Coding Standards

This document describes the conventions, styles, and tools used in our project to ensure uniformity, clarity, flexibility, reliability, and efficiency of our code.

Language

The project uses Dart as the primary programming language, with Flutter for mobile development.

File Structure

- cos301_capstone/
    - .dart_tool/
    - .firebase/
    - .firebaserc
    - .flutter-plugins
    - .flutter-plugins-dependencies
    - .gitignore
    - .metadata
    - analysis_options.yaml
    - android/
    - assets/
    - build/
    - coverage/
    - docs/
    - firebase.json
    - ios/
    - lib/
        - Navbar/
            - Desktop_View.dart
            - Mobile_View.dart
        - User_Profile/
            - Desktop_View.dart
            - Mobile_View.dart
            - Tablet_View.dart

- linux/
- macos/
- pubspec.lock
- pubspec.yaml
- test/
- web/
- windows/

Linting

The project uses the `flutter_lints` package for linting, which includes a set of recommended lints to encourage good coding practices. The lint rules are configured in the `analysis_options.yaml` file located at the root of your package. This configuration ensures uniformity across the codebase. `flutter analyze` is run to test these linting rules.

Custom Lint Rules

In addition to the default rules provided by `flutter_lints`, the project has enabled some custom lint rules. These include:

- `camel_case_types`: Ensures that type names use CamelCase.
- `curly_braces_in_flow_control_structures`: Ensures that flow control structures are wrapped with curly braces.
- `avoid_empty_else`: Avoids empty else blocks.
- `avoid_unused_constructor_parameters`: Avoids declaring constructor parameters that are not used.
- `unnecessary_null_checks`: Avoids null checks when the value can't be null.
- `avoid_returning_null_for_void`: Avoids returning null in void methods.

Ignoring Lint Rules

If necessary, lint rules can be ignored for a single line of code or a specific Dart file by using the `// ignore: name_of_lint` and `// ignore_for_file: name_of_lint` syntax.

Testing

The project uses the `flutter_test` package for testing. Unit and integration tests are written for Flutter components and Firebase Functions to ensure reliability and ease of maintenance.

Dependencies

Dependencies are managed through the `pubspec.yaml` file. To upgrade to the latest versions, run `flutter pub upgrade --major-versions`.

Architecture

The project adheres to a modular architecture in Flutter, separating UI, business logic, and data layers. This structure provides flexibility and clarity in the codebase.

Extensibility

The project uses Flutter's plugin system to integrate new features and services seamlessly. The Firebase Firestore schema is structured to allow easy addition of new data fields and relationships.