Recruit Right: Precision Hiring with AI Insight

Coding Standards Document

Submitted by,

Muhammad Naeemuddin 1955-2021
Muhammad Abdullah 2206-2021
Muhammad Raza 2207-2021

Supervisor,

Dr. Umer Farooq

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 for the **Recruit Right** project to ensure consistent, readable, and maintainable code across all modules.

2. General Guidelines

- Use **UTF-8 encoding** for all source files.
- Write code and comments in **English**.
- Keep line length **≤ 100 characters**.
- Indentation:
  - Python: 4 spaces, no tabs.
  - JavaScript and CSS: 2 spaces, no tabs.
- Use meaningful, descriptive names for variables, functions, and classes.
- Avoid commented-out or dead code before merging.

3. Backend (Python) Standards

3.1 Structure & Formatting

- Follow PEP 8 style guide.
- Imports grouped: standard library → third-party → local, separated by blank lines.
- Use **snake_case** for variables and functions.
- Use **PascalCase** for classes.
- Constants in **ALL_CAPS_WITH_UNDERSCORES**.
- Use type hints for all function parameters and return types.

3.2 Docstrings & Comments

- Use Google style docstrings for all public functions and classes.
- Provide concise summaries and parameter/return explanations.
- Use inline comments only when logic is complex.

3.3 Error Handling

- Define custom exceptions for domain errors.
- Return meaningful HTTP status codes and JSON error messages in API responses.

3.4 Testing

- Use `pytest` framework.
- Target minimum 80% coverage.
- Test typical cases and edge cases.

4. Frontend Standards (Plain HTML/CSS/JavaScript)

4.1 File Organization

- Organize files into folders such as:
- `/static/`
- `  /css/`
- `    styles.css`
- `  /js/`
- `    main.js`
- `    auth.js`
- `  /images/`
- `/templates/`
- `  index.html`
- `  login.html`
- `  dashboard.html`
- Keep CSS and JS in separate files, not inline in HTML.

4.2 HTML

- Use semantic HTML5 elements (`<header>`, `<nav>`, `<main>`, `<footer>`, `<section>`, `<article>`, etc.).
- Use descriptive `id` and `class` names (kebab-case preferred).
- All forms must have associated `<label>` elements for accessibility.
- Use `aria-` attributes where needed to improve screen reader experience.

4.3 CSS

- Use plain CSS or SCSS if preprocessor is configured (please confirm).
- Follow consistent naming conventions for classes (e.g., BEM or simple kebab-case).
- Avoid !important; prefer specificity and cascade.
- Organize styles logically by page or component in separate CSS files if large.
- Use CSS variables for colors and fonts to maintain consistency.

4.4 JavaScript

- Use ES6+ syntax but ensure compatibility with target browsers.
- Use `const` and `let` instead of `var`.
- Encapsulate code in modules or IIFEs to avoid polluting global namespace.
- Follow **camelCase** naming for variables and functions.
- Separate DOM manipulation, event handling, and business logic into functions.
- Use meaningful event delegation to optimize listeners.
- Avoid inline JavaScript in HTML attributes.

4.5 Testing

- Use manual testing and browser dev tools for debugging.
- If automated frontend testing is needed, consider tools like Selenium or Cypress (not mandatory).

5. Version Control & Commits

- Use Git for version control.
- Branch naming:
  - Features: `feat/<description>`
  - Fixes: `fix/<description>`
  - Chores: `chore/<description>`
- Commit message format:
- `<type>(<scope>): <short summary>`

  Where `<type>` is one of `feat`, `fix`, `docs`, `style`, `refactor`, `test`, `chore`.

- Example:
- `feat(auth): add password reset functionality`
- Write clear commit messages that explain the reason and impact.

---

6. Security Best Practices

- Use environment variables for sensitive data; do NOT hardcode secrets.
- Validate all user inputs both on frontend and backend.
- Sanitize inputs to prevent XSS and SQL injection (if applicable).
- Use HTTPS on all pages.
- Use JWT tokens with expiry for session management.
- Passwords hashed with bcrypt with 12 rounds.
- Role-based access control enforced server-side.

7. Code Review & Pull Requests

- All pull requests require at least one review before merging.
- PRs must pass all tests and linters.
- PRs should be focused and small where possible.
- Include screenshots or logs if UI or behavior changed.
- Reference related issues or tasks in PR description.

8. Linters & Formatters

- Backend: Use `flake8` and optionally `black` for Python formatting and linting.
- Frontend: Use `eslint` configured for ES6 JavaScript.
- Configure lint checks to run automatically on commit (via `husky` or CI) and in CI pipelines.
- Fix all lint errors before pushing code.

9. Documentation

- Keep API documentation up-to-date, preferably with OpenAPI (Swagger).
- Update README with setup and deployment instructions.
- Comment complex code sections clearly.

10. Continuous Integration

- Use GitHub Actions or similar to run linting, tests, and builds on every PR and push.
- Failing checks block merging.
- Automate deployment steps on `main` branch after successful checks.