Snap Vision

Coding Standards Document

Demo 4 – Updated Version

Table of Contents

This document defines the coding standards and conventions followed by our team for Snap Vision. These guidelines are designed to maintain consistency and ensure high-quality code throughout the project. Adhering to these practices is crucial in a team environment to promote clarity and adaptability for all members.

1. Objectives

- Maintain a uniform coding style across all contributors.
- Improve code readability and facilitate collaboration.
- Minimize bugs and streamline onboarding for new developers.
- Enable future scalability and support modular development.

2. File Structure

The project is divided into two main components:

- **Frontend**: A React Native app (snap-vision) built using TypeScript and following Atomic Design principles.

- **Backend**: A Node.js service (snap-vision-backend) with the plan to use Firebase Cloud Functions.

```
snap-vision/
├── __tests__/              # Jest test files
├── android/                # Android-specific native configurations
├── src/
│   ├── assets/             # Fonts, icons, images
│   ├── components/
│   │   ├── atoms/          # Basic UI components
│   │   ├── molecules/      # Combinations of atoms
│   │   └── organisms/      # Larger composed UI structures
│   ├── context/            # React context providers (e.g., theming, auth)
│   ├── hooks/              # Custom reusable hooks
│   ├── navigation/         # Stack/tab navigators
│   ├── screens/            # Main app screens (e.g., Login, Map, Home)
│   ├── store/              # Global state (e.g., Redux, Zustand)
│   ├── theme/              # Theme config for light/dark mode
│   ├── types/              # Shared TypeScript types/interfaces
│   └── utils/              # Helper functions
│   └── services/           # Handle business logic
├── web/                    # Web support
├── .env                    # Environment variables
├── App.tsx                 # Entry point
└── config_files/           # Configs for Firebase, testing, etc.


snap-vision-backend/
├── __tests__/              # Jest test files
├── pois/                   # Building and Room POIs
├── scripts/                # Scripts to manage POIs, crowd reports
├── middleware/             # Authorisation
├── src/
├── index.js/               # Entry point
├── .env                    # Environment variables
```

3. General Coding Standards

Languages:

- TypeScript (frontend)
- JavaScript (Node.js backend)

Naming:

- Files: camelCase (e.g., userProfile.ts)
- Components: PascalCase (e.g., HomeScreen.tsx)
- Functions: camelCase
- Tests:
  - Unit: PascalCase (e.g. BadgeService.tsx)
  - Integration: .integration suffix

Linting & Testing:

- Ensure all tests pass before committing and merging
- Ensure that the lint is successful before committing and merging
- Prettier and es linting are used to ensure uniformity and prevent broken code from being committed

Best Practices:

- Write clear, descriptive comments
- Keep code formatting clean and consistent.
- Follow an atomic structure to maximize reusability and efficiency.
- Require at least one team member to review code before approving pull requests.

Error Handling:

- Use try/catch blocks to avoid uncaught errors
- Allow the app to fail gracefully

- Provide meaningful and user-friendly error messages to aid debugging and improve user experience.

Security Standards:

- All API keys, tokens, and credentials are stored in `.env` files.
- No sensitive keys may be hardcoded in source code or committed to version control.

4. Firebase Standards

Authentication:

- Firebase Auth is used for user login, registration, and session management.

Data Management:

- Firestore is used to store user profiles, POIs, floorplans, timetables, crowd reports and navigation metadata.

Security Rules:

- Firebase Storage and Firestore security rules strictly enforce role-based access control.
- Rules ensure only authenticated and authorized users can access/update relevant data.

Emulators:

- Firebase Emulator Suite is used in development to test rules and database interactions without modifying the database.

Testing & Monitoring:

- Firebase Test Lab is used to run the app on a wide range of physical and virtual devices for usability and compatibility testing.

- Firebase Performance Monitoring (FPM) tracks real-time application behavior, including API response times, data loading, and user interactions, to identify and resolve performance bottlenecks.

5. Version Control

Git Branching Strategy:

- main: Stable, production-ready code, release are tagged here
- dev: Development branch for integration and sprint testing
- feature/<name>: New features or components
- test/<name>: Test-specific branches for isolated testing scenarios
- fix/<name>: For bug fixes without interrupting the rest of the workflow

Project Board (GitHub Projects):

- Columns: To Do → In Progress → Done
- Labels used for priority, size of task, component type (frontend/backend), and feature area
- Roadmap used to track sprint planning and milestone targets.

Tagging & Releases:

- Use semantic versioning: v<MAJOR>.<MINOR>.<PATCH>
- Tag releases in main and document release notes.
- Hotfixes follow fix/<name> workflow and are merged back into main and dev.

6. Testing

Unit Testing:

- Jest is used to test components and services.
- Test files are located in the __tests__ folder and separated into unit and integration tests

Integration Testing:

- The Firebase Emulator Suite is used to test Firestore, Authentication, and Cloud Functions in isolation.
- Emulators enable local simulation of real-time database behavior and security rules.

Security Testing:

- Firebase Rules Unit Testing is used to simulate Firestore and Storage security rules.
- Tests verify that only authorized users can access or modify protected resources.

Performance Testing:

- Firebase Performance Monitoring (FPM) tracks real-time application performance.
- Metrics include API response times, screen load times, and user interaction latency.
- Identified bottlenecks must be addressed before release.

Usability Testing:

- Firebase Test Lab is used to run the app on a variety of physical and virtual devices.

- Tests include layout responsiveness, navigation flows, and user accessibility.
- Usability feedback is incorporated into sprint review and iteration planning.

Coverage & CI:

- A CI pipeline (via GitHub Actions) executes tests and generates coverage badges.
- Pull requests are prevented from merging if test coverage falls below the specified thresholds.