Coding Standards and Repository Structure

Table of Contents

1. Purpose

This document defines the coding standards and repository structure for the project. It aims to improve code quality, readability, maintainability, and team collaboration.

2. File Structure Overview

The repository uses a GitFlow branching model. See "GitHub Contributions" section for branching policies.

Typical file structure:

```
📂 BRAD
├── 📂 api/                # backend and data storage
├── 📂 backend/            # backend and data storage
├── 📂 bot/                # Used to analyze reports
├── 📂 frontend/           # UI
├── 📂 docs/               # Documentation
├── 📄 README.md           # Project overview
├── 📄 docker-compose.yml  # Docker setup
├── 📄 .gitignore          # Ignore unnecessary files
├── 📄 .dockerignore       # Ignore unnecessary files
```

3. Coding Conventions

3.1 Naming Conventions

Element	Convention	Example
Classes	PascalCase	`UserProfile`
Variables	camelCase	`userName`
Constants	UPPER_SNAKE_CASE	`MAX_USERS`

Element	Convention	Example
Functions	camelCase	`calculateScore()`
Files	kebab-case	`user-profile.cpp`

3.2 Formatting

- Use 4 spaces for indentation.
- Limit lines to 100 characters.
- Braces on new lines for classes and functions.
- One space between control statements and ():

Example:

```
if (condition) {
    doSomething();
}
```

3.3 In-code Commenting

Use `//` for short comments and `/* ... */` for block comments.

Every function and complex logic block must have a comment explaining purpose or reasoning.

Example:

```
// Calculate total score based on weights and penalties
int totalScore = baseScore + bonus - penalty;
```

4. Responsibilities

Clear responsibilities are assigned to ensure coding standards are followed:

- Developers: Apply these standards while writing code.
- Reviewers: Ensure code adheres to standards before approving.
- Team Leads: Train team members and ensure adherence.

5. Justification for Standards

Using coding standards:

- Enhances code readability and team collaboration
- Ensures consistent formatting and structure
- Makes integration and testing easier
- Improves tool compatibility (e.g., Doxygen, linters)
- Simplifies onboarding of new developers

6. Guidelines for Practice

To successfully implement these standards:

- Document work early to avoid overhead.
- Involve all developers in standard design and updates.
- Review and update standards as the project evolves.
- Use tools (e.g., linters, formatters, CI) to enforce rules automatically.

GitHub Contributions

Branching Strategy

We will be making use of a **GitFlow-based branching strategy**, with a layout that looks as follows:

```
-> main
    -> dev
        -> ui
        -> docs
        -> api
        -> docker-container
            -> derived branch A
        -> ai-bot
            -> derived branch B
```

Follow the Chain (VERY IMPORTANT)

Please:

```
> ❌ DO NOT COMMIT TO MAIN/DEV
> ❌ DO NOT MERGE TO MAIN/DEV
> ❌ DO NOT BRANCH FROM MAIN/DEV
```

Explaining main and dev:

- **main** is our production branch that should always be in a working condition. Direct changes that do not follow our branching strategy can introduce issues on the version that other people might see.

- **dev** is the branch where all other branch additions are combined into an almost working state. In this branch all *minor* bugs and integrations are fixed before sending to main.

Our **dedicated DevOps team-member** will be managing these two branches and taking all new changes from the respective **feature branches** through dev to main.

Creating Branches

Each person is allowed to create personal/feature branches, as long as:

- They branch off one of the **feature branches**.
- The branches are created from an **Issue**. See the Issues section below if you are unsure how to do this.
- They **use issues to create the branch**. This will help the DevOps team manage everything.

Deleting Branches

When you’re done with an issue and it has been merged, you can select the option to delete the branch along with closing the issue.

Otherwise, if you wish to unassign yourself from the issue, you may delete the branch and update the project board **without closing the issue** to do so.

Merging and Pull Requests

When to Merge

Before making a pull request, you should **merge the feature branch** you are branching from into your current branch.

Use the command:

```
git pull origin <branchname>
```