Dungeons & Dragons Game State Manager

Coding Standards
Developed for Retro Rabbit by Optimize Prime

Team:
James Hertzog
Thomas Scholtz
Jason Davidson
Ruben Denner

Introduction

This is a document that enumerates and explains the various coding standards followed by the team Optimize Prime. 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.

1. 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.

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 langage.
- 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’ .

Layout Rules:

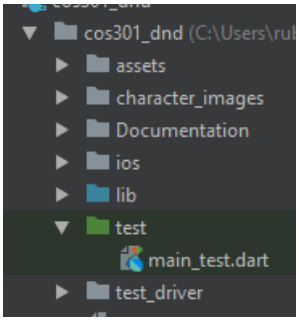
- In Dart and specifically Flutter, your code can quickly become longer horizontally than your screen or workspace due to the amount of tabbing that arises from the use of embedding Widgets inside each other. Meaning it often leads to unstructured-looking or unfriendly code. As such we take steps to 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 net strictly encapsulated and as such must always be kept together in code to prevent confusion or misunderstanding.

- We have a Singleton AppData class that we use to transfer data around our app without losing it or having to re-fetch it on every new screen. This class has a strict Static only policy and it has been made a singleton to enforce zero duplication as this could have disastrous results.
- Apart from these rules we make use of general coding standards as explained below for spacing etc.

Commenting Practices:

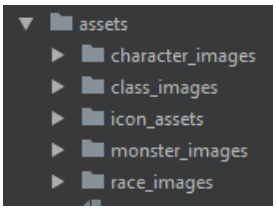
- Whenever a new Widget is created, its purpose should be briefly commented.
- Whenever a member has implemented a new feature or a new way of doing something, as we are learning all the time due this being a new framework using a new language, we expect them to briefly comment how it works and why they 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.

2. File Structure

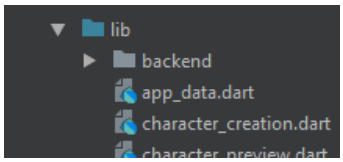


Front end:

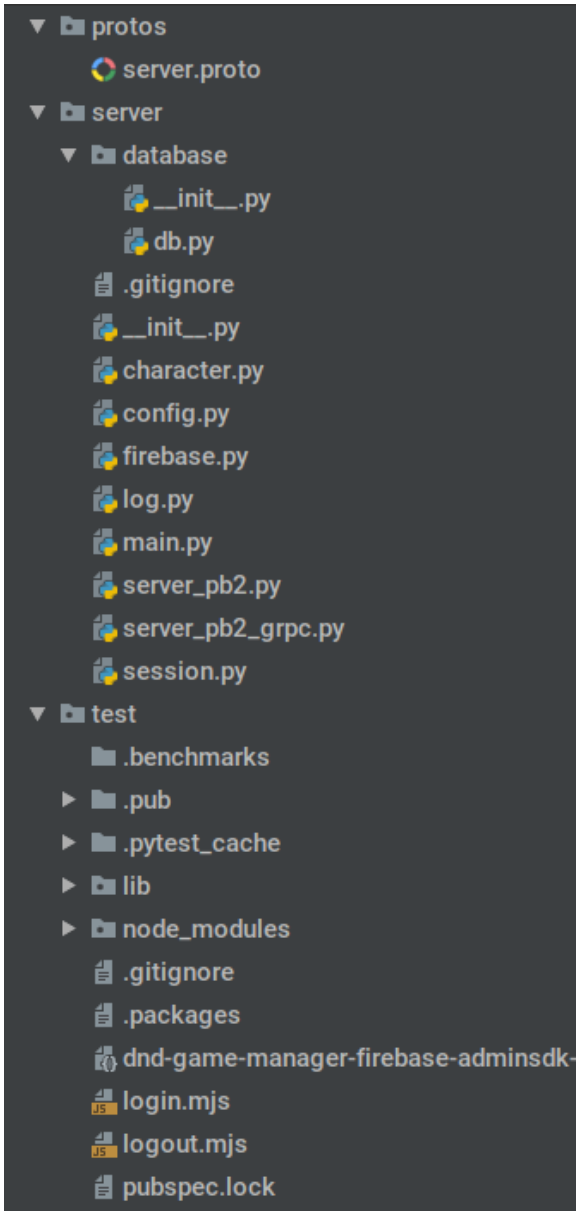
The file structure of the front end is shown on the left.
Assets is the resources that the application requires. It mostly consists of images and is shown on the right.
Documentation is where artifacts relating to the documentation of the product is stored.



Lib (shown right) is where the source code of the product is stored. Lib is has a folder called backend that contains code that integrates with the backend of the product. App_data.dart contains global information about the instance of the app.



Back end:



The back end file structure is shown above. The main application is inside the server directory. The database directory contains the database ORM functions and objects. The test directory contains all the testing code. The lib directory inside the test directory contains the back end library for the Android app to communicate with the server.

3. Code Review Process

When a feature is completed and pushed, the team member that pushed the code asks the team 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.