CODING STANDARDS

FULL STACK DEVELOPMENT

Professional Networking Platform

Created By: Krishna Panditrao Chavan
Intern id: IP-3865

Approved By: Harshada Topale
Project Manager:- Harshada Topale

Cloud Counselage

GIFT A CAREER FOUNDATION

Contents

1 PURPOSE **2**

2 SCOPE **2**

3 FILE STRUCTURE **2**
- 3.1 Standard File Conventions . . . 2
- 3.2 Markdown Files . . . 2
- 3.3 Common Conventions . . . 2

4 FORMATTING CONVENTIONS **3**
- 4.1 Indentation . . . 3
- 4.2 Using Capitalization to Aid Readability . . . 3
- 4.3 Formatting Single Statements . . . 3
- 4.4 Formatting Declarations . . . 3
- 4.5 Formatting Multi-line Statements . . . 3

5 NAMING CONVENTIONS **3**

6 SCOPING CONVENTIONS **3**
- 6.1 Lexical/Static Scoping . . . 3
- 6.2 Dynamic Scoping . . . 3

7 COMPILE ERRORS & WARNINGS **4**
- 7.1 Errors . . . 4
- 7.2 Warnings . . . 4

8 ENFORCING CODING STANDARD **4**

9 APPENDICES **4**
- 9.1 Appendix A – Example React Component . . . 4

1 PURPOSE

The Coding Standards are guidelines for software Developers to create uniform coding habits that ease the reading, checking, and maintaining of code. The intent of these standards is to define a natural style and consistency, yet leave to the authors, the freedom to practice their craft without unnecessary burden.

These coding standards shall enable the following:

- Improve Code Quality
- Increase Efficiency
- Facilitate Collaboration
- Ensure Compatibility
- Reduce Maintenance Costs

2 SCOPE

This document describes general software coding standards for code written in JavaScript and React.js specific to the Professional Networking Platform project and shall be implemented while developing the code for the said project.

3 FILE STRUCTURE

3.1 Standard File Conventions

- Use kebab-case for file names (e.g., user-profile.js, job-search.jsx)
- Group related files in directories (e.g., components/, pages/, utils/)
- Use index.js files for cleaner imports

3.2 Markdown Files

- Use .md extension for markdown files
- Include a README.md in each major directory
- Use proper markdown syntax for headings, lists, and code blocks

3.3 Common Conventions

- Keep files focused on a single responsibility
- Limit file size to 400 lines where possible
- Use consistent naming across related files

4 FORMATTING CONVENTIONS

4.1 Indentation

Use 2 spaces for indentation in all JavaScript and JSX files.

4.2 Using Capitalization to Aid Readability

- Use camelCase for variable and function names
- Use PascalCase for component names and class names
- Use UPPER_CASE for constants

4.3 Formatting Single Statements

Place single statements on the same line as the condition in if statements and loops.

4.4 Formatting Declarations

Declare each variable on a new line, even when in the same scope.

4.5 Formatting Multi-line Statements

For multi-line statements, break after operators and align with the start of the expression on the previous line.

5 NAMING CONVENTIONS

- Components: PascalCase (e.g., UserProfile, JobSearch)
- Functions: camelCase (e.g., getUserData, searchJobs)
- Variables: camelCase (e.g., userData, jobResults)
- Constants: UPPER_CASE (e.g., API_URL, MAX_RESULTS)
- Files: kebab-case (e.g., user-profile.js, job-search.jsx)

6 SCOPING CONVENTIONS

6.1 Lexical/Static Scoping

Use const and let for variable declarations to ensure proper block scoping.

6.2 Dynamic Scoping

Avoid using var to prevent unexpected scoping issues.

7 COMPILE ERRORS & WARNINGS

7.1 Errors

Address all compiler errors immediately. Common errors include:

- Syntax errors (e.g., missing brackets, semicolons)
- Type errors (e.g., assigning wrong type to a variable)
- Reference errors (e.g., using undeclared variables)

7.2 Warnings

Pay attention to ESLint warnings and address them promptly. Common warnings include:

- Unused variables
- Missing prop types in React components
- Accessibility issues in JSX

8 ENFORCING CODING STANDARD

To enforce coding standards across the project:

- Use ESLint with a custom configuration
- Implement Prettier for consistent code formatting
- Set up pre-commit hooks to run linters before allowing commits
- Use continuous integration to run linters on every pull request

9 APPENDICES

9.1 Appendix A – Example React Component

```
import React from 'react';
import PropTypes from 'prop-types';

const UserProfile = ({ name, email, jobTitle }) => {
  return (
    <div className="user-profile">
      <h2>{name}</h2>
      <p>Email: {email}</p>
      <p>Job Title: {jobTitle}</p>
    </div>
  );
};
```

```
UserProfile.propTypes = {
  name: PropTypes.string.isRequired,
  email: PropTypes.string.isRequired,
  jobTitle: PropTypes.string.isRequired,
};

export default UserProfile;
```