Criterion C: Development

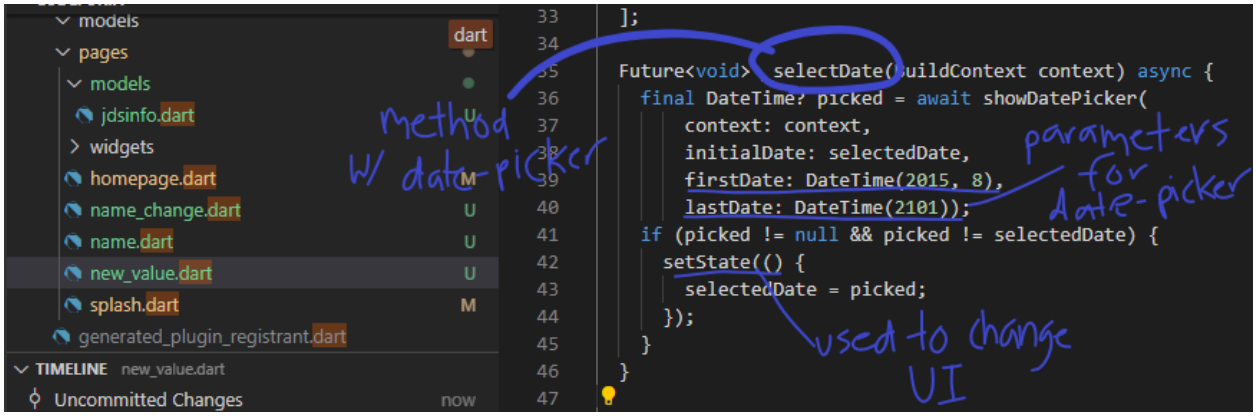
Vidhya Narayanan
Pierson
Period 3 Computer Science
22 March 2022

**(SOURCES USED/REFERENCED HERE ARE PROPERLY CITED IN THE APPENDIX.
DIFFERENT DESIGN VERSIONS WILL ALSO BE INCLUDED IN THE APPENDIX)**

Key Techniques:

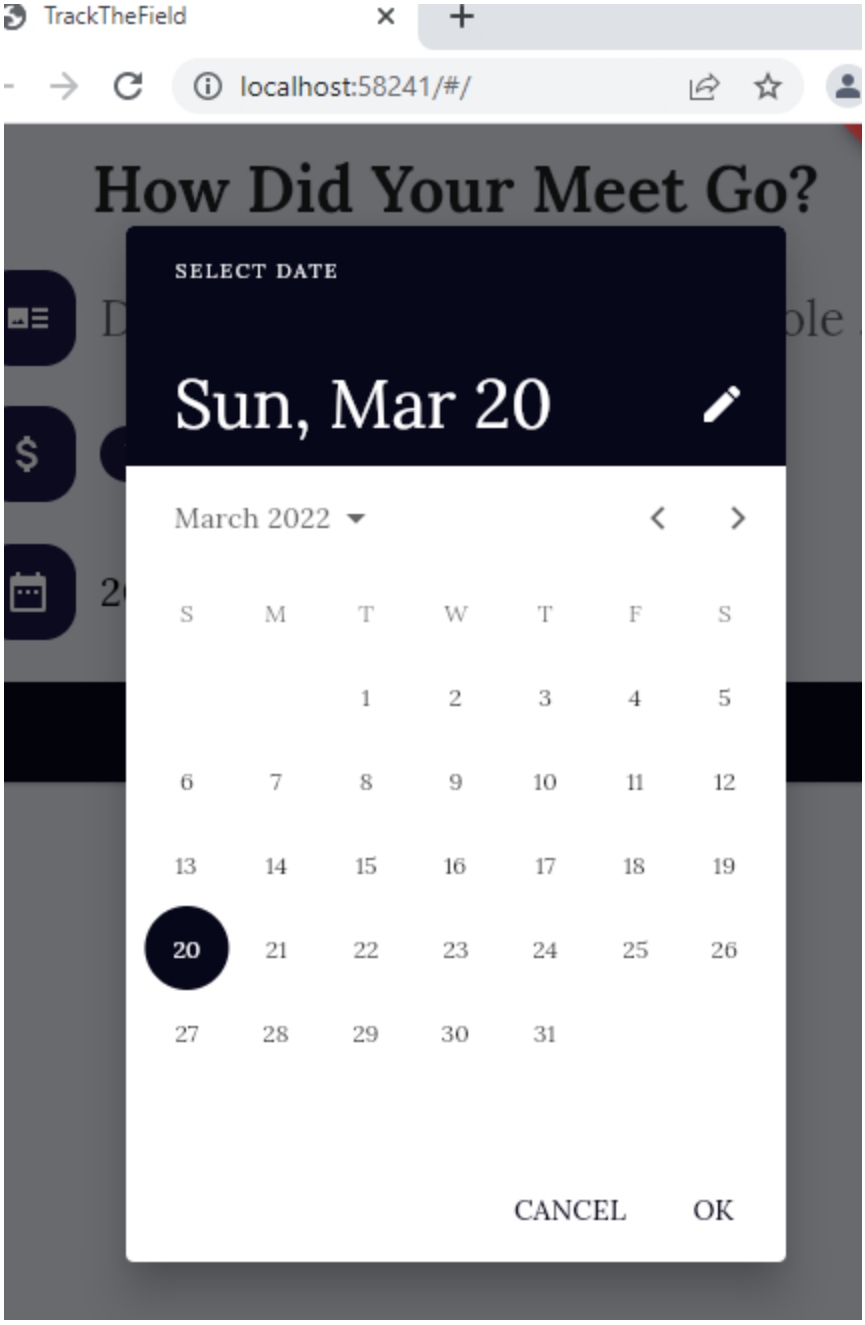
1.) Procedural Decomposition
    - user-defined methods (with parameters and appropriate return values)
2.) Storing User Input as Objects
    - User-defined objects as data records
    - If-else statements
    - Try catch blocks
    - For-loops
3.) Usage of FL spots & Arrays for Line Graph
    - Sorting
    - Searching
    - Use of additional libraries
4.) Use of Arrays to Establish Separate Tabs
    - Complex selection / multiple conditions
5.) Use of Hive Database

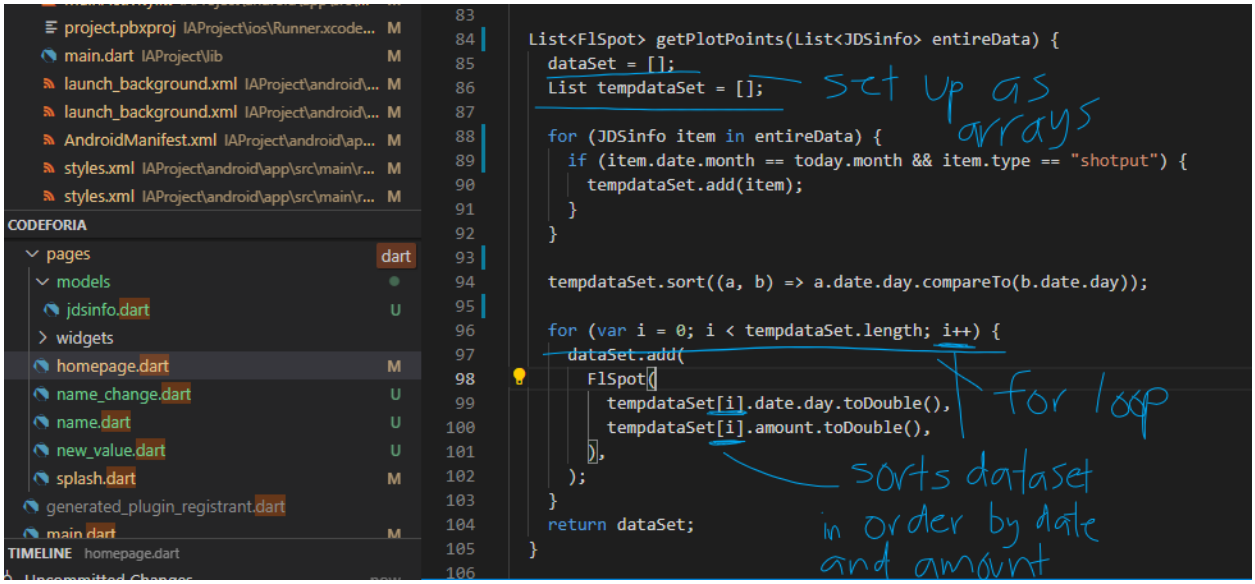
Procedural Decomposition:



My first example of **procedural decomposition is in my selectDate method** located in the new_value class,, which is used to initialize flutters Date Picker feature when the user chooses what date their throw took place. The method provides an initial date, parameters for what time span the calender can operate in, and includes and if statement for what to do if the user does not choose a date, chooses the current, date or chooses a different date. **(Source #1).** If the user doesn't choose a date or picks the current date, the UI of the info entry screen will show the current date. If the user manually picks a different date, that is the date that will show. *The ingenuity of this technique is that it allows for the programmer (me) to be able to change the UI (w/ stateful widget) by simply calling this method rather than having to manually set up the DateTimePicker whenever the user inputs data.* **(Source #7)** This data is processed in the

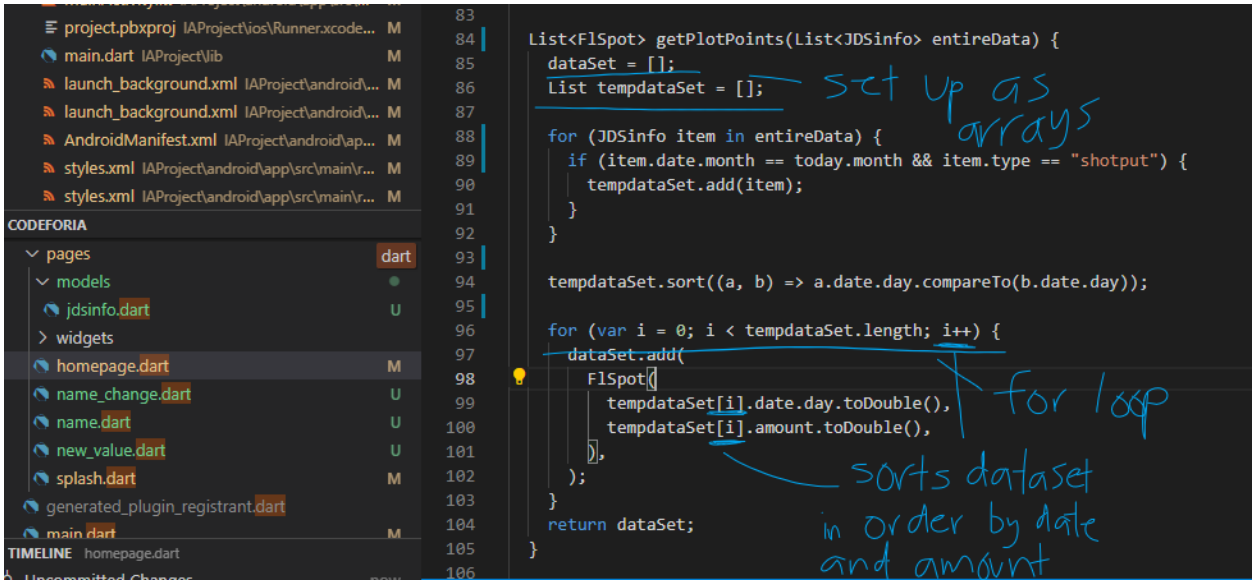
JDSinfo, or database helper class that I will elaborate on further later.





Next, on our homepage, we have a **user-defined method with parameters** as shown in our ***List<FlSpot> getPlotPoints method***, which is the method used to collect data for shot put. There are two other methods identical to this one that process data for discus and javelin. The **parameters** for this method are the entireData located inside the database helper "JDSinfo" class. Further down in the method you can see that the **method also provides an appropriate return value titled dataSet(Source #1).** dataSet is an array that is initialized at the top of the method (as shown in the screenshot) which then accepts data based on whether the item type (event chosen) matches the if statement's requirements **(Source #2). The ingenuity of this technique is that it separates each event into different dataSets, making the process of only needing to weed out the date of each throw a reality.**

Usage of FL spots and Arrays for the Line Graph:

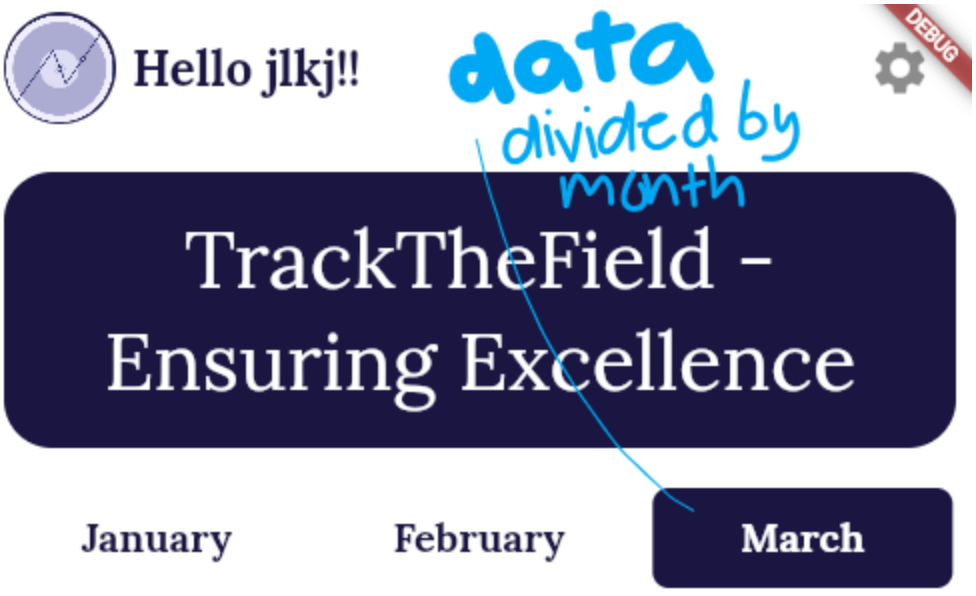


In this screenshot, as you can see I used arrays in order to organize the dates of when my client performed their throws. Directly prior to the for loop, there is a statement that ***sorts*** the date and time the user chooses to the date and time of the previous input, and this process repeats each time the user inputs new data. **(Source #3)** Then, inside the for loop, you can see that for each date and the amount the user adds, it is added to the tempDataSet which will be used for our line graphs. This method is just an example of what happens when the user chooses item type shot put, and there are two identical methods to this one that deals with discus and javelin, with names getPlotPoints1 and getPlotPoints2. **(Source #4)** The reason the if statement asks whether the item.date.month is equal to today's month is that in the selectMonth() method, **each month is given an index number and that number divides up the data month by month, SEARCHING through the data, and helps users select which month they want to see (example of ingenuity)**:

```
Widget selectMonth() {
  return Padding(
    padding: EdgeInsets.all(
      8.0,
    ), // EdgeInsets.all
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceAround,
      children: [
        InkWell(
          onTap: () {
            setState(() {
              index = 0;
              today = DateTime(now.year, now.month - 2, today.day);
            });
          },
          child: Container(
            height: 50.0,
            width: MediaQuery.of(context).size.width * 0.3,
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(
                8.0,
              ), // BorderRadius.circular
              color:
                  index == 0 ? Color.fromARGB(255, 26, 22, 65) : Co
            ), // BoxDecoration
            alignment: Alignment.center,
```

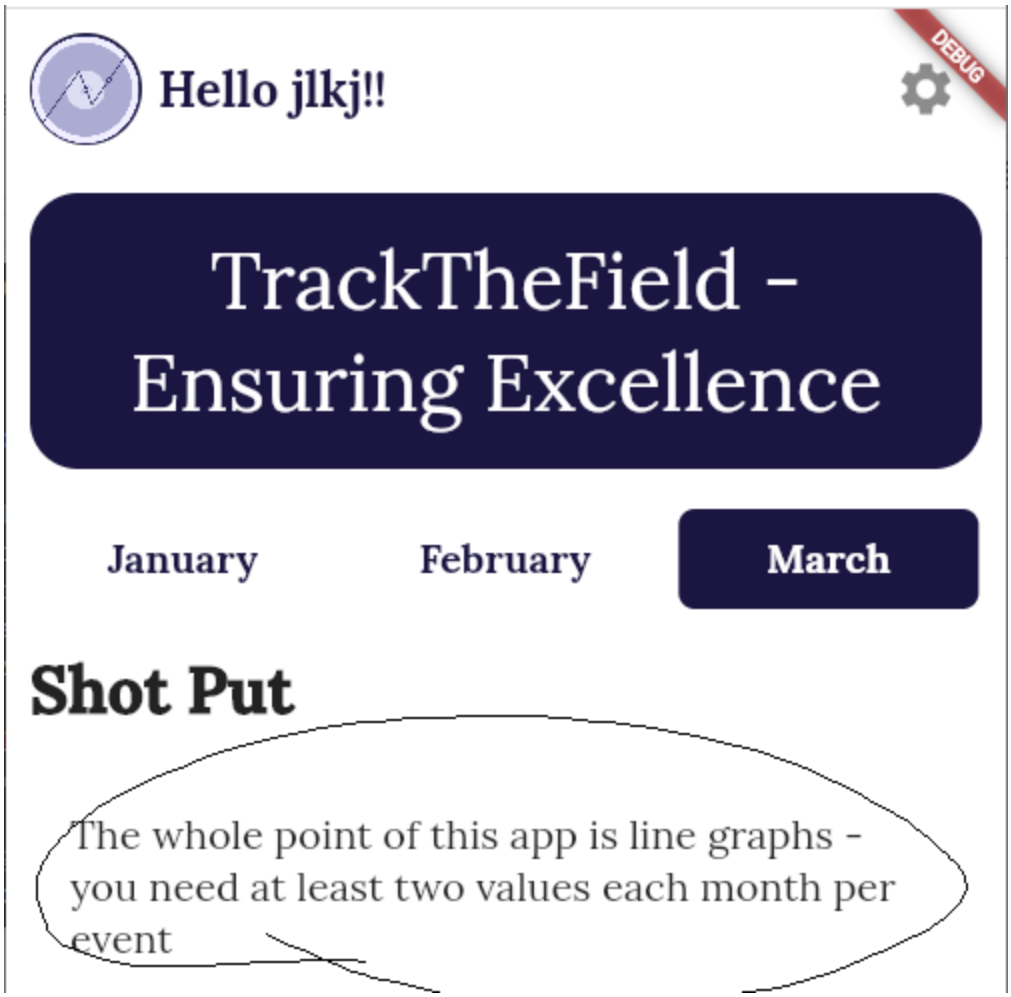
example: index for two months ago

creates select month tab



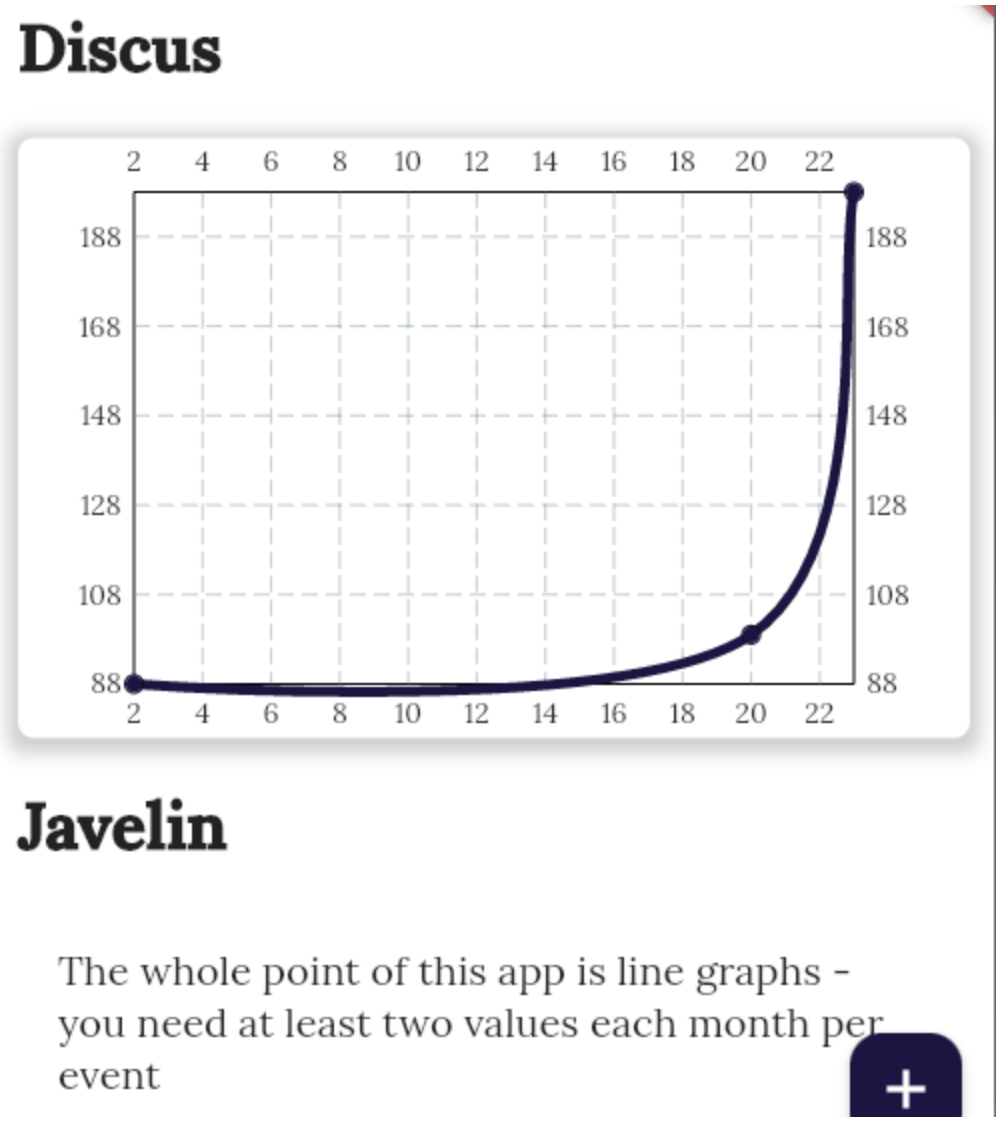
```
selectMonth(),
//

Padding(
  padding: const EdgeInsets.all(
    12.0,
  ), // EdgeInsets.all
  child: Text(
    "Shot Put",
    style: TextStyle(
      fontSize: 32.0,
      color: Colors.black87,
      fontWeight: FontWeight.w900,
    ), // TextStyle
  ), // Text
), // Padding
dataSet.isEmpty || dataSet.length < 2
```

After selecting which month the user wants to view, the homepage will display data for that month. At the bottom of the screen, you can see an if-scenario for when the user inputs less than two pieces of data per month per event. (Source #8) In that instance, the app will tell them to add more if they want a line graph: **the ingenuity of this is that it makes it clear to the user what exactly the function of the app is**


Hello jlkj!!
DEBUG
TrackTheField - Ensuring Excellence
January
February
March
Shot Put
The whole point of this app is line graphs - you need at least two values each month per event

```
        ), // BoxDecoration
        child: LineChart(
          LineChartData(
            borderData: FlBorderData(
              show: true,
            ), // FlBorderData
            lineBarsData: [
              LineChartBarData(
                spots: getPlotPoints(snapshot.data!),
                isCurved: true,
                barWidth: 5,
                colors: [
                  Color.fromARGB(255, 26, 22, 65),
                ],
                showingIndicators: [100, 200, 90, 10],
                dotData: FlDotData(
                  show: true,
                ), // FlDotData
              ), // LineChartBarData
            ],
          ), // LineChartData
        ), // LineChart
      ), // Container
```

^^ This graph above is an example of how the line graph is made for data provided for shot put values. As shown under the LineChartBarData(widget, in order to establish the graph's spots the getPlotPoints data-set is called, and a snapshot is taken of the data. After this, the shape of the graph, colors, and overall layout of the graph is established in order to create a graph that looks like this:



(The units on the x-axis represent the days of the month, and the units on the y-axis represent how far the implement was thrown in feet)

```
import 'package:fl_chart/fl_chart.dart';
```

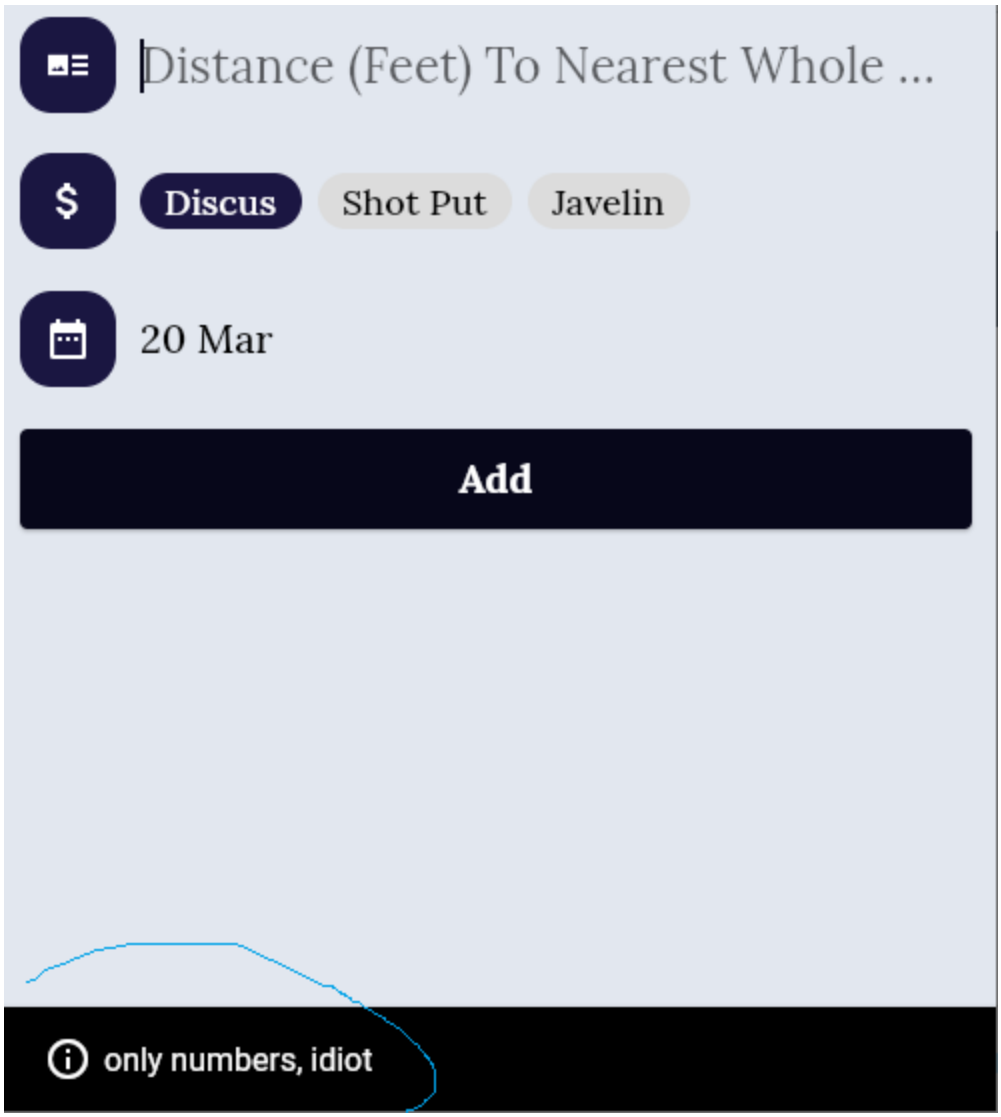
<< use of an external library. **Ingenuity: allows us to establish charts in the first place**

In order to make sure that the user inputs their data in the right format (distances in numbers only,) I used a **try-catch block** in order to warn users when there is bad input. **The ingenuity of this is that it would make sure that the value inputted fits the integer value that I declared our "amount" (distance) variable to be:**

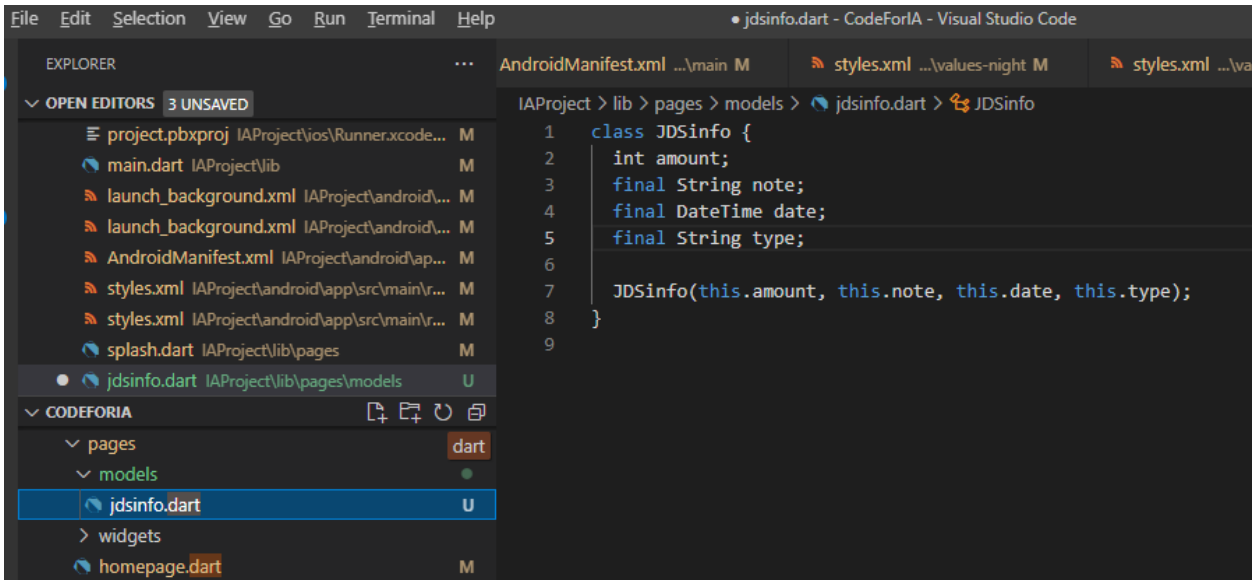
```
try {
  amount = int.parse(val);
} catch (e) {
  // show the error
  ScaffoldMessenger.of(context
```

(the try catch block attempts to parse the integer, and if it can't an error message is shown.

UI:



Use of Hive Database:



JDSinfo packages together all of the data that the user inputs, including the amount/distance thrown in the form of an **integer**, the date and time of the throw, and the event type as a **string.** **The ingenuity of this is that it allows the app to keep track of information in the other database helper class which puts the values in “boxes” (Hive Syntax):**

```dart
class JDSInfo {
  late Box box;
  late SharedPreferences preferences;

  JDSInfo() {
    openBox();
  }

  openBox() {
    box = Hive.box('money');
  }

  void addData(int amount, DateTime date, String type, String note) async {
    var value = {'amount': amount, 'date': date, 'type': type,};
    box.add(value);
  }

  Future deleteData(
    int index,
  ) async {
    await box.deleteAt(index);
  }
```

```dart
import 'package:hive_flutter/hive_flutter.dart';
```

The screenshots above and below this text both are examples of how to access the hive database within the code. DbHelper method is used in the add_name, new_value, homepage, and new_name section because those are the pages that require information from the database. (Source #1 And #5)

```dart
DbHelper dbHelper = DbHelper();
```

```dart
void main() async {
  await Hive.openBox('box');
  await Hive.initFlutter();
  runApp(const MyApp());
}
```

This set of code in the main method makes sure that the app does not start functioning unless the database is properly initialized (Source #6).

Words: 930