Flutter In Action: Design Patterns : Coding Simplified

Posted by Khushboo Uchat May 23, 2021

As part of the CE program badge evidence, I am here with one more blog post. This time it is about Design Patterns. You have started reading further means you are interested *(or may be you are just checking out how good I am with boring topics !!)* Well whatever the case may be, but this time I will not bore you with theoretical write-up. **I will assume that you are familiar with different type of design patterns.** I am here to talk about my personal experience and how it helped me to solve the practical and complex problems.

1. Singleton

For me, **this pattern helped me solving the complex issue of keeping the data at a single and global place** without having Redux architecture in react application. Well, just to give overview, Redux is state management framework which enables data management easy by keeping data at one centralized location when combined with React. Without Redux your data is spread across the application in the form of state of different components. Also there is no such place where you can keep the data globally just like we do in Redux Data Store.

So when it came to implement Theme in Arena, I was presented with the same question. ***Where can I keep the value of selected theme by user so that it is available through out the application?*** Because every component needs to change the style based on selected theme so it should be available globally. **I choose the React Context as appropriate place and introduced getter and setter functions** in context so that every component needs to read only at single place to get the selected theme.

Code Artifact :
https://devcloud.ubs.net/ubs-ag/gt/gcto/arena/aa45404-arena/arena-frontend/-/tree/master/src/components/Common/ThemeSwit…
https://devcloud.ubs.net/ubs-ag/gt/gcto/arena/aa45404-arena/arena-frontend/-/tree/master/src/context

2. Redux (Observer Design Pattern)

Though we do not have Redux in Arena, but personally **I am very big fan of Redux design pattern.** It keeps the application clean and prevents lots of nesting of states and components. Redux design pattern combines lots of design pattern principles and it may not be possible to describe here. *(I can go on but then you are not here to read a book !!).* So I will restrict myself with the example I am interested in which is **the Oberserver Pattern**. All your "write" logic goes into a single function, and the only way to run that logic is to give Redux a plain object that describes something that has happened. The Redux store calls that write logic function and passes in the current state tree and the descriptive object, the write logic function returns some new state tree, and the Redux store notifies any subscribers that the state tree has changed. I have designed a few components and you can find the collection here.

https://devcloud.ubs.net/t653098/react-redux-demo

3. Single Responsibility Principle

Arena is all about gamification and core component which creates an event or game was written by me 8 months ago. After that Arena has evolved a lot and different requirements keeps on coming. We are using Azure Cosmos Graph DB which does not have transactions and rollback like we have in full fledge database (Why doesn't Microsoft have it ? A point to investigate. I would really love to know the thought process of the person who decided not to have it !! Again I started talking... Back to blog please).

So we have implemented backup restore functionality in event flow to handle internal errors and loss of data. While working on this feature, a colleague faced an issue where response was getting returned earlier though error handling cleanup is running in background. The culprit was throw error and handling it in try catch block. At various places it was being handled and two flows were out of sync. I have changed the code so that we only have one logError function which handles all kind of errors and run the clean up flow. The result was that now we only have one place and we know where to start restore action if error occurs and clenaup flow is run. So I followed Single Responsibility Principle on my own way as design patterns are all about concepts rather than definitions.

https://devcloud.ubs.net/ubs-ag/gt/gcto/arena/aa45404-arena/arena-serverside/-/blob/master/EventAPI/routes/post-calls/dr…
https://devcloud.ubs.net/ubs-ag/gt/gcto/arena/aa45404-arena/arena-serverside/-/blob/master/EventAPI/utils/eventUtil.js

I hope this examples will help someone to find out solutions of their customized issues. What I would really like to share here by my personalized experience is that **"There is no such place where you will find the hard core definitions of design patterns. It is something to learn by experience and implementation in your day to day programming just like life experiences"**
48 Views

Rohit Chandiramani
May 24, 2021 8:24 AM
Great article with examples to understand the usually ignored "Design Patterns"! Liked the crisp and efficient style.
Keep them coming!