CODING STANDARDS

BMW IT HUB

COS 301 | University of Pretoria

FIRE-FIGHTER ACCESS MANAGEMENT PLATFORM

Table of Contents

1. Introduction

This Coding Standards Document outlines the conventions and practices adopted by Team Apex to ensure our codebase is consistent, maintainable, readable, and secure. These standards promote team cohesion, streamline onboarding, and enhance long-term project quality and sustainability. The document is aligned with academic guidance (Kung, 2014) and tailored to our FireFighter Access Management Platform's stack (Angular, Spring Boot, PostgreSQL, etc.).

2. Coding Standards Overview

We follow professional coding standards across all layers of the system:

2.1 Goals

- **Clarity**: Code should be self-explanatory.
- **Consistency**: Uniform style across the team.
- **Maintainability**: Easy to modify and extend.
- **Reliability**: Error-resistant and secure.
- **Efficiency**: Optimised for performance where necessary.

3. Language-Specific Guidelines

3.1 Java (Spring Boot)

- **Naming Conventions**:
    - Classes: PascalCase (e.g., UserService)
    - Methods/Variables: camelCase (e.g., getUserToken)
    - Constants: UPPER_CASE_WITH_UNDERSCORES
- **File Structure**:

src/main/java/com/apex/firefighter

├── controller

├── service

├── model

├── repository

└── security

- **Practices**:
    - Use @Transactional for DB operations.
    - Inject dependencies using constructor injection.
    - Follow RESTful controller principles (status codes, clear endpoints).
    - Log actions using SLF4J (logger.info(...)), not System.out.

3.2 TypeScript (Angular + Ionic)

- **Naming Conventions**:
  - Components: PascalCase (e.g., AccessRequestComponent)
  - Variables/Functions: camelCase
  - Files: kebab-case.component.ts
- **Practices**:
  - Strong typing everywhere.
  - Use Angular Services for business logic.
  - Use reactive forms and avoid manipulating the DOM directly.
  - Group files per component (HTML, SCSS, TS).

4. Style Guides and Linters

- **Java**: Google Java Style Guide
  - Tooling: Checkstyle, SpotBugs
- **TypeScript**: Angular Style Guide
  - Tooling: ESLint, Prettier
- **Common**:
  - Tabs/spaces: 2-space indentation (Angular), 4-space (Java)
  - Line length: 100 characters max
  - Bracing style: K&R style (allman discouraged)
  - Comments: Use Javadoc-style (/** */) for Java, // or /** */ for TS

5. Code Review and Quality Control

5.1 Peer Reviews

- All PRs require review by at least one other developer.
- Checklist includes:
    - Does the code follow style guides?
    - Are error-handling and logging appropriate?
    - Is the code readable and well-documented?

5.2 Static Code Analysis

- CI pipeline checks formatting and known security issues.
- Java: SpotBugs, PMD
- Angular: ESLint, SonarCloud (via GitHub Actions)

6. File Structure and Naming in Repository

Our **monorepo** structure:

/firefighter-platform

```
├── FF-API      # Spring Boot backend
├── FF-Angular        # Angular front-end
├── integration-tests    # Integration testing between frontend and backend
└── README.md
```

7. Git Workflow

- **Branching Strategy**:
  - main: Production-ready code
  - dev: Active development integration branch
  - feature/xyz: Feature branches
  - bugfix/xyz, hotfix/xyz: For urgent fixes
- **Semantic Commits**:
  - feat: for new features
  - fix: for bug fixes
  - test: for adding or fixing tests
  - docs: for documentation updates

8. Security and Defensive Programming

- Input validation on both frontend and backend.
- Avoid hard-coded secrets (use .env or secrets manager).
- Handle exceptions gracefully; never expose stack traces to the frontend.
- Follow OWASP top 10 guidelines.

9. Testing Standards

- **Unit Testing**:
  - Backend: JUnit & Mockito
  - Frontend: Jasmine & Karma
- **Integration Testing**:
  - REST API flows and DB tests using Spring Boot Test
- **E2E Testing**:
  - Cypress for frontend
- **Test Naming Convention**:
  - shouldReturn403WhenUserIsUnauthorised()

All tests must run on CI before merging to main.

10. Tools and Automation

- **CI/CD**: GitHub Actions
- **Testing & Coverage**:
  - JUnit, Cypress, SonarCloud
  - Coverage thresholds: 80%+ unit test coverage for new features
- **Automated Testing**:
  - Jenkins is used to automatically test if all our testing standards passed
- **Formatting**:
  - Auto-format on commit using Prettier (Angular) and Google Format (Java)

11. Coding Standards Review Process

The coding standards are:

- Reviewed bi-weekly during retrospectives.
- Updated when tools or team practices evolve.
- Enforced via pre-commit hooks, CI checks, and code review feedback.

12. References

- Kung, D. C. (2014). *Object-Oriented Software Engineering: An Agile Unified Methodology*.
- Google Java Style Guide
- Angular Style Guide
- OWASP Secure Coding Practices