SecuroGuard

E-Commerce Fake Review Detection Application

Coding Standards Document

<u>Submitted by</u>

Faarah Khan (2578-2021)
Hafsa Nisar (1988-2021)

<u>Supervisor</u>

Mr. Afzal Hussain

<u>Co-Supervisor</u>

Mr. Maaz Ahmed

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 used throughout the **SecuroGuard** project to ensure clean, consistent, and maintainable code. Adopting these standards helps every team member write better quality code and contributes to efficient testing, debugging, and future upgrades.

2. Purpose

The purpose of this coding standards document is to:

- Promote consistent and readable code.
- Reduce errors and bugs in functionality.
- Support better teamwork and smoother collaboration.
- Improve overall development and testing efficiency.

3. Programming Languages and Frameworks

The following technologies are used in SecuroGuard:

- **PHP** – Backend APIs and server-side scripting
- **Python (3.11)** – Machine Learning model and prediction logic
- **HTML, CSS, JavaScript** – Frontend interface
- **Chart.js** – For visualization and report graphs
- **MySQL** – Relational database management
- **AJAX** – For smooth frontend-backend interactions
- **Google Authentication** – For secure user login

4. Naming Conventions

- **PHP Files:** snake_case.php (e.g., db_connection.php)
- **Python Scripts:** lowercase_with_underscores.py (e.g., predict_model.py)
- **Classes:** PascalCase (e.g., ReviewAnalyzer)
- **Functions/Methods:** lower_case_with_underscores() (e.g., get_reviews())
- **Variables:** descriptive_name (e.g., review_text, user_email)
- **Constants:** ALL_CAPS_WITH_UNDERSCORES (e.g., MAX_REVIEW_COUNT)
- **Frontend Templates:** lower-hyphen.html (e.g., review-result.html)
- **CSS Classes:** lower-hyphen (e.g., card-container, btn-analyze)

5. Code Formatting

- Indentation: **4 spaces** per level (tabs not allowed)
- Line length: Max **120 characters**
- Use spaces around operators and after commas
- End each file with a newline
- No trailing whitespace
- Group related code into logical blocks with line breaks

6. Comments and Documentation

- Every class and function must have a **docstring** or comment explaining its purpose
- Inline comments explain **why** something is done, not what
- Use // TODO: or # TODO: for future improvements
- Write comments in **English**, simple and to the point

7. Error Handling

- Always use try-except in Python and try-catch in PHP for external calls or file operations
- Log exceptions with useful messages
- Never leave errors unhandled or suppressed silently
- Display user-friendly messages on frontend; log actual errors in backend

8. Security Practices

- Never hard-code passwords, tokens, or database credentials
- Use **Google Authentication** for secure login
- Sanitize and validate **all** user input (especially URLs and form data)
- Implement **CSRF protection** and session timeouts
- Escape data before displaying to prevent **XSS attacks**

9. Version Control

- Use **Git** for version control
- Commit messages should be **clear and meaningful** (e.g., fix: improved review parser)
- Follow a **branching strategy** for each new feature or bug fix
- Only merge into main after testing and review

10. Code Review

- Every code change must be reviewed by another team member
- Reviewer checks:
    - Code quality and correctness
    - Adherence to coding standards
    - Security and performance risks
- Code must be tested before final approval