Project Coding Standards

Summary
This project uses **Prettier** as the standard code formatter. Visual Studio Code's Format on Save feature automatically applies Prettier's **default** formatting rules, ensuring consistent code style without requiring any custom configuration files. Beyond formatting, **TypeScript ESLint** is used to maintain code quality and enforce coding best practices. All developers must keep Format on Save enabled to ensure that formatting and linting standards are consistently applied across the project's TypeScript codebase.

Formatting Standard

• Tool: Prettier (integrated via the esbenp.prettier-vscode extension in VS Code).
• Activation: VS Code's editor.formatOnSave applies formatting automatically on file save.
• Configuration: The project uses Prettier's **defaults**. There is no project-level .prettierrc or other Prettier config file.

Prettier Scope

Prettier handles stylistic formatting only (e.g., indentation, semicolons, quotes, trailing commas, line wrapping). It does **not** enforce naming conventions, complexity limits, or architectural rules—those are the responsibility of linters and code review.

Linting and Code Quality

TypeScript ESLint is used to enforce code quality, best practices, and rules that Prettier does not cover. The linting configuration (e.g., .eslintrc) contains rules for potential errors, stylistic choices not covered by Prettier, and TypeScript-specific constraints via @typescript-eslint.

Developer Requirements

• Ensure VS Code has Format on Save enabled: editor.formatOnSave: true.
• Set the default formatter for TypeScript to Prettier in VS Code settings if desired:
{ "[typescript]": { "editor.defaultFormatter": "esbenp.prettier-vscode" } }
• Run ESLint as part of CI or pre-commit hooks to enforce lint rules beyond formatting.
• During code review, focus on logic and architecture; formatting issues should be minimal due to automatic formatting.

Naming Conventions

• Use PascalCase for all interfaces and types.
• Name functions in camelCase using a verb + object pattern.
• Use camelCase for all variables and function parameters.
• Prefix boolean variables with *is* or *has* when possible,
• Write constants and configuration presets in UPPER_SNAKE_CASE.
• Name Firestore collections in lowercase plural form.
• Use camelCase for Firestore document fields and nested object keys.
• Give files descriptive names and group them by feature or utility.