Coding Standards

Helix

August 12, 2024

Contents

1 Introduction

This document outlines coding standards for our project, aiming to ensure consistency, maintainability, reliability, scalability, efficiency, and collaboration.

2 Git Strategy

- Use Git Flow with main branches (`main`, `develop`) and supporting branches (feature, release, hotfix, bugfix).
- Commit messages: `type(scope): description`
- Use Pull Requests for merging, with at least one review.
- Tag releases using semantic versioning.

3 File Structure

Monorepo approach with `client`, `server`, `config`, `docs`, and `scripts` directories.

4 Naming Conventions

- General: Use descriptive names, camelCase for variables/functions, PascalCase for classes/types, UPPER_CASE for constants.
- Angular: Suffix components with `Component`, services with `Service`, etc.
- DynamoDB: Use snake_case for tables and columns.
- Git: Use kebab-case for branch names.

5 Interface Schema

- Use PascalCase for interface names, camelCase for properties.
- Use appropriate data types and descriptive names.

6 Function Conventions

- Use camelCase, descriptive names, and follow single responsibility principle.
- Keep functions short (< 100 lines) and document complex ones.
- Use async/await for asynchronous operations.

7 Code Layout

- Use consistent indentation (4 spaces for JS/TS).
- Limit line length to 80 characters.
- Use braces for all control structures.
- Organize imports and group related code.

8 Indentation and Spaces

Aspect	JS/TS (Angular)	CSS/SCSS	JSON/YAML
Indentation	4 spaces	2 spaces	2 spaces
Line Length	80 characters	80 characters	80 characters
Braces	Same line opening, new line closing	Same as Angular	N/A

9 Testing and Debugging

- Use `describe/it` blocks for test structure.
- Use Cypress for component and end-to-end tests.

10 Error Handling

- Use try-catch blocks and throw specific exceptions.
- Implement logging and graceful degradation.

11 Comments

- Use single-line comments for brief notes, block comments for longer explanations.

12 Formatting

- Follow language-specific indentation and spacing rules.
- Use consistent naming conventions and document functions.
- For CSS/SCSS, use BEM methodology and responsive design principles.

13 Linting Rules

- We use Prettier as our code formatter to maintain consistent code styles across our projects.
- This configuration ensures:
  - A maximum line width of 80 characters
  - Indentation using 4 spaces (not tabs)
  - Semicolons at the end of statements
  - Single quotes for strings
  - Spaces inside object literal braces
  - Parentheses around arrow function parameters
  - Unix-style line endings (LF)
- By adhering to these formatting rules, we maintain clean, consistent, and readable code across our codebase. All team members should integrate this Prettier configuration into their development environment to ensure uniformity in code style.

14 Code Review Process

- All code changes must go through a pull request and be reviewed by at least one other team member.
- Reviewers should focus on code quality, adherence to standards, potential bugs, and areas for improvement.
- Use constructive feedback and discuss alternative approaches when necessary.

15 Performance and Optimization

- Consider performance implications when writing code, especially in critical paths.
- Avoid premature optimization; focus on writing clean, maintainable code first.

16 Security Best Practices

- Validate and sanitize all user inputs to prevent injection attacks.
- Use parameterized queries or prepared statements to avoid SQL injection.
- Store sensitive data securely and follow encryption best practices.
- Keep dependencies up-to-date and monitor for security vulnerabilities.

17 Documentation

- Maintain a README file for the project, outlining its purpose, setup instructions, and key information.
- Use inline comments to explain complex code segments or non-obvious functionality.
- Keep documentation up-to-date as the codebase evolves.

18 Continuous Integration and Deployment (CI/CD)

- Implement a CI/CD pipeline to automate build, test, and deployment processes.
- Run automated tests on each commit to catch regressions early.
- Use a staging environment for testing before deploying to production.
- Include code examples demonstrating good and bad practices for each programming language used in the project.
- Highlight common pitfalls and how to avoid them.

19 Regular Review and Updates

- Encourage team members to propose improvements and discuss them during review sessions.
- Update the document as needed and communicate changes to the entire team.