Coding Standards

1. General Guidelines

- Code should be **clean**, **modular**, and **well-documented**.
- Follow **DRY (Don't Repeat Yourself)** principles.
- All features must be developed following a **feature-branch model** and **merged via pull requests**.
- Use **meaningful variable**, function, and class names (e.g., postSkill(), sendMessage()).
- Always use **version control** (Git) and commit with meaningful messages.

2. File & Folder Naming Conventions

Type	Convention
Folder name	lowercase-with-hyphen
File name	camelCase or lowercase_with_underscores
Class name	PascalCase
Function	camelCase
Variable	camelCase

Example folder: user-profile
Example file: skillController.js

3. Language-Specific Standards

- *JavaScript (Frontend/Node.js)*
  -> Use **ES6+ syntax**
  -> Prefer const and let over var
  -> Use arrow functions:
  const greet = () => console.log("Hello");
  ->Always handle promises using async/await

```
try {
const result = await fetchSkills();
} catch (error) {
console.error(error);
}
```

  -> Use **strict equality (===)**
  ->Use eslint and prettier for code linting and formatting
- *HTML*
  ->Use semantic HTML5 tags (e.g., <section>, <article>)
  ->Indent nested elements properly
  ->Use alt attributes for images
  ->Use lowercase for tag names and attributes
- *CSS*
  ->Use external stylesheets or CSS-in-JS
  ->Prefer class over id for styling
  ->Use lowercase and hyphen-separated class names:
  .skill-card, .user-profile

4. Backend (If using Express/Node.js)

- Separate routes, controllers, services, and models
- Use .env for environment variables
- Use consistent status codes and response formats
  res.status(200).json({ message: "Skill added successfully." });
- Log errors and handle all edge cases
- Use middleware for authentication, validation, and error handling

5. Commenting & Documentation

- Use single-line comments for inline explanations
- Use multi-line comments for blocks
- Comment **why**, not **what**, especially for complex logic

6. UI/UX Design Principles

- Keep UI consistent across pages
- Use responsive design
- Provide user feedback
- Use color and spacing consistently

7. Security Best Practices

- Sanitize user input to prevent XSS and SQL Injection
- Hash passwords using bcrypt or similar libraries
- Use HTTPS and secure headers (if deployed)
- Validate all form data on both frontend and backend

8. Code Review Checklist

Before merging your code:

- Code compiles/runs without errors
- No console logs or commented-out code
- Follows naming conventions and formatting
- Has meaningful commit messages
- Passes lint and test checks
- Feature is complete and meets requirements