StudyMate – AI Powered Advanced Quiz Creator

Coding Standards Document

Submitted by

Syed Muhammad Abubakar Zaidi (1600-2021)
Hanzala Siddiqe (2577-2021)

Supervisor(s)

Asst. Prof. Mr. Osama Ahmed Khan
Senior Lecturer Mr. Waqas Pasha

In partial fulfilment of the requirements for the degree of
Bachelor of Science in Software Engineering
2025

Faculty of Engineering Sciences and Technology

Hamdard Institute of Engineering and Technology

Hamdard University, Main Campus, Karachi, Pakistan

1. Introduction

This document defines the coding standards and conventions applied throughout the **StudyMate** project to ensure consistency, readability, and maintainability of the codebase. All contributors are expected to follow these guidelines during development.

2. Purpose

The purpose of these coding standards is to:

- Promote clear and consistent code.
- Reduce the risk of errors.
- Simplify onboarding of new developers.
- Support efficient code reviews and maintenance.

3. Programming Languages and Frameworks

The project uses:

- **Python 3.11 (Django Framework)**
- **HTML/CSS/JavaScript (Frontend Templates)**
- **AJAX for asynchronous interactions**

4. Naming Conventions

- **Python Modules & Packages:** Use lowercase with underscores (e.g., `quiz_generation.py`).
- **Classes:** Use PascalCase (e.g., `ExtractedTextProcessor`).
- **Functions and Methods:** Use lowercase with underscores (e.g., `generate_quiz()`).
- **Variables:** Use descriptive lowercase names with underscores (e.g., `user_email`).
- **Constants:** Use uppercase with underscores (e.g., `MAX_RETRIES`).
- **Templates and Static Files:** Use lowercase with hyphens (e.g., `user-profile.html`).

5. Code Formatting

- Indentation: **4 spaces per level** (no tabs).
- Line length: **Max 120 characters**.
- Use spaces around operators and after commas.
- Always include a newline at the end of files.
- Avoid trailing whitespace.

6. Comments and Documentation

- Each module and class must include a **docstring** describing its purpose.
- Functions must include a brief docstring explaining inputs and outputs.
- Inline comments should clarify **why** something is done, not **what**.
- Use `# TODO:` for planned improvements.

7. Error Handling

- Always use `try/except` blocks when interacting with external APIs or file operations.
- Log exceptions with appropriate messages.
- Never suppress exceptions silently.

8. Security Practices

- Never hard-code sensitive credentials or API keys in code.
- Use Django's built-in security features for user authentication and CSRF protection.
- Sanitize and validate all user inputs.

9. Version Control

- All code changes must be committed with **clear, descriptive messages**.
- Follow a feature-branch workflow.
- Merge only after code reviews and successful testing.

10. Code Review

- All code must be reviewed by at least one other team member.
- Reviews should focus on:
    - Code correctness and functionality.
    - Adherence to these coding standards.
    - Performance considerations.