**TABLE OF CONTENTS**

[1. INTRODUCTION](#_Toc197036776)

[1.1 Purpose](#_Toc197036777)

[1.2 Scope](#_Toc197036778)

[1.3 Definitions, Acronyms, and Abbreviations](#_Toc197036779)

[1.4 References](#_Toc197036780)

[2. SYSTEM ARCHITECTURAL DESIGN](#_Toc197036781)

[2.1 System Description](#_Toc197036782)

[2.2 System Architecture](#_Toc197036783)

[2.3 Design Constraints](#_Toc197036784)

[3. COMPONENTS DESCRIPTION](#_Toc197036785)

[3.1 Introduction](#_Toc197036786)

[3.2 Decomposition Description](#_Toc197036787)

[3.2.1 Component Details](#_Toc197036788)

[4. EXTERNAL INTERFACES](#_Toc197036789)

[1. User Interfaces](#_Toc197036790)

[1.1 Teacher Interface](#_Toc197036791)

[1.2 Student Interface](#_Toc197036792)

[2. External System Interfaces](#_Toc197036793)

[2.1 Learning Management System (LMS) Integration](#_Toc197036794)

[2.2 Third-Party Grammar and Plagiarism APIs (Optional)](#_Toc197036795)

[2.3 Email Notification Service](#_Toc197036796)

[5. DETAILED DESIGN](#_Toc197036797)

[5.1 Introduction](#_Toc197036798)

[5.2 Component Descriptions](#_Toc197036799)

[5.2.1 Input Manager](#_Toc197036800)

[5.2.1.1 General Description](#_Toc197036801)

[5.2.1.2 Detailed Behavior](#_Toc197036802)

[5.2.2 Grading Engine](#_Toc197036803)

[5.2.2.1 General Description](#_Toc197036804)

[5.2.2.2 Detailed Behavior](#_Toc197036805)

[5.2.3 Feedback Generator](#_Toc197036806)

[5.2.3.1 General Description](#_Toc197036807)

[5.2.3.2 Detailed Behavior](#_Toc197036808)

[5.2.4 Database Manager](#_Toc197036809)

[5.2.4.1 General Description](#_Toc197036810)

[5.2.4.2 Detailed Behavior](#_Toc197036811)

[5.2.5 User Interface (UI)](#_Toc197036812)

[5.2.5.1 General Description](#_Toc197036813)

[5.2.5.2 Detailed Behavior](#_Toc197036814)

[5.2.6 Admin Module](#_Toc197036815)

[5.2.6.1 General Description](#_Toc197036816)

[5.2.6.2 Detailed Behavior](#_Toc197036817)

[6. ANNEXES](#_Toc197036818)

[6.1 Traceability Table](#_Toc197036819)

# 1. INTRODUCTION

## 1.1 Purpose

The purpose of this document is to provide a comprehensive architectural design for the Automated Essay Grading Tool. This system aims to streamline the assessment process by enabling teachers to create essay questions, collect student responses, and receive suggested grades based on predefined grading criteria. The document outlines the structure, components, external interfaces, and detailed designs of the system to guide developers, designers, and stakeholders in the system's implementation and evaluation.

## 1.2 Scope

This document addresses the architectural design of a web-based tool designed for educational institutions. The system allows teachers to manage essay-based assessments and facilitates automatic grading based on configurable criteria such as phrase detection, word count thresholds, numerical calculations, and custom combinations thereof. The intended audience includes software architects, developers, testers, and project stakeholders involved in the development lifecycle of the system.

## 1.3 Definitions, Acronyms, and Abbreviations

| Term/Acronym | Definition |
| --- | --- |
| UML | Unified Modeling Language |
| IEEE | Institute of Electrical and Electronics Engineers |
| SRS | System Requirements Specification |
| PRD | Product Requirements Document |
| CRUD | Create, Read, Update, Delete |
| NLP | Natural Language Processing |

## 1.4 References

* Jurafsky, D., & Martin, J. H. (2020). *Speech and Language Processing* (3rd ed.).
* spaCy: <https://spacy.io>
* NLTK (Natural Language Toolkit): <https://www.nltk.org>
* Django Framework: <https://www.djangoproject.com>
* OWASP Security Guidelines: <https://owasp.org>

# 2. SYSTEM ARCHITECTURAL DESIGN

## 2.1 System Description

The Automated Essay Grading Tool is a web application that allows teachers to create and configure essay-based questions, while enabling students to submit answers. The system uses natural language processing techniques to evaluate the submissions against a set of grading criteria and provides teachers with grading suggestions. Teachers retain full control to review and adjust the final grade.

## 2.2 System Architecture

**Frontend**:

* React or simple HTML/CSS/JS interface for teachers and students
* Responsive UI for ease of access on various devices

**Backend**:

* Django/Node.js for REST API services
* Essay storage and retrieval using PostgreSQL or MongoDB
* Essay analysis and scoring using Python-based NLP tools (e.g., spaCy, NLTK)

**Grading Engine**:

* Rule-based and keyword-based criteria evaluator
* Modular system to allow combining multiple criteria (e.g., "contains phrase X" AND "longer than 150 words")

**Data Flow**:

1. Teacher creates question + grading criteria
2. Student submits essay
3. Backend processes and analyzes submission
4. Grading engine evaluates essay
5. Suggested score and breakdown shown to the teacher

## 2.3 Design Constraints

* Limited to English-language essays in initial version
* Accuracy of scoring depends on the quality of input and defined criteria
* Does not fully replace human judgment—intended to assist, not automate entirely
* Requires stable internet connection and secure server deployment
* Data privacy and secure storage must comply with regulations (e.g., GDPR, local policies)

# 3. COMPONENTS DESCRIPTION

## 3.1 Introduction

This section outlines the decomposition strategy for the Automated Essay Grading Tool (AEGT), also referred to as the Automated Grading Tool (AGT). The system is a web application designed to assist educators by allowing them to create essay prompts, enabling students to submit responses, and providing automated grading suggestions based on configurable criteria. The final evaluation authority rests with the teacher. The decomposition strategy follows a modular, multi-tiered architectural approach, separating the system into distinct layers and functional components to enhance maintainability, reusability, testability, and potential integration with external systems like Learning Management Systems (LMS). The architecture comprises a Frontend, a Backend providing REST API services, a distinct Grading Engine, and a Database layer. This structure facilitates independent development and scaling of different system parts.

## 3.2 Decomposition Description

The AEGT system is primarily partitioned into the following major architectural layers and logical components:

1. **Frontend Layer:** Provides the user interfaces for all roles (Teacher, Student, Admin). Built using technologies like React or standard HTML/CSS/JS. Responsible for presenting information and capturing user input.
2. **Backend Layer:** Implemented using frameworks like Django or Node.js, it exposes RESTful APIs. It houses the core application logic, orchestrates workflows, and interacts with the data layer and external services.
3. **Grading Engine:** A distinct backend component, potentially developed using Python NLP tools (spaCy, NLTK). It focuses specifically on analyzing essay content against defined criteria. Designed to be modular.
4. **Data Persistence Layer:** Manages data storage and retrieval, using PostgreSQL or MongoDB.

Within the Backend Layer and interacting across layers, the system is further decomposed into the following key functional components, largely derived from the "Detailed Design" document:

1. **UI Module:** (Frontend) Handles user interaction for students and instructors.
2. **Input Manager:** (Backend) Manages submission uploads, validation, and optional checks.
3. **Grading Engine:** (Backend) Performs the automated evaluation of submissions.
4. **Feedback Generator:** (Backend) Creates feedback reports for students.
5. **Admin Module:** (Backend/Frontend) Allows administrators to manage system configurations like assignments and users.
6. **Database Manager:** (Backend - Data Persistence Layer) Handles all database interactions.

These components interact via defined interfaces (primarily REST APIs between Frontend and Backend ) and internal function calls. UML diagrams (like component or sequence diagrams) would further illustrate these interactions and dependencies.

### 3.2.1 Component Details

Each major functional component identified above is described below:

1. **UI Module**
   * **Purpose:** Provide interfaces for each user role (Student, Teacher, Admin).
   * **Function:** Displays assignments, allows essay submission (text/file upload), presents feedback and grades. Enables teachers to create/manage questions, define criteria, review/override grades, and provide feedback. Presents dashboards and reports. Facilitates admin tasks like user and assignment management. Ensures responsive design and accessibility (WCAG 2.1).
   * **Interfaces:** Interacts with the Backend Layer via REST APIs to fetch data and trigger actions. Presents information visually to end-users.
2. **Input Manager**
   * **Purpose:** Handle incoming student submissions reliably and securely.
   * **Function:** Receives uploaded assignment files. Validates file format and size against requirements. Checks submissions against deadlines. Optionally performs plagiarism checks via external APIs. Rejects invalid submissions. Corresponds to requirements RQ-01 and RQ-02.
   * **Interfaces:** Receives data from the UI Module (submission uploads). Interacts with the Database Manager to store submission metadata. May call External Plagiarism APIs. Passes validated submissions to the Grading Engine or stores them for processing.
3. **Grading Engine**
   * **Purpose:** Evaluate submitted essays automatically based on defined criteria.
   * **Function:** Applies grading rules, which can include test cases (for code), rubrics, keyword analysis, word count checks, phrase matching, and other NLP-based evaluations. Calculates scores based on these evaluations. Designed to be modular to combine multiple criteria. Its output serves as a suggestion for the teacher. Corresponds to requirement RQ-03. Limited to English initially.
   * **Interfaces:** Receives submission data (essays, criteria) likely from the Essay Workflow logic in the backend. Uses NLP libraries (spaCy, NLTK). Passes results (suggested scores, analysis breakdown) to the Grading Review/Feedback components.
4. **Feedback Generator**
   * **Purpose:** Create detailed, personalized feedback reports for students.
   * **Function:** Analyzes grading results. Composes feedback highlighting strengths and suggesting improvements. Generates the final report presented to the student via the UI. Corresponds to requirement RQ-04.
   * **Interfaces:** Receives grading results from the Grading Engine. Interacts with the Database Manager to store/retrieve feedback data. Provides formatted feedback data to the UI Module.
5. **Admin Module**
   * **Purpose:** Allow admins to manage core system settings and entities.
   * **Function:** Provides functionality (via specific UI views and backend APIs) to create/manage assignments, define grading rubrics/criteria, set deadlines, and manage user roles and permissions. Corresponds to requirement RQ-07.
   * **Interfaces:** Interacts with the UI Module for administrative tasks. Interacts heavily with the Database Manager to modify system configurations, user data, and assignment details.
6. **Database Manager**
   * **Purpose:** Handle all data persistence operations securely and efficiently.
   * **Function:** Stores and retrieves student submissions, grading criteria, grading results/scores, feedback, user information (including roles), assignment details, and audit logs. Ensures data integrity and compliance with regulations like GDPR. Provides CRUD-like methods. Corresponds to requirement RQ-05.
   * **Interfaces:** Provides data access services to most other backend components (Input Manager, Grading Engine, Feedback Generator, Admin Module, general backend logic). Interacts directly with the chosen database system (e.g., PostgreSQL, MongoDB).

# 4. EXTERNAL INTERFACES

## 1. User Interfaces

The system provides web-based user interfaces tailored to each user role, ensuring accessibility, responsiveness, and ease of use.

### 1.1 Teacher Interface

- Purpose: To allow teachers to manage essay tasks, define grading rules, review automated scores, and provide feedback.  
- Features:  
 • Login/registration page  
 • Dashboard with question creation and assignment tools  
 • Grading criteria builder (phrase match, word count, etc.)  
 • Interface for reviewing, overriding, and commenting on grades  
 • Report generation for student/group performance  
- Design Constraints:  
 • Must follow WCAG 2.1 accessibility standards  
 • Responsive layout for desktop and tablet use

### 1.2 Student Interface

- Purpose: To enable students to submit essays and receive feedback.  
- Features:  
 • Login page with role-based access  
 • Dashboard showing assigned questions  
 • Essay submission form (supports text and file upload)  
 • Feedback page with automated grading breakdown  
- Design Constraints:  
 • Mobile-friendly interface  
 • Must provide status indicators for submitted/draft essays

## 2. External System Interfaces

### 2.1 Learning Management System (LMS) Integration

- Purpose: To synchronize users, assignments, and scores with external LMS platforms like Moodle, Canvas, or Google Classroom.  
- Interface Type: RESTful API (or LTI if required by LMS)  
- Functions:  
 • Push/pull assignment metadata  
 • Sync grades automatically upon completion  
 • Authenticate users via LMS SSO (Single Sign-On)  
- Constraints:  
 • Must comply with LMS data schemas and security standards  
 • Configurable endpoints for different platforms

### 2.2 Third-Party Grammar and Plagiarism APIs (Optional)

- Purpose: To enhance feedback with grammar suggestions or plagiarism checks.  
- Examples: Grammarly, LanguageTool, Turnitin  
- Integration Method: REST API calls with document text payload  
- Constraints:  
 • Requires API key and quota management  
 • Response times must not block essay submission UI

### 2.3 Email Notification Service

- Purpose: To notify users of submission status, grading completion, or feedback availability.  
- Technology: SMTP or cloud-based services (e.g., SendGrid, AWS SES)  
- Triggers:  
 • New assignment posted  
 • Essay successfully submitted  
 • Grades published

# 5. DETAILED DESIGN

## 5.1 Introduction

This section presents the internal design details of the Automated Grading Tool (AGT). Each system component is described in terms of its general purpose and detailed behavior.  
Behavior is expressed using textual algorithms and visual notations (UML diagrams) where appropriate.  
Both passive (methods, functions, or procedures) and active behavior (state machines, activity diagrams) are documented.

## 5.2 Component Descriptions

### 5.2.1 Input Manager

#### 5.2.1.1 General Description

The **Input Manager** is responsible for handling student submissions. It ensures that uploaded files meet format, size, and deadline requirements. It also optionally checks for plagiarism.

#### 5.2.1.2 Detailed Behavior

**Passive Behavior (Methods)**:

* uploadSubmission(studentId, assignmentId, file): Receives the uploaded assignment file.
* validateFileFormat(file, allowedFormats): Validates that the file meets format requirements.
* checkDeadline(assignmentId): Verifies that submission is before the deadline.
* plagiarismCheck(file): (Optional) Checks for plagiarism in the submitted file.

**Active Behavior (Activity Diagram)**:

Ảnh có chứa văn bản, biểu đồ, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.z

### 5.2.2 Grading Engine

#### 5.2.2.1 General Description

The **Grading Engine** evaluates student submissions automatically. It applies test cases, grading templates, or rubrics depending on assignment type (e.g., code, MCQs, written reports).

#### 5.2.2.2 Detailed Behavior

**Passive Behavior (Methods)**:

* runTestCases(submission, testCases): Executes the student code against predefined test cases.
* evaluateRubric(submission, rubric): Applies scoring rules based on an instructor-defined rubric.
* assignGrade(submission): Calculates the final score and maps it to a grade.

### 5.2.3 Feedback Generator

#### 5.2.3.1 General Description

The **Feedback Generator** creates a detailed, personalized feedback report for each student.  
The report includes both positive highlights and suggestions for improvement.

#### 5.2.3.2 Detailed Behavior

**Passive Behavior (Methods)**:

* generateFeedback(submission, gradingResults): Composes the feedback report based on grading.
* highlightStrengths(submission): Points out well-done aspects.
* suggestImprovements(submission): Recommends specific areas for improvement.

**Active Behavior (Activity Diagram)**:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Nội dung do AI tạo ra có thể không chính xác.

### 5.2.4 Database Manager

#### 5.2.4.1 General Description

The **Database Manager** handles all system data storage, including student submissions, grading results, user information, and audit logs.

#### 5.2.4.2 Detailed Behavior

**Passive Behavior (Methods)**:

* storeSubmission(submission): Saves student submissions securely.
* saveGrade(studentId, assignmentId, grade): Records the final grade for a submission.
* fetchSubmissions(assignmentId): Retrieves all submissions related to a specific assignment.

### 5.2.5 User Interface (UI)

#### 5.2.5.1 General Description

The **User Interface** provides interaction channels for students and instructors. Students can submit assignments and review feedback; instructors can create assignments and monitor grading.

#### 5.2.5.2 Detailed Behavior

**Passive Behavior (Methods)**:

* displayAssignments(studentId): Lists available assignments.
* uploadAssignment(studentId, file): Allows students to upload submissions.
* viewGrades(studentId): Displays students' grades and feedback.

### 5.2.6 Admin Module

#### 5.2.6.1 General Description

The **Admin Module** allows administrative users to manage assignments, rubrics, deadlines, and user roles.

#### 5.2.6.2 Detailed Behavior

**Passive Behavior (Methods)**:

* createAssignment(details): Enables creation of new assignments.
* defineRubric(assignmentId, rubricDetails): Defines grading criteria for assignments.
* manageUserRoles(userId, role): Changes permissions and access rights for users.

# 6. ANNEXES

## 6.1 Traceability Table

| **Requirement ID** | **Functional Description** | **Design Component** | **Test Case ID** |
| --- | --- | --- | --- |
| RQ-01 | Upload student assignments | Input Manager | TC-01 |
| RQ-02 | Validate submission format and deadline | Input Manager | TC-02 |
| RQ-03 | Automatic grading of submissions | Grading Engine | TC-03 |
| RQ-04 | Generate detailed feedback | Feedback Generator | TC-04 |
| RQ-05 | Store submissions and grading results | Database Manager | TC-05 |
| RQ-06 | Student and instructor interactions | UI Module | TC-06 |
| RQ-07 | Administrative configuration of assignments | Admin Module | TC-07 |