Software Architecture Specification

For the PictsManager project we have chosen the Bloc Architecture pattern.

Why Bloc ?

Bloc makes it easy to separate presentation from business logic, making your code fast, easy to test, and reusable.

When building production quality applications, managing state becomes critical.

As developers we want to:

- know what state our application is in at any point in time.
- easily test every case to make sure our app is responding appropriately.
- record every single user interaction in our application so that we can make data-driven decisions.
- work as efficiently as possible and reuse components both within our application and across other applications.
- have many developers seamlessly working within a single code base following the same patterns and conventions.
- develop fast and reactive apps.

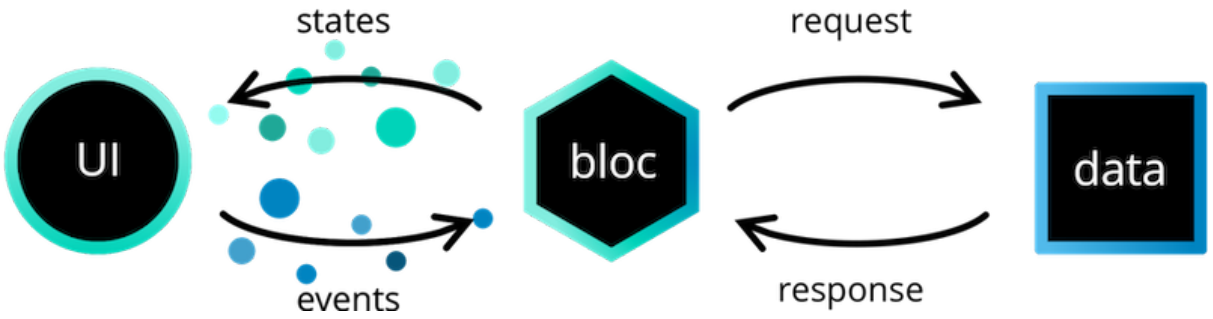
Bloc was designed to meet all of these needs and many more.
There are many state management solutions and deciding which one to use can be a daunting task. There is no one perfect state management solution! We chose Bloc because it was the one that works best for us.

Bloc was designed with three core values in mind:

- **Simple**: Easy to understand & can be used by developers with varying skill levels.
- **Powerful**: Help make amazing, complex applications by composing them of smaller components.
- **Testable**: Easily test every aspect of an application so that we can iterate with confidence.

Overall, Bloc attempts to make state changes predictable by regulating when a state change can occur and enforcing a single way to change state throughout an entire application.

Architecture



Using the bloc library allows us to separate our application into three layers:

- Presentation
- Business Logic
- Repository

We're going to start at the lowest level layer (farthest from the user interface) and work our way up to the presentation layer.

Repository

The repository serves as an abstraction between the client code and the data provider so that as a developer working on features, you don't have to know where the data is coming from. It may come from an API provider or Local database.

Business Logic Layer

The business logic layer's responsibility is to respond to input from the presentation layer with new states. This layer can depend on one or more repositories to retrieve data needed to build up the application state. Think of the business logic layer as the bridge between the user interface (presentation layer) and the data layer. The business logic layer is notified of events/actions from the presentation layer and then communicates with the repository in order to build a new state for the presentation layer to consume.

Presentation Layer

The presentation layer's responsibility is to figure out how to render itself based on one or more bloc states. In addition, it should handle user input and application lifecycle events.