COS301 Capstone Demo 2 Coding standards

de Villiers, Charles
u16056559

Kirsten, Eric
u16020431

Nel, Johan
u16354029

Ross, Justin
u17080526

Schwikkard, Dylan
u16120206

24 May 2019

1 Introduction

This is a document that enumerates and explains the various coding standards followed by the team The Tenacious Technicians. These standards ensure for readable and consistent code that is essential for group programming. For our project we chose a new emerging technology to develop our app with: Flutter. Flutter uses a new OOP language developed by Google for their internal projects - Dart.

2 Coding conventions

Flutters framework works with a very modular 'plug-and-build' approach in the form of Widgets. Widgets are at the core of the Flutter framework and everything that is built or displayed is a Widget. Widgets all perform different operations and influence the Widgets that are nested within them. Using this top-down tree like approach where Widgets plug into and hold each other we can achieve any and all operations and UI designs that would be possible on either android or iOS.

Using these Widgets has a variety of benefits for performance. This is because only Widgets that have changed visually are redrawn if they are still visible on the screen. We have the design option between either a Stateless Widget or a Stateful Widget. The later being able to update itself on request, with the Stateless having to be redrawn. By using these two types intelligently and correctly we are able to streamline the efficiency of our app with regards to rendering and drawing it on the screen.

3 Naming conventions

- In Dart there is a standardized naming convention, and we intend to follow it to prevent ambiguity or going against the general feel of the language.
- Variables use lowerCamelCase while methods and functions use UpperCamelCase.
- Encapsulation and class-object variable and method scoping is built into the naming of the variables or methods. For instance if you want a field to be private, it needs to start with an underscore: '_exampleVar'.

4 Layout Rules

- In Dart and specifically Flutter, your code can quickly become longer horizontally than your screen or work space due to the amount of tabbing that arises from the use of embedding Widgets inside each other. Meaning it often leads to unstructured-looking or unfriendly code. As such we take steps to ensure our code still adheres to the Flutter and Dart structures while remaining easy to read and understand.
- When code has grown too far horizontally we extract it as a new Widget Object and simply plug that into the code to reduce the clutter.
- Stateful Widgets are essentially a Stateless Widget that manages States, making it Stateful, the states are not strictly encapsulated and as such must always be kept together in code to prevent confusion or misunderstanding.

Apart from these rules we make use of general coding standards as explained below for spacing etc.

5 Commenting Practices

- Whenever a new Widget is created, its purpose should be briefly commented.
- Whenever a member has implemented a new feature or a new way of doing something, as we are learning all the time due this being a new framework using a new language, we expect them to briefly comment how it works and why they did it.
- When a member feels what they have done might not be intuitive to read and understand it should be briefly explained in comments.
- Each file should have a comment header at the top explaining what should be in the file, to ensure we encapsulated our code correctly and for good practice.

- In regards to commenting we follow good practices as described below.

6 Code Review Process:

When a feature is completed and pushed, the team member that pushed the code asks at least two team members to review the new feature. The team then downloads the new code using git and compiles the application using Android Studio to an Android device or emulator. Feedback is then given to the team member responsible.