TEAM MENTORSHIP DOCUMENT

Development Team Leadership & Collaboration Framework

Professional Development & Technical Standards Guide

Prepared For: Development Team (Oviya, Priya, Interns)
Purpose: Team guidance, coding standards, and workflow protocols

---

1. INTRODUCTION

Leadership Approach

My role as team lead is to ensure clarity, consistency, and continuous growth for every team member. This document establishes the framework for how we work together, maintain code quality, and deliver professional results.
I believe in mentorship through empowerment—providing clear guidance while encouraging independent problem-solving. Every team member will receive the support needed to grow their skills while contributing meaningfully to our projects.

---

2. MENTORSHIP APPROACH FOR JUNIOR DEVELOPERS

2.1 Guiding Oviya, Priya, and Interns

Clear Task Instructions

- Every task comes with step-by-step guidance
- Clear acceptance criteria defined upfront
- Relevant examples and documentation provided
- Realistic deadlines with learning time built in

Support for Problem-Solving

- Encourage 15-20 minutes of independent troubleshooting first
- Available for questions and guidance when blocked
- Focus on teaching concepts, not just giving answers
- Share debugging strategies and best practices

Regular Check-ins

- Daily 15-minute standup to review progress
- Discussion of blockers and challenges
- Clear goals set for each day
- Immediate support when issues arise

Understanding Core Concepts

- Teaching the "why" behind technical decisions
- Explaining patterns and best practices
- Building foundational knowledge in:
    - API design and REST principles
    - Frontend component architecture
    - Database relationships
    - Authentication and security basics
    - Version control workflows

---

3. CODING STANDARDS

3.1 Folder and File Structure

Consistency is Key:

- Organized by feature/module
- Clear separation of concerns
- Logical grouping of related files
- Easy navigation for any team member

3.2 Naming Conventions

Components: PascalCase

Example: `ClientList.jsx`, `TaskCard.jsx`, `DashboardHeader.jsx`

Variables & Functions: camelCase

Example: `fetchClients()`, `userId`, `handleSubmit()`

Constants: UPPER_SNAKE_CASE

Example: `API_BASE_URL`, `MAX_RETRY_ATTEMPTS`

Files (non-components): kebab-case

Example: `api-client.js`, `date-formatter.js`

3.3 Commit Message Standards

Format: `type: brief description`

Types:

- `feat:` New feature
- `fix:` Bug fix
- `refactor:` Code improvement
- `docs:` Documentation updates
- `style:` Formatting changes

Examples:

```
feat: add client list component
fix: resolve login redirect issue
refactor: optimize task query performance

docs: update API endpoint documentation
```

3.4 Code Quality Requirements

Clean, Readable Code:

- Self-documenting variable names
- Short, focused functions (single responsibility)
- Consistent indentation (2 spaces)
- Comments for complex logic only

Before Every Commit:

- ✗ No `console.log()` statements
- ✗ No unused variables or imports
- ✗ No commented-out code blocks
- ✓ All functions properly named
- ✓ Error handling implemented

Modular & Maintainable:

- Functions should be reusable
- Components should be composable
- Logic separated from presentation
- DRY principle (Don't Repeat Yourself)

---

4. TASK ALLOCATION STRATEGY

4.1 Skill-Based Assignment

Oviya — Frontend Development:

- UI component creation

- Screen layouts and styling
- Form implementations
- Data display components
- API integration in frontend

Priya — Backend Development:

- API route creation
- Controller logic implementation
- Database queries and operations
- Middleware development
- Server-side validation

Interns — Supporting Tasks:

- Activity logging features
- Documentation updates
- Minor UI enhancements
- Testing and bug reporting
- Code cleanup tasks

4.2 Task Breakdown Philosophy

Clear Daily Goals:

- Tasks scoped to 2-4 hours max
- Short, achievable deliverables
- Immediate feedback on completion
- Progressive difficulty increase

Example Task Assignment:

```
Oviya: Create Client List Page
- Build table component
- Add filters for status
- Integrate with API
- Due: End of day

Priya: Build Client API Endpoints
- GET /api/clients (list with pagination)
- POST /api/clients (create new)
- PUT /api/clients/:id (update)
- Due: End of day

Intern: Document Client Module
- API endpoint documentation
- Component usage guide
- Due: End of day
```

5. BRANCHING STRATEGY

5.1 Branch Structure

`main`

- Production-ready code only
- Always stable and deployable
- Protected branch (no direct commits)

`dev`

- Integration branch for testing
- All features merged here first
- Testing ground before production

`feature/*`

- Individual task branches
- Naming: `feature/task-description`
- Examples:
    - `feature/client-list-component`
    - `feature/task-api-endpoints`
    - `feature/login-validation`

5.2 Workflow Process

Step-by-Step:

1. Create Branch:

bash

```bash
git checkout dev
 git pull origin dev

 git checkout -b feature/your-task-name
```

2. Work on Task:
    - Make your changes
    - Test thoroughly
    - Commit with clear messages
3. Push Branch:

bash

```bash
git push origin feature/your-task-name
```

4. Create Pull Request:
    - From `feature/*` → `dev`
    - Add clear description
    - Request review
5. After Approval:

    - Merge to `dev`
    - Delete feature branch
    - Pull latest `dev` for next task

Golden Rules:

- ✗ Never commit directly to `main`
- ✗ Never push broken code
- ✓ Always pull latest before creating new branch
- ✓ Always create PR for code review

---

6. PULL REQUEST REVIEW GUIDELINES

6.1 What I Review

Code Readability:

- Is the code easy to understand?
- Are variable names descriptive?
- Is the logic clear and well-structured?

Error Handling:

- Are errors caught and handled properly?
- Are user-facing error messages clear?
- Are edge cases considered?

Naming Conventions:

- Does it follow our standards?
- Consistent with existing codebase?

Breaking Changes:

- Will this affect existing functionality?
- Are database changes backward compatible?

Validations:

- Input validation on both frontend and backend
- Data sanitization implemented
- Security considerations addressed

API Consistency:

- Response format matches standards
- Proper HTTP status codes used
- Error responses structured correctly

6.2 PR Requirements

Before Submitting:

- Code runs without errors

- Feature tested locally
- No console logs or debugging code
- Commit messages are clear

PR Description Should Include:

- What was changed and why
- How to test the changes
- Any breaking changes or dependencies
- Screenshots (for UI changes)

6.3 Review Process

Timeline:

- PRs reviewed within 4 hours during work hours
- Feedback provided with clear explanations
- Requested changes must be addressed before merge

Feedback Approach:

- Constructive and educational
- Explain the "why" behind change requests
- Provide examples or references
- Celebrate good implementations

---

7. COLLABORATION & COMMUNICATION

7.1 Daily Team Sync (15 minutes)

Format:

- What did you complete yesterday?
- What are you working on today?
- Any blockers or questions?

Purpose:

- Stay aligned as a team
- Identify issues early
- Share knowledge and help each other

7.2 Communication Standards

Clear Communication:

- Ask questions without hesitation
- Explain blockers clearly
- Share progress proactively

Blocker Reporting:

- Report immediately, don't wait
- Describe what you've tried
- Share relevant error messages

Major Issue Protocol:

- Notify team lead immediately
- Document the issue clearly
- Avoid making assumptions about fixes

7.3 Team Culture

Respectful & Supportive:

- Every question is valid
- Learning is encouraged
- Mistakes are opportunities
- Collaboration over competition

Shared Documentation:

- Update docs when you learn something new
- Share useful resources with the team
- Contribute to team knowledge base

---

8. FINAL NOTES

Commitment to Excellence

Our goal is to deliver high-quality, maintainable code while ensuring every team member grows professionally. This requires:

- Clarity in all communications and task assignments
- Teamwork in supporting each other's growth
- Continuous improvement in our skills and processes
- Professional standards in everything we build

Open Door Policy

I am always available to:

- Answer questions and provide guidance
- Review code and discuss improvements
- Solve blockers and technical challenges
- Support your professional development

Growth Mindset

Remember: Everyone was a beginner once. Ask questions, make mistakes, learn continuously, and help your teammates do the same.
Together, we build better.
