Coding Standards Document

Purpose

This document aims to define the coding standards to be followed by developers working on the Student Alumni Mentorship Portal project. Adhering to these standards ensures consistency, readability, and maintainability across the codebase.

Language and Frameworks

- Language: JavaScript (for both front end and back end)
- Front End Framework: React.js
- Back End Framework: Node.js with Express.js
- Database: MongoDB

General Guidelines

Naming Conventions

- Variables & Functions: Use camelCase for variables and function names.
- Constants: Use SCREAMING_SNAKE_CASE for constants.
- File Names: Use descriptive names that indicate the file's purpose.
- Components: Use PascalCase for React component names.
- Boolean Variables: Prefix with "is" or "has" for clarity.

Indentation and Formatting

- Use 2 spaces for indentation.
- Avoid trailing spaces or tabs.
- Use descriptive and meaningful variable and function names.
- Maintain consistent code formatting across the project.

Comments and Documentation

- Write descriptive comments to explain complex logic or algorithms.
- Document functions, APIs, and modules using JSDoc-style comments for clarity.
- Ensure that code changes are accompanied by appropriate updates in documentation.

Error Handling

- Implement consistent error handling across the application.

- Use try-catch blocks for handling synchronous errors and promise rejections.
- Log errors with relevant information for debugging purposes.

Security

- Sanitize inputs and validate data to prevent security vulnerabilities (use libraries like Joi for input validation).
- Avoid hardcoding sensitive information like API keys or credentials directly in the code.
- Implement security best practices for authentication and authorization.

Front End Guidelines

React and Redux

- Follow functional component patterns where possible.
- Use Redux for state management, maintaining a single source of truth for application state.
- Divide components logically, promoting reusability and separation of concerns.

Component Structure

- Organize components into folders based on functionality.
- Use container components to manage state and presentational components for UI rendering.

Styling

- Utilize Material-UI for consistent styling across components.
- Follow a modular approach for styling using CSS-in-JS or CSS modules.

API Calls

- Use Axios for making asynchronous HTTP requests to the backend.
- Implement error handling for API requests.

Back End Guidelines

Express.js and MongoDB

- Structure routes logically and modularly.
- Utilize middleware for authentication, error handling, etc.
- Follow RESTful API conventions for route naming and HTTP methods.

Database Interactions

- Use Mongoose for interacting with the MongoDB database.
- Implement schemas and models for data consistency and validation.

Authentication

- Integrate Auth0 for user authentication and authorization.
- Implement secure token handling and validation.

Version Control (GitHub)

Commit Format

Write clear, descriptive commit messages that explain the changes made.

To maintain a clean and structured commit history, follow this format for commit messages:

<type>(<scope>): <subject>

<body>

<footer>

- Type:
  - feat: New feature
  - fix: Bug fixes
  - docs: Documentation changes
  - style: Code style changes (no impact on code functionality)
  - refactor: Code refactoring or restructuring
  - test: Adding or modifying tests
  - chore: Regular maintenance or tooling changes
- Scope:
  - Indicate the scope of changes (e.g., component name, module).
- Subject:
  - Briefly describe the changes in present tense (50 characters or less).
- Body (optional):
  - Provide additional context, details, or reasoning behind the changes.
  - Use bullet points for clear readability.
- Footer (optional):
  - Include any relevant issue or task IDs linked to the commit.

Example Commit Message

feat(ProfileComponent): Add user profile editing functionality

- Implemented user profile form with validation

- Connected UI with backend API for updating user profiles

Resolves: #123

Branch Naming Conventions

Use meaningful branch names that reflect the purpose of changes.

Feature Branches:

- Format: `feature/<short-description>`
- **Example:** `feature/user-authentication`

Bug Fix Branches:

- Format: `bugfix/<issue-number>-<short-description>`
- **Example:** `bugfix/123-fix-login-issue`

Release Branches:

- Format: `release/<version>`
- **Example:** `release/v1.0.0`

Testing

Unit Testing

- Write unit tests using frameworks like Jest for both front end and back end code.
- Aim for good code coverage to ensure the reliability of the codebase.
- Format: `<component-name>.test.js`
- Example: `userProfile.test.js`

Integration Testing

- Conduct integration tests to ensure seamless interaction between front end and back end components.
- Format: `<component-name>.integration.test.js`
- Example: `userProfile.integration.test.js`

Additional Commit Details

- Atomic Commits:
    - Make commits atomic and focused on specific tasks or changes.
    - Avoid mixing unrelated changes in a single commit.
- Rebase Before Merge:
    - Rebase feature branches onto the main branch before merging to maintain a clean and linear commit history.

Continuous Integration/Continuous Deployment (CI/CD)

CI/CD Pipeline

- Implement automated CI/CD pipelines to ensure smooth integration and deployment of code changes.

Deployment

- Define deployment strategies for staging and production environments.

Conclusion

Adherence to these coding standards will promote a consistent, maintainable, and high-quality codebase for the Student Alumni Mentorship Portal project.