**Criterion C: Development**

### Techniques

Provide a list at the top covering the important coding techniques you used. Then include a section below for each, with details and often with **CODE AND UI screenshots**. Do not summarize or talk through your code directly. Rather discuss WHY you chose to code the way you did, WHY you chose a certain organization of classes, HOW you designed difficult algorithms, etc.

**Provide explicit evidence of algorithmic thinking** (i.e., how and why you chose a loop and an arraylist)

Include citations to sources INLINE in your discussion, wherever you got ideas or needed to look something up. **If you don't include INLINE citations that refer to the appendix, you'll lose a point.**

Anytime you go to the web and find something worthwhile that you’ll use/learn from, add it to your appendix.

Technique #1

The first technique that I employed was creating my own ADT, which is comprised of a list of nodes. When I was tasked with creating custom ADTs for complexity, I recalled what I had learned while I was learning about Nodes and binary search trees. Creation of this ADT enabled me to create the priceWatchList for each user which would store the items that a user desires to receive alerts for when they reach an affordable price.

I implemented my own map and made use of the Dart language’s support for “Templates”. This enabled me to create a Map class that I can use for multiple different class variables.

Graphical user interface, text, application

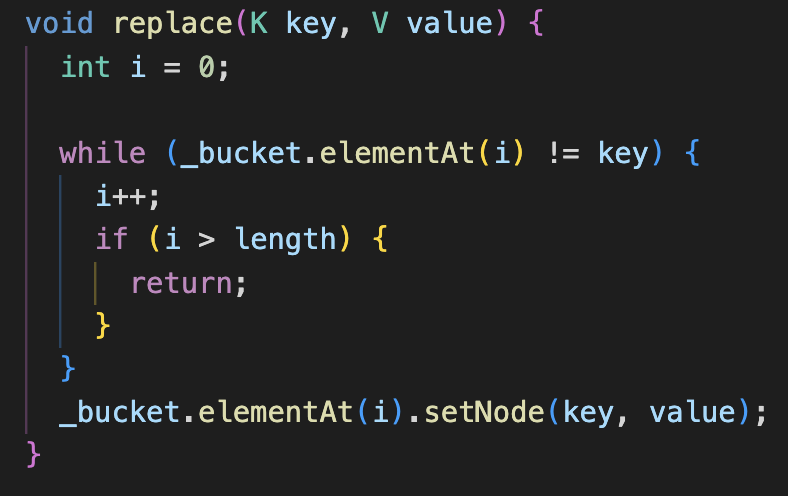
Description automatically generated

Text

Description automatically generated

The replace method I created for the Map class looks at whether a key-value pair already exists. Here I used a while loop to traverse through the bucket of Nodes. This sentinel check looks to see whether a key is already existing, and if it doesn’t exist, it will create a new node with that assigned key value pair and add it to the bucket of nodes.

In order to implement my own map, I created my own Node class. When creating this class, I referenced the powerpoint lecture that we had on Binary Search Trees to come up with this solution. **Looking ahead, since I am using this Node class, I will be able to do traversals quite easily using recursion**.



Technique #2

Multiple users are likely to use the application I create. Also, I added the variable PriceWatchList which is a Map with the Key-Value pair being String and double value. When a user adds an item to a price watch list, the key will represent the item name and the value with be the expected price the user is currently looking out for.

Text

Description automatically generated

Technique #3

The third technique I employed was making use of global constants. To have appropriate procedural decomposition and maintain clean coding conventions, the most common functions that I needed to call when creating flutter UI elements were instead put in the constants class.

Since Flutter makes use of stateful and stateless widgets that have a wide variety of parameters, I created my own customTextFormField that will enable me to create text fields with ease. Instead of having to write lots of code, I can simply put in the parameters of what I want. This is what I used when I created my login page.

Text

Description automatically generated

Text

Description automatically generated

The PriceMatch application has multiple UI screens to navigate. To make it easier to route to different pages, I added my main application class to my global constants file, making it accessible and eliminating redundancy of creating new routing methods.

Technique #4

The fourth technique I made use of was using a database. In my case, I made use of Google’s Firebase Firestore, which is a NoSQL database.

Once a user registers for an account on the signup page, , the following method is called. This method serves 2 purposes. Firstly, it creates the user using the provided credentials in the database, and two, the User constructor is called, creating a new user.

Graphical user interface, text

Description automatically generated