GlucoPredict – AI Powered Diabetes Predictor

Coding Standard Document

Submitted by

M. Hassan bin Sabih (2203-2021)
M. Moiez Siddiqui (2270-2021)

Supervisor(s)

Dr. Khurram Iqbal

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, conventions, and best practices applied throughout the GlucoPredict project to ensure consistency, readability, security, and maintainability of the codebase. All team members are expected to adhere to these guidelines during the entire development process.

2. Purpose

The purpose of these coding standards is to:

a. Promote clear, consistent, and professional coding practices.
b. Reduce the risk of programming errors and technical debt.
c. Simplify onboarding for new developers.
d. Facilitate efficient code reviews, debugging, and long-term maintenance.

3. Programming Languages and Frameworks

The GlucoPredict project uses:

a. Python 3.11 (Backend APIs and ML Model)
b. TensorFlow & Scikit-learn (Machine Learning)
c. Android (Java/Kotlin) (Mobile Application)
d. Firebase Authentication and Firestore
e. AWS EC2 and DynamoDB
f. Jupyter Notebook (ML Development)

4. Naming Conventions

a. Python Modules & Packages: lowercase with underscores (e.g., data_preprocessing.py)
b. Classes: PascalCase (e.g., RiskPredictorModel)
c. Functions and Methods: lowercase with underscores (e.g., predict_risk_score())
d. Variables: descriptive lowercase names with underscores (e.g., user_blood_sugar)
e. Constants: UPPERCASE with underscores (e.g., MAX_THRESHOLD)
f. Android Layout Files: lowercase with underscores (e.g., activity_main.xml)
g. Android Activities & Classes: PascalCase (e.g., MainActivity)
h. Firebase Collections: lowercase with hyphens (e.g., user-records)

5. Code Formatting

a. Indentation: 4 spaces per indentation level (no tabs)
b. Line Length: Maximum 120 characters
c. Spacing: Use spaces around operators and after commas
d. Newlines: Include a newline at the end of each file
e. Avoid trailing whitespace

f. Consistently structure Android XML layouts with proper indentation and line spacing

6. Comments and Documentation

a. Each Python module and class must include a docstring summarizing its purpose.
b. Functions should have a docstring detailing input parameters, processing, and return values.
c. Inline comments should clarify why a specific operation is performed.
d. Use # TODO: tags for planned improvements or pending issues.
e. Android Java/Kotlin code should use single-line (//) and block comments (/**/) appropriately.

7. Error Handling

a. Use try/except blocks for all interactions with external services, APIs, and file operations.
b. Log exceptions with meaningful messages and context.
c. Never suppress or ignore exceptions silently.
d. In Android, validate user inputs before operations and handle exceptions using try-catch blocks.

8. Security Practices

a. Never hard-code sensitive credentials, API keys, or tokens in source files.
b. Use environment variables or Firebase/AWS security features for sensitive data.
c. Sanitize and validate all user inputs at both client-side (Android) and server- side (APIs).
d. Leverage Firebase Authentication and HTTPS for secure communication.
e. Apply role-based access control where needed.

9. Version Control

a. Use Git with clear, descriptive commit messages.
b. Follow a feature-branch workflow:

Example: feature/user-authentication, bugfix/api-timeout

c) Merge branches only after code reviews and successful testing.
d) Maintain an up-to-date.gitignore file for ignoring system files and secrets.

10. Code Review

a. All code must be reviewed by at least one other team member before merging.
b. Code reviews should focus on:
   i. Code correctness, functionality, and security.
   ii. Adherence to these coding standards.
   iii. Performance optimization opportunities.