning features, but it’s straightforward to implement and can deliver high engagement, so it’s included.

**Stimulus/Response Sequences:** 1. **Score Update:** Whenever a student receives a grade for an assignment, their cumulative score or points tally is updated. - *Stimulus:* A grading event finalizes (for example, student gets 85 on a quiz). - *Response:* The system converts that grade into some points (maybe directly 85 points or some system like assignment weight) and updates a stored “points total” for that student in the class or overall. 2. **View Leaderboard:** - *Stimulus:* A student clicks on “Leaderboard” in the portal. - *Response:* The system retrieves the ordered list of top students (for the relevant scope, e.g., their class or grade level). It displays names (or aliases if privacy needed) and scores. The student finds their position highlighted. - If the student is not in top N displayed, optionally show “Your rank: 15th out of 30”. - This likely updates in real-time or near real-time after each assignment. 3. **Periodic Reset or Categories (if applicable):** Some systems reset leaderboard each week or have multiple leaderboards (e.g., per class, or per subject). - Implementation may vary; assume one per class for now. 4. **Badges/Achievements:** - *Stimulus:* A student reaches a milestone (e.g., completed 5 assignments on time). - *Response:* System awards a “badge” and maybe notifies the student (“You earned the Consistency Badge!”). - This adds a fun element but is optional; can be part of gamification if planned. 5. **Teacher/Admin Monitoring:** - Teachers might view the leaderboard too, as a quick way to see who’s excelling or struggling (if at bottom). - Admins might see overall usage scoreboard etc.

**Functional Requirements:** (Label LB for leaderboard)

* **REQ-LB-1:** The system **shall** calculate a score or point total for each student to be used in leaderboard rankings. This could be the sum of points from all graded assignments, or an average, or a custom formula. (We need to decide: a common approach is to sum percentage points or give fixed points per activity completion. For simplicity, we could sum raw scores or assign 100 points per assignment proportionally to grade).
* **REQ-LB-2:** The system **shall** maintain a leaderboard listing students in descending order of their score/points. This should be scoped appropriately (likely per class or course). It may also have an overall leaderboard if the client wants cross-class competition, but that could be optional.
* **REQ-LB-3:** The student **shall** be able to view their position on the leaderboard. If they are in the top N visible, they’ll see themselves there; if not, the system should still display their rank below the main list. For example, “... 9. [Name] - 450 pts ... 15. You - 300 pts”.
* **REQ-LB-4:** Privacy consideration: The system **shall** ensure that if displaying a leaderboard, it’s among a cohort that is allowed. Usually within a class, all students know each other’s names and competition is acceptable. If there’s any policy issue, we could allow anonymized display (like only show “Top Student” etc., but likely not needed).
* **REQ-LB-5:** The leaderboard **shall** update whenever new grades are entered. This could be implemented by recalculating after each assignment’s grading or on a schedule. It should be up-to-date when a student opens it.
* **REQ-LB-6:** The system **shall** allow the teacher or admin to disable the leaderboard if desired (some pedagogical approaches might not want competition). So a configuration flag should exist. If disabled, students won’t see the option, or it might show a message “Leaderboards are not enabled.”
* **REQ-LB-7:** The system **should** award simple achievements or badges as part of gamification (optional). If implemented:
* For certain milestones (like perfect score on 3 assignments, improvement streak, consistency in submissions, etc.), a badge icon appears on the student’s profile or leaderboard entry.
* **REQ-LB-7.1:** The system shall display badges next to the student’s name on the leaderboard or profile.
* **REQ-LB-7.2:** The system shall notify the student when they earn a badge.
* **REQ-LB-8:** The leaderboard data **shall** be stored in the database or derived from grades as needed. Possibly store total points for each student for quick access, and update those totals transactionally with each new grade.
* **REQ-LB-9:** If ties in score occur, the system **shall** handle ordering by some secondary criterion (like whoever reached it first or just treat as same rank). This is minor but for display clarity.
* **REQ-LB-10:** The system **should** present the leaderboard in a visually engaging format: e.g., highlight top 3 with special colors (gold, silver, bronze icons).
* **REQ-LB-11:** The system **should not** expose actual scores of others except via this gamified context. That is, a student sees others’ points but not necessarily their individual assignment grades (unless known by other means). Leaderboard essentially reveals relative performance; this is usually acceptable but we note it under privacy.
* **REQ-LB-12:** Teachers **shall** have a view to see the full ordered list including all students, maybe with scores. This helps them quickly gauge distribution.
* **REQ-LB-13:** (If relevant to client’s goals) The system **may** allow resetting or time-bound leaderboards (like monthly leader or per term). If needed:
* Could store historical periods.
* Not mandatory in initial version unless requested.
* **REQ-LB-14:** The system **shall** integrate the leaderboard points in the student’s profile or dashboard summary (like “Total Points: X”).
* **REQ-LB-15:** The presence of a leaderboard **shall** not impact the grading logic; it’s a separate layer. If a grade is adjusted, that naturally flows into points recalculation.

By implementing leaderboard and possibly badges, we expect an increase in student engagement. We will monitor usage and feedback to ensure it’s having a positive effect (e.g., no negative competition issues). Because it’s an additional feature, we made it configurable (enable/disable).

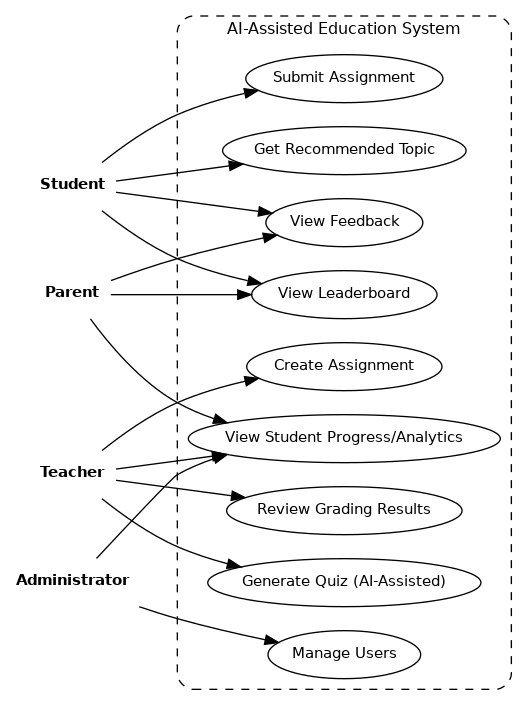
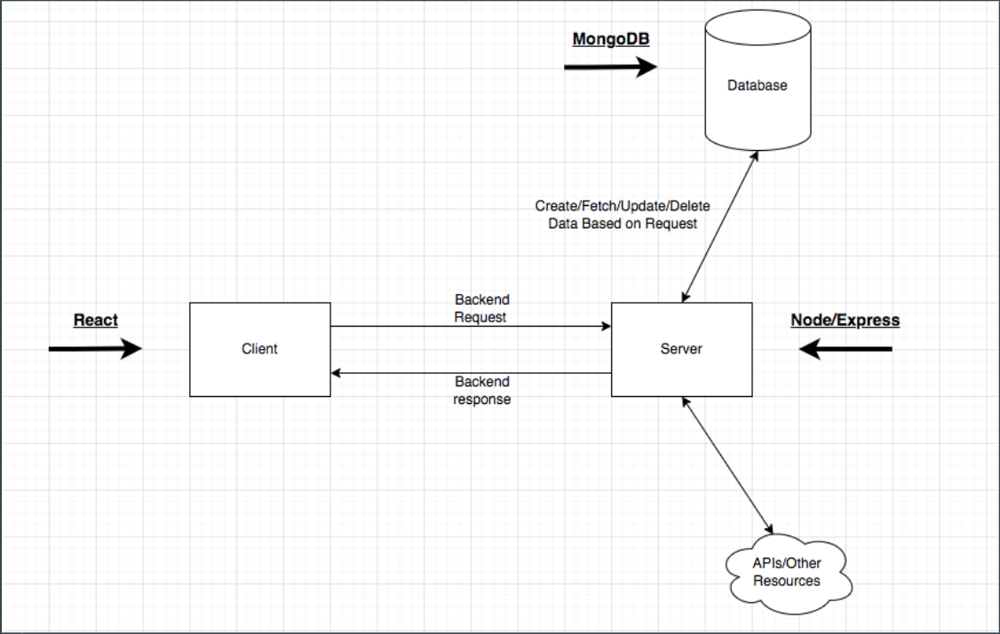
### 4.6 Feature: Content Creation Assistant (Quiz Generator) - *Optional*

**Description and Priority:** This is an optional feature where the AI helps teachers generate quiz questions or other content. Priority is **Low** compared to core features; it’s a “nice-to-have” that can be added if time permits or in a later iteration. It showcases the versatility of the AI beyond grading. The idea: a teacher inputs a topic and gets some suggested questions, saving prep time.

**Stimulus/Response Sequences:** 1. **Teacher Requests Quiz Questions:** - *Stimulus:* Teacher clicks a “Generate Questions” button in the assignment creation form and enters a prompt like “Grade 5 fractions, 5 questions”. - *Response:* The system sends this prompt to the Gemini API (perhaps a different model or the same text generation endpoint) asking for a list of questions (and answers). - AI returns a list of questions with answers. 2. **Teacher Reviews/Edits:** - *Stimulus:* The generated questions are displayed to the teacher. - *Response:* Teacher reviews them, possibly edits any question or discards those they don’t like. They then save them as the new assignment’s questions. 3. **Assignment Distributed:** - *Stimulus:* Students later take this quiz (the normal assignment flow). - *Response:* (Not directly part of this feature beyond content creation). 4. **Alternate Use - Lesson Planning:** - *Stimulus:* Teacher asks AI “Give me an outline for a lesson on photosynthesis” (if we allow such usage in a teacher’s assistant tool section). - *Response:* AI returns an outline or ideas. Teacher can then copy or adapt that.

**Functional Requirements:** (Label TA for Teacher Assistant features)

* **REQ-TA-1:** The system **shall** provide an interface in the Teacher Portal for AI-assisted content generation. This could be a dialog or section where a teacher inputs a prompt describing what they need (e.g., specify topic, number of questions, type of questions).
* **REQ-TA-2:** When the teacher submits a content generation request, the system **shall** call the appropriate AI API endpoint to fulfill it. For questions, it might be the same models/gemini-pro:generateContent but with a prompt like “Generate X quiz questions on Y topic with answers.” Alternatively, if Gemini has specialized models or parameters, those should be used to get structured output.
* **REQ-TA-3:** The system **shall** display the AI-generated content to the teacher for approval. It should not automatically send it to students without teacher review. Teacher must have the opportunity to edit or regenerate if not satisfied.
* **REQ-TA-4:** The quality of generated content **shall** meet basic criteria:
* Relevant to the prompt topic.
* Appropriate difficulty for the indicated grade level (if provided).
* The system cannot fully ensure quality, so that’s why teacher oversight is required.
* **REQ-TA-5:** The system **should** allow multiple iterations: e.g., a “Regenerate” button if the teacher wants a new set of suggestions, or an option to refine the prompt.
* **REQ-TA-6:** If multiple content types are supported (questions, hints, lesson plans, explanations), the system should categorize or label what the AI output is. For now, the focus is quiz questions, possibly including the expected answer or solution for each (so teacher has them for grading or key).
* **REQ-TA-7:** The AI prompt from teacher might be freeform text. The system **shall** include context if needed in the actual call, e.g., if teacher’s prompt doesn’t specify number of questions clearly, maybe default to 5. We might also instruct AI to format the output clearly (like numbered list of Q&A).
* **REQ-TA-8:** This feature being optional, the system **shall** allow it to be disabled (some organizations might not want AI generating content for whatever reason). If disabled, the UI won’t show the option.
* **REQ-TA-9:** The content generation **shall** respect content guidelines:
* It should not produce any inappropriate questions (the AI likely won’t, but e.g., avoid sensitive topics unless asked).
* If the prompt is ambiguous, the system should try to clarify (maybe not, since it's just passing to AI).
* **REQ-TA-10:** The system **shall** log these AI content generation requests similarly to grading ones for future auditing. If a question was generated by AI, maybe mark it (so later if there’s an issue, one knows its origin).
* **REQ-TA-11:** The AI-generated questions or content **shall** be editable just like any manually entered content before finalizing the assignment. No difference after insertion – the teacher can modify text, remove or add more questions.
* **REQ-TA-12:** (Future scalability) Possibly allow AI to generate different levels of questions if students need more practice. Out of scope now, but the design might consider it.
* **REQ-TA-13:** The system **should** caution that these questions are AI-generated and need review. Perhaps a disclaimer in the UI like “These questions were generated by AI. Please verify their accuracy and appropriateness.”
* **REQ-TA-14:** Performance: Generating content might be a heavier request (if asking for long output). It should still be relatively quick (a few seconds). We should handle timeouts similarly. If it times out or gives an error, show error to teacher and allow retry.
* **REQ-TA-15:** If the AI supports it, the system might allow specifying format (like multiple-choice vs short answer questions). The interface can have options, which the system then includes in the prompt.

* **Use Case Diagram:** Provided as Figure 2 in section 2.2  
  , showing actors and their interactions with system functions.
* **System Architecture Diagram:** Provided as Figure 1 in section 2.1  
  , illustrating the MERN architecture and external integrations.
* **UML Class Diagram:** *TBD* – We could include a high-level class diagram of the data model. For instance, classes (or collections) like User, Assignment, Submission, with relationships. (Not drawn here, but to be provided possibly in design stage.)
* **Sequence Diagram for AI Grading:** *TBD* – Could illustrate the sequence of messages for a student submission going through AI and back with result.
* **State Diagram:** If needed, maybe for assignment states (Not started -> Submitted -> Graded -> Reviewed).
* **ER Diagram:** Entity-Relationship diagram of the MongoDB data model could be included, albeit Mongo is schemaless but we can show conceptual schema.
* **Flowchart of Recommendation Logic:** If complex logic is used, a flowchart might be drawn.

**Appendix C: To Be Determined (TBD) List**

Collect a list of items that are marked as TBD (to be determined) which require resolution:

1. **Points Calculation for Leaderboard (TBD):** Decide formula for calculating points from grades. Will it be sum of percentages, or weighted by assignment, etc.? (Current assumption: sum of scores normalized to some scale).
2. **Recommendation Engine Implementation (TBD):** Decide whether recommendations will be purely AI-driven or rule-based or hybrid. Possibly start with simple rules + AI confirmation.
3. **Content for Adaptive Recommendation (TBD):** What resource or content is provided when a topic is recommended? Do we have built-in lessons or just say review textbook? This may need definition by client (if they have digital content or we just point to generic advice).
4. **Policy on AI Feedback Release (TBD):** Will we allow the AI’s feedback to go directly to students, or require teacher approval? Currently we assume direct release for speed, but if any school policy requires teacher to okay it first, we might need a mechanism.
5. **Language Support (TBD):** Will the initial deployment require multi-language (e.g., Arabic interface for Cairo if in Egypt)? The stakeholder should clarify to plan translation.
6. **Threshold for Overriding (TBD):** Should AI only handle certain types of questions (essays, short answers) and not others (like multiple-choice)? Probably teacher will input correct answers for auto-grade for objective questions anyway. Define scope of AI grading usage.
7. **Deployment specifics (TBD):** Will the system be delivered as SaaS by the company to multiple clients or installed on client’s own servers? This affects how we configure multi-tenancy or separate instances. For now assume single deployment per client with their data.
8. **User Load (TBD):** Confirm expected user counts (how many students/teachers) to properly size infrastructure and test scenarios.
9. **Integration APIs (TBD):** Check if need to integrate with any existing SIS (Student Info System) for roster or grade export (like output final grades to school system). Not in scope now, but if needed in phase 2, note it.
10. **AI API Costs/Budget (TBD):** Determine how many API calls per month is acceptable cost-wise, and whether to restrict usage (like maybe limit students to requesting re-evaluation or extra feedback to avoid runaway cost). Possibly not an immediate requirement but a business decision to set usage caps if needed.