Gym Manager Coding Standards

Rome was Built in a day

1 Introduction

This document outlines the coding standards used in the Gym Manager project. These standards ensure uniformity, clarity, reliability, and maintainability across the codebase. They cover conventions for TypeScript, file structure, formatting tools, and commit enforcement via automated tooling.

2 Language and Style Conventions

2.1 Language

- Language: TypeScript
- Frameworks: Express (API), Drizzle ORM (DB), Expo (Mobile Frontend)
- Use `interface` over `type` aliases
- Prefer named exports for consistency

2.2 Naming Conventions

- **Files**: kebab-case (e.g. `user-controller.ts`)
- **Variables & Functions**: camelCase (e.g. `getUserData`)
- **Classes & Interfaces**: PascalCase (e.g. `UserService`)
- **Constants**: UPPER_SNAKE_CASE (e.g. `MAX_RETRIES`)

2.3 Code Style Rules

- Enforce semicolons
- Limit line length to 100 characters
- No unused variables or imports
- Avoid using `any`; prefer typing or generics
- Explicit return types on exported functions
- Avoid floating promises (`await` or handle)

3 Tooling and Configuration

3.1 Prettier

Used for consistent formatting.

```
{
  "semi": true,
  "singleQuote": true,
  "tabWidth": 2,
  "trailingComma": "es5"
}
```

3.2 ESLint

Configured with Flat Config (ESLint 9+). File: `eslint.config.js`

```
import js from '@eslint/js';
import tseslint from '@typescript-eslint/eslint-plugin';
import parser from '@typescript-eslint/parser';
import prettier from 'eslint-plugin-prettier';

export default [
  js.configs.recommended,
  tseslint.configs.recommended,
  {
    ignores: [
      '**/*.test.ts',
      '**/*.test.tsx',
      '**/*.spec.ts',
      '**/*.spec.tsx',
      'tests/**',
      'node_modules/**',
    ],
  },
  {
    files: ['**/*.ts', '**/*.tsx'],
    languageOptions: {
      parser,
      parserOptions: {
        ecmaVersion: 2020,
        sourceType: 'module',
        project: './tsconfig.json',
      },
      globals: {
        console: true,
        process: true,
      },
    },
    plugins: {
      '@typescript-eslint': tseslint,
```

```
      prettier,
    },
    rules: {
      'prettier/prettier': 'error',
      '@typescript-eslint/no-unused-vars': ['warn'],
      '@typescript-eslint/explicit-function-return-type': 'warn',
      '@typescript-eslint/no-explicit-any': 'warn',
      '@typescript-eslint/consistent-type-definitions': ['error',
          'interface'],
      '@typescript-eslint/no-floating-promises': 'error',
    },
  },
];
```

3.3 Husky and lint-staged

Used to enforce code quality on commit.

```
npx husky install
npx husky add .husky/pre-commit "npx␣lint-staged"
```

3.3.1 lint-staged config in `package.json`

```
"lint-staged": {
  "*.{ts,tsx}": ["eslint --fix", "prettier --write"]
}
```

4 Maintaining Standards

- Running `npm run lint` and `npm run format` before pushing
- Using meaningful commit messages
- Enforcing code reviews for merge requests
- Auto linting on commits
- CICD Pipeline automatically runs linting on push to main