OWCA Advanced FitchFork Coding Standards

1. Overview

This document defines the coding conventions, styles, and repository structure for the Advanced-FitchFork project. Consistent adherence ensures clarity, maintainability, flexibility, reliability, and efficiency across the entire codebase.

2. Repository Structure

- **Advanced-FitchFork/**
  - **backend/**
    - **api/** Main API server (Actix)
    - **code_runner/** Code execution microservice
    - **common/** Shared Rust libraries
    - **db/** Database models and logic
    - **marker/** Automated marking logic
    - **migration/** Database migrations
    - **seeder/** Database seeding scripts
    - **util/** Utility crates
  - **frontend/**
    - **public/** Static assets
    - **src/**
      - **components/** Reusable UI components
      - **context/** React context providers
      - **constants/** Application constants
      - **config/** API and app configuration
      - **hooks/** Custom React hooks
      - **layouts/** Layout components
      - **pages/** Page-level components
      - **routes/** Route definitions and guards
      - **services/** API service functions
      - **types/** TypeScript type definitions
      - **utils/** Generic utility functions
    - **index.html**
    - Configuration files (ESLint, Prettier, tsconfig, Vite, Tailwind)
  - **docs/** Architecture diagrams, design guides, PDFs
  - **.github/** Actions workflows, issue & PR templates
  - **README.md** Project overview, setup, and contribution guide
  - Additional top-level configuration (Dockerfiles, CI meta, etc.)

3. Frontend (TypeScript + React)

1. **Linting & Formatting**
    - **ESLint** (`frontend/eslint.config.js`)
        - Plugins: React, Accessibility (jsx-a11y), Import order, Prettier integration
    - **Prettier** (`frontend/.prettierrc`)
        - Rules: semicolons required, single quotes, print width = 100, tab width = 2, trailing commas = all

2. **Scripts**
    - `npm run lint` - report lint issues
    - `npm run lint:fix` - auto-fix lint violations
    - `npm run format` - apply Prettier formatting

3. **TypeScript Practices**
    - Strict mode enabled in `tsconfig.json`
    - Types in `src/types/`, organised by feature
    - Explicit interfaces/types for props, state, API responses

4. **React Practices**
    - Functional components + hooks exclusively
    - Context API for global state
    - Presentational vs. container component separation

5. **File & Naming Conventions**
    - Files/folders: kebab-case (e.g. `user-profile.tsx`)
    - Components: PascalCase (e.g. `UserProfile`)
    - Variables/functions: camelCase

4. Backend (Rust)

1. **Workspace Layout**
    - Top-level `backend/` as a Cargo workspace
    - Each subdirectory (`api`, `marker`, `db`, etc.) is its own crate

2. **Module Organization**
    - Root modules via `mod.rs`; submodules grouped by feature
    - Public APIs documented with `///` comments

3. **Error Handling & Types**
    - Custom `MarkerError` for domain errors

    - Use `Result<T, E>` and the `thiserror` crate for ergonomics

4. **Formatting & Linting**
    - **rustfmt** (default) enforced in CI
    - **Clippy** for lints, treat warnings as errors in CI

5. **Dependencies & Versions**
    - Pin versions in each `Cargo.toml`; workspace-wide `.cargo/config.toml` for overrides
    - Avoid wildcard versions; follow semver

5. Configuration Files

- **Frontend**
    - `eslint.config.js` - lint rules and plugin list
    - `.prettierrc` - formatting rules
    - `tsconfig.json` - compiler strictness and module resolution
    - `vite.config.ts` - build and dev server configuration
    - `tailwind.config.ts` - design system tokens

- **Backend**
    - `Cargo.toml` / `Cargo.lock` - dependency management
    - `rustfmt.toml` - any custom formatting settings
    - `clippy.toml` - lint overrides (if needed)
    - `Makefile.toml` or `Justfile` - build, test, lint commands
    - `Dockerfile` - production container build

6. Best Practices & Guidelines

- **Consistency**: follow naming, module layout, and style across all code.
- **Documentation**: every public function/type must have a doc comment.
- **Testing**: unit tests alongside code (`#[cfg(test)]`), integration tests in `tests/`.
- **Code Reviews**: no merges without peer review. Focus on correctness, style, and readability.
- **CI/CD**: enforce formatting, linting, and test suite on every commit to `main`.
- **Security**: sandbox student code; TLS for all communications; do not commit secrets.

7. Maintenance & Evolution

This document is a living artefact. As technologies, frameworks, or project needs evolve, these standards will be updated to keep the codebase healthy and maintainable.