M. Kufleitner Winter term 2023

Practical Course: Logic App Development

Exercise sheet week 5

To be completed before 17th November 2023, 11am.

1. This week you will build a single-route flutter app around your `Dice` class from week 3. The app needs to work offline (e.g., it should not load images from the web) and it should be intuitive to use (in particular, you should try to make the interaction with the user as unambiguous as possible). Basically, the app allows the user to throw a pair of dice and show some information regarding statistics. The app should only rely on stateless and stateful widgets for the UI and not use any other state management system. Except for `get` (aka `getx`), you are free to use any other public library available on pub.dev. The app should provide the following information and functionality:

   - There should be two dice in a row which display the current throw. Each die should use around 20 to 30% of the width of the screen with around 10 to 20% space between them. The remaining space to the left and right should be distributed evenly. You can use icons of a die or a large square with a written (arabic) number inside for showing the result of a die. When the user clicks on one of the dice or anywhere near (i.e., on one of the dice or somewhere within the empty space surrounding the dice), this should trigger a new throw. The result of this throw should then be shown on the dice on the screen. The dice or the area around them should **not** be implemented as a button.

   - The number of throws since the last reset should be shown to the user.

   - There should be a switch for choosing an equal distribution for the sum of the two dice.

   - There should be a button for resetting the statistics. Before actually resetting, the user should be asked for confirmation first using a `SnackBar`.

   - There should be a button for making 1000 throws. Only the result of the last throw should be shown to the user, but the statistics needs to be updated for all throws.

   - For the next two items, you need to choose two sufficiently distinct colors. The description below will use *red* and *green*, but in your app you can choose other colors which match the requirements (and are a better fit for your design). You can also use color transitions with more than two colors, but this is up to you.

   - There should be a (vertical or horizontal) line with 11 squares without any space between the squares. Every square (in order) should correspond to one of the eleven possible outcomes of the sum of the dice. Depending on how often a sum occurred, the color of the corresponding square should gradually change from red to green. Red means that the sum did not occur at all and green means

that this is (one of) the sum(s) which occurred most often. All values in between are represented by a shade of the two colors; it should be more red if the sum occurred less frequently and more green if it occurred more frequently.

- Similar to the above visualization, there should be a 6×6 grid of squares with colors ranging from red to green for the statistics of the individual throws. Again there should be no space between the squares; red at coordinate (i, j) means no occurrence of this particular throw (i, j) and green means that this throw occurred the most often. Note that the latter property is different from naively translating the quotient `dieStatistics[i-1][j-1] / numberOfThrows` into a shade between red and green.

The layout of the app is up to you, but you should try to come up with a good one (naturally, *good* in this context has many different aspect: user experience, aesthetics, technical complexity, etc.). You might want to consult `https://material.io/` to get some inspiration.

2. The following contains a list of optional requirements. You do not have to implement these features but it certainly would add even more value to your learning experience.

   - Organize your project in different files and directories.

   - Depending on whether the app is running on Android or iOS, use a different widget for the switch (e.g., `Switch` or `CupertinoSwitch`); implement the distinction yourself rather than using an adaptive switch.

   - Implement a light and a dark mode; allow the user to change between the two using some widget (i.e., do not use the user's system setting on their phone as the sole source for this choice; not that this would be a bad approach, but it makes evaluating your app more difficult).

   - Implement two different layouts for portrait and landscape (have a look at `LayoutBuilder`).

   - If the user clicks on one of the colored squares, show the number corresponding to this square as a snackbar which automatically vanishes after a short time.

   - You can try different color presentations when mixing red and green (resp. your choice of the two colors), e.g., you can compare the classes `Color` and `HSVColor`.

   - You can use images to make your app look fancier. For locally storing images and other data in your app, see

     `https://docs.flutter.dev/development/ui/assets-and-images`

     Make sure that the copyright of the images allows you to actually use them in your app.