WriteMe Coding Standards Document

Introduction

This document outlines the coding conventions and styles that our team follows to ensure our code is uniform, clear, flexible, reliable, and efficient. Adhering to these standards will help maintain consistency across our codebase, making it easier to read, maintain, and collaborate on projects.

General Conventions

Code Style

R1: **Indentation:** We use tabs for indentation with a length of two spaces.

R2: **Line Length:** Limit lines to 80 characters. This improves readability and makes it easier to have multiple files open side by side.

R3: **Semicolons:** Always use semicolons to terminate statements.

R4: **Quotes:** Use single quotes for strings, except to avoid escaping.

R5: **Trailing Commas:** Use trailing commas where valid in multi-line constructs (e.g., arrays, objects).

Naming Conventions

R1: **Variables:** Use *camelCase* for variable names.

R2: **Constants:** Use *UPPER_CASE* for constant values

R3: **Functions:** Use *camelCase* for function names.

R4: **Classes:** Use *PascalCase* for class names.

R5: **Files:** Use *kebab-case* for filenames.

Comments

R1: **Single Line Comments:** Use ' // ' for single-line comments.

R2: **Multi-line Comments:** Use ' /* ... */ ' for multi-line comments.

R3: **Documentation Comments:** Use ' [//]: #comment ' for comments made in documentation.

```
//Comments Example
/**
 * Function to add two numbers.
 * @param {number} a - The first number.
 * @param {number} b - The second number.
 * @return {number} The sum of the two numbers.
 */
function add(a, b) {
  return a + b;
}
```

Error Handling

For error handling, *try-catch-finally* statements are used to ensure that errors are properly caught and handled, and any necessary cleanup is performed.

```
try {
  // Code that might throw an error
} catch (error) {
  // Handle the error
} finally {
  // Code that will run regardless of an error occurring
```

```
}
```

Linting & Formatting

We use **Prettier** for code formatting to enforce a consistent style across our codebase. The configuration for Prettier is as follows:

```
{
  "semi": true,
  "singleQuote": true,
  "trailingComma": "all",
  "printWidth": 80,
  "tabWidth": 2,
  "useTabs": true
}
```

Prettier is integrated into each of our IDEs and configured to save on format, making it easier to maintain consistency automatically.

File Structure

The repository follows a structured and logical file organization to make navigation intuitive and to segregate different types of files effectively. Below is the standard file structure for our projects:

```
.
├── apps
│   ├── writeme #Nextjs app
│   │   ├── app
│   │   │   └── api # additional api routes
│   │   ├── public
│   │   └── specs
│   ├── writeme-docs # documentation website
│   │   ├── docs
│   │   ├── guides
│   │   ├── src
│   │   │   ├── components
│   │   │   ├── css
```

```
│   │   │   └── pages
│   │   └── static
│   │       └── img
│   └── writeme-e2e # end-to-end tests for writeme app
│       └── src
├── wmc  # components library
│   └── src
└── wmc-utils # utilities for components library
    └── src
```

Branching Strategy

Our branching strategy is designed to ensure a smooth workflow and maintain the stability of the main codebase.

Branching

Main Branch: The *main* branch contains the stable, production-ready code. **Development Branch:** The *dev* branch is the primary branch for ongoing development. **Feature Branches:** Developers should create feature branches from dev for new features, bug fixes, or any significant work. The naming convention for feature branches is *feat/feature-name*.

Process

R1: **Branch Off:** Always branch off *dev* for new work.

R2: **Commits:** Commit changes regularly to your feature branch. Small, frequent commits make it easier to track changes and roll back if needed.

R3: **Pull Requests:** Once the feature is complete, create a pull request (PR) from your feature branch to *dev*. At least one reviewer is required to review and approve the PR before it can be merged.

R4: **Merge:** After approval, merge the feature branch into *dev*. Resolve any conflicts that arise during merging.

Commits & Pull Requests

- **Commit Messages:** Write clear, concise commit messages that describe the changes made. Use the following format: feat(*type*): *detailed message*

- **Type:** Indicates the type of change (e.g. fix, docs, style, refactor, test, chore, component, page). **Example:**

  ```
  feat(fix): fixed an error in the SignUp component that allowed
  duplicate email registration.
  ```

- **Pull Requests:**

  R1: **Description:** Provide a detailed description of the changes made in the pull request.

  R2: **Review:** Ensure that at least one team member reviews and approves the PR.

  R3: **Testing:** Make sure all tests pass before merging the PR.