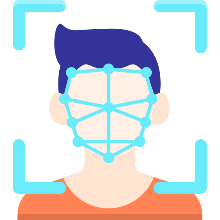
CS5381 Final Report by Jose G. Perez

**App Name:** PhotoFilters

**Description:** A camera application with added features to develop your own “Snapchat”-like filters/lenses and “Instagram”-like photo filters, save them, and share them with your friends. Uses deep learning models of face recognition and computer vision algorithms to track your face in real-time and overlay the filters, just like Snapchat.

**Requirements:** I devised an initial list of (not completely formal) requirements for this project which are listed below.

* R1: The user should be able to view, edit, and create new filters.
* R1.1: The user should be guided in a “wizard”-like manner towards anchoring images to specific face landmarks
* R1.2: The filter creation process should be easy to use and understand
* R2: The user should be able to apply the existing filters to static image
* R3: The user should be able to apply the existing filters to a live view of their camera, updated in real time
* R4: The user should be able to save the currently viewed image with the applied filter to their storage
* R5: The filter information should be stored in a database and the subsequent linked image files to the phone storage

**Development Challenges:**

* There are 3 classes named the same (Image), converting between them is not straightforward at first
  + dart:ui.Image – Dart image that contains width/height information, loading requires async
  + flutter.Image – Flutter image WIDGET that displays an image, does not contain image size information only widget size information, and can be loaded without async
  + image.Image – Image library (from pub.dev) class that allows you to perform advanced image operations
  + FirebaseVisionImage (honorable mention) – Class that is passed to the Firebase ML detectors, you cannot access any of its members only create it
* There are issues with Flutter’s camera package and flutter\_camera\_ml\_vision that crashed my app randomly from time to time
  + <https://github.com/flutter/flutter/issues/39109>
  + <https://github.com/flutter/flutter/issues/19595>
  + Had to use someone else’s patched camera package and force Flutter to use it with a dependency\_overrides section in my pubspec because it conflicted with one of my libraries that used a previous version of it
* IndexedStack loads its children only ONCE during its creation, and then never reloads them. Therefore, if you use a child multiple times you must RESET ITS STATE when opening or closing it. You also need to make sure to be careful with nulls because ALL CHILDREN ARE ALWAYS LOADED, even if they are not visible or currently in focus.
* Performing face detection on your live camera stream is NOT well documented by the firebase developers, so I had to use someone’s library to help me with it called flutter\_camera\_ml\_vision. It is not completely bug-free, it has issues, but it works most of the time and is easy to implement and use.
  + <https://github.com/rushio-consulting/flutter_camera_ml_vision/issues>

**Interesting Design/Implementation Decisions:**

* Using form\_builder instead of Flutter’s form API
  + Reasoning: To streamline the process, the library helps get rid of boilerplate code and provides a lot of functionality for easy form building as well as to implement your custom form fields more easily.
* Using a speed-dial floating action button instead of the regular one
  + Reasoning: To provide a better interface for the main actions of the application so there is not a lot of buttons covering up the image, and it looks cool when all the dial options open up
* Combining a Swiper with CircularText and an Icon for the filter list
  + This is like how it looks in some other apps like Snapchat, albeit a bit more simplistic.
* Using FeatureDiscovery to highlight what different components of the app do when the user first opens the app
  + This is like a small skippable mini tutorial that a lot of modern apps use to guide new users

**Technical Lessons:**

* You can change your flutter version easily with the “flutter version” and “flutter channel <stable/dev> command”
* If your app uses a library whose version conflicts with one of your dependencies, you can try to ignore the conflict with dependency\_overrides
* You can use JSON to easily encode and decode complex data in your DB
* Every time you re-create a GlobalKey, the widget who uses that key is completely re-built
* Search through pub.dev for packages, there is no need to re-invent the wheel for a lot of things

**Interesting Flutter Features Used**

* CustomPaint and CustomPainter were used to overlay a canvas on top of an image so we can draw shapes and figures on top of a face
* Streams and StreamBuilder to build a widget incrementally using an async\* function
  + Like FutureBuilder, but can provide “snapshots” of current progress using yield keyword instead of needing to wait until it finishes the future
* Stack and Positioned were used for a lot of layouts to have two widgets overlap each other
* SingleChildScrollView and Scrollbar were used for layouts that require scrolling if they don’t fit the screen space
* InputDecorator, GestureDetector, and FormField were used to make custom form fields not included with flutter\_form\_builder

**Flutter Libraries Used (from pub.dev):**

* camera (git://github.com/hillelcoren/plugins.git)
  + Flutter’s camera package is currently buggy and crashes the app, so I had to override the flutter implementation of camera with someone else’s from github
* firebase\_core + firebase\_ml\_vision + flutter\_camera\_ml\_vision
  + Machine learning face detection component
* feature\_discovery
  + To show a small tutorial on what some components of the app do
* flutter\_form\_builder
  + Used to streamline the form building process
* flutter\_circular\_text, flutter\_swiper, flutter\_speed\_dial, flutter\_iconpicker, page\_slider, carousel\_slider
  + Widgets used for better user interface design
* permission\_handler
  + To ask for permission when trying to save a file
* image\_gallery\_saver
  + Allows for saving of raw byte data into the phone gallery
* flutter\_launch\_icons
  + To generate the launch icons for Android and iOS
* scoped\_model, path\_provider, image\_picker, sqflite
  + For components used in the Flutterbook implementation

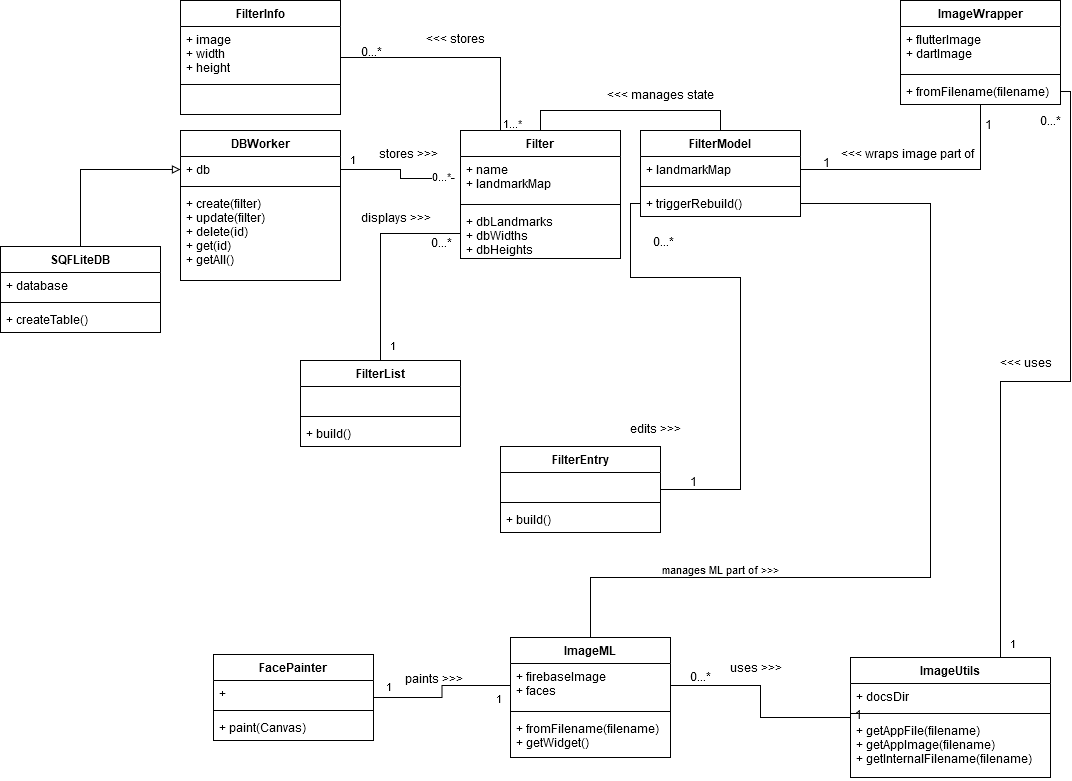
**Code Analytics:**

Generated using Android Studio’s Find function and the Statistic plugin

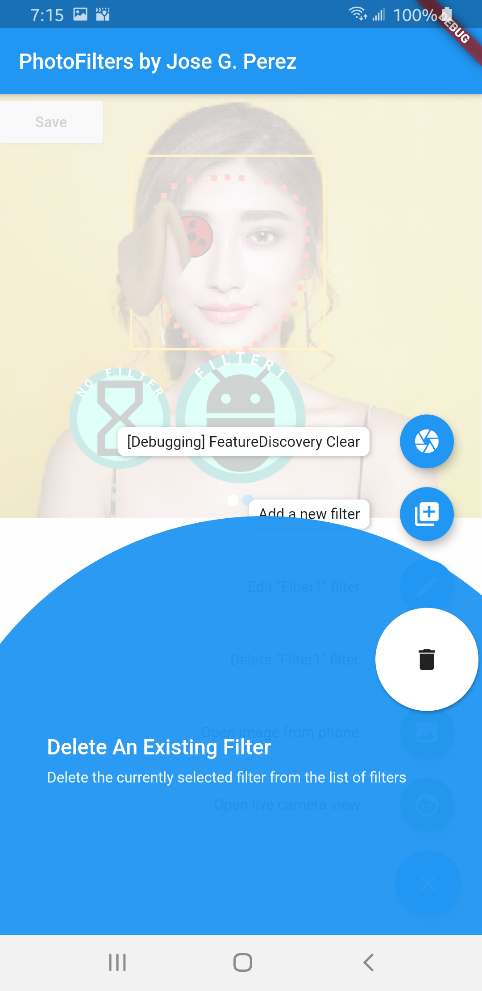
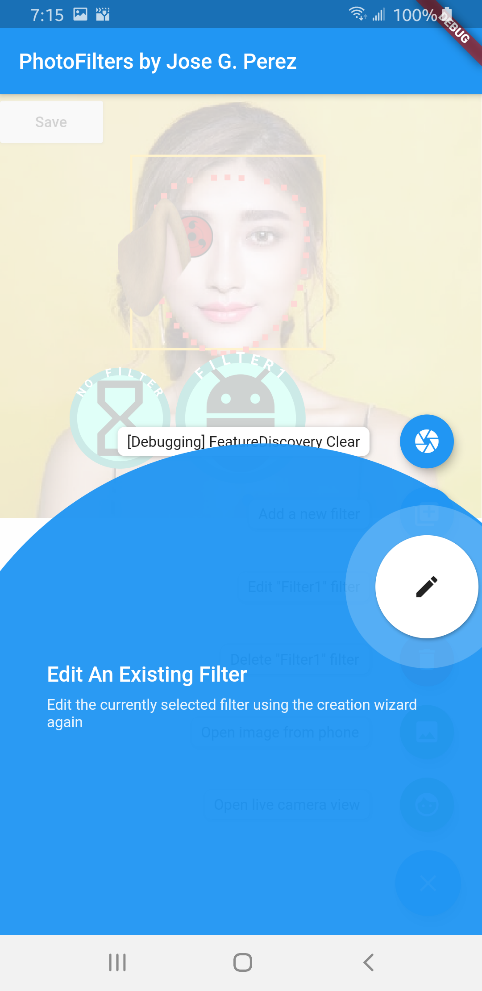
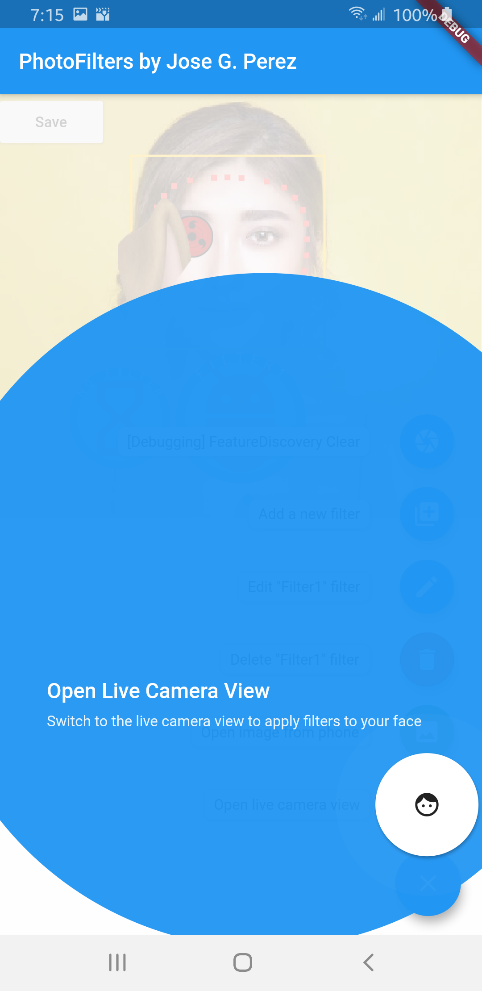
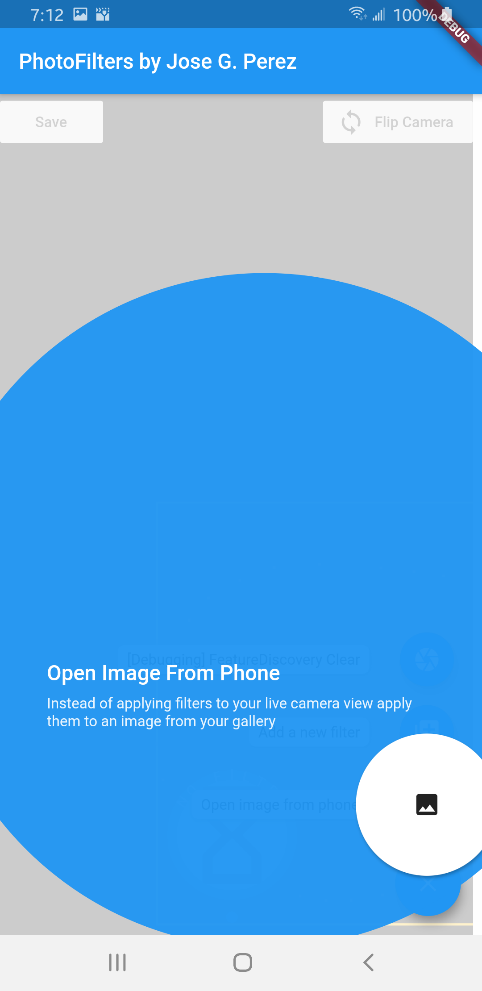
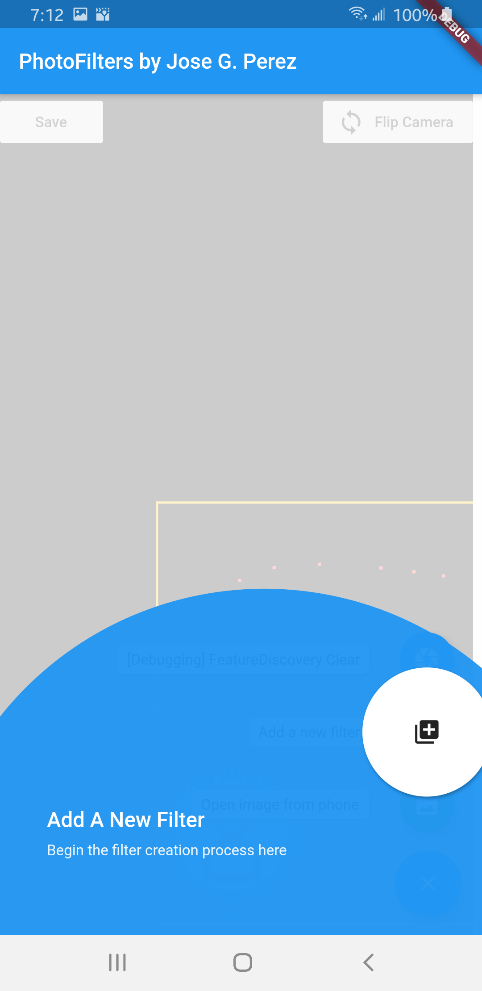
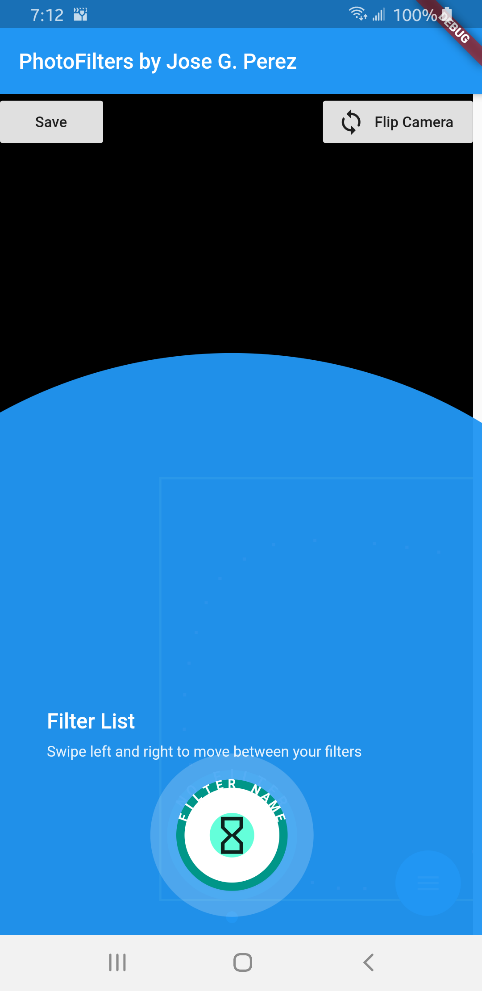
<https://plugins.jetbrains.com/plugin/4509-statistic>

|  |  |
| --- | --- |
| Number of files | **14** |
| Number of classes | **20** |
| Number of code lines | **1513** |
| Git commits  <https://github.com/DeveloperJose/Dart-Flutter-PhotoFilters>) | **30** |
| WakaTime Plugin (<https://wakatime.com/>)  Active code typing hours in the past 2 weeks | **49** |

**Appendix:**

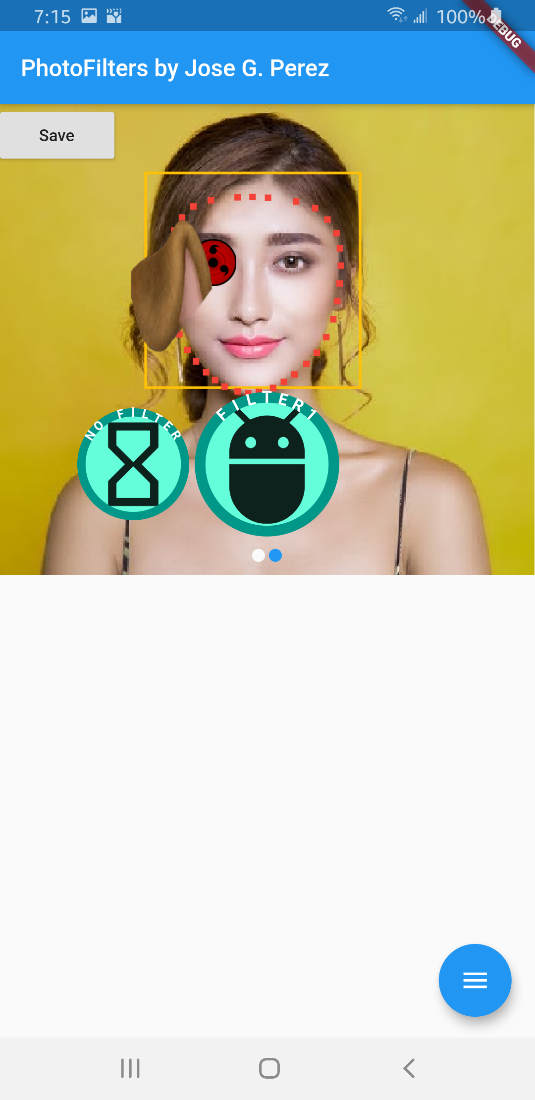
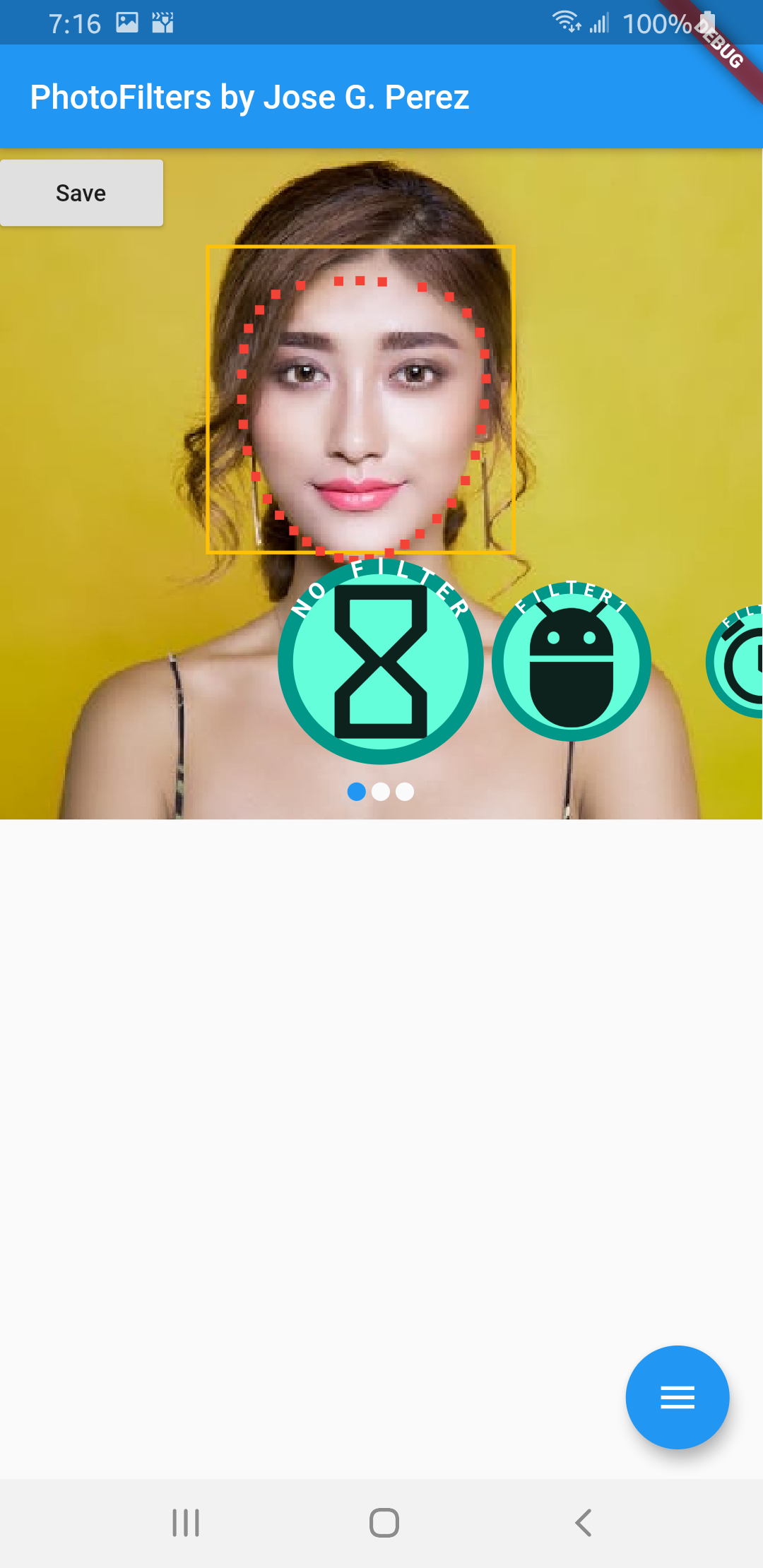


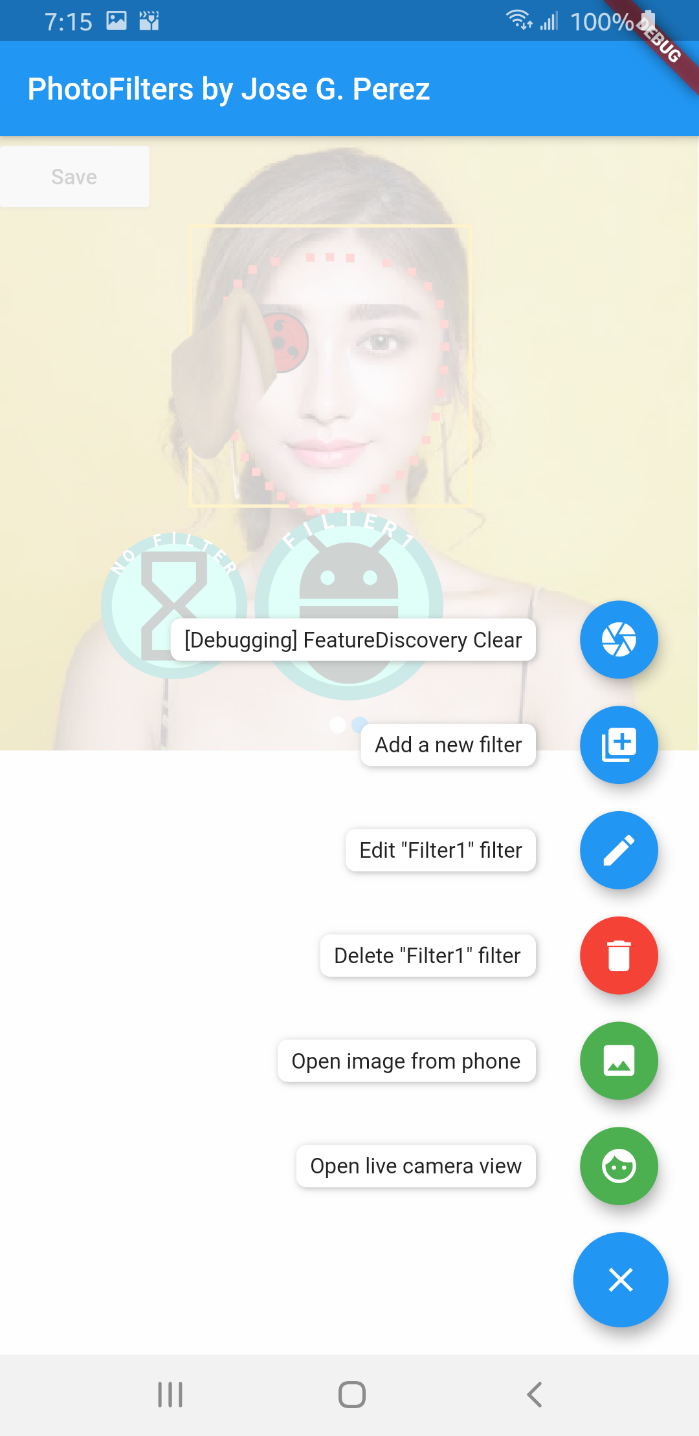
**Figure 1.** When opening the app for the first time, you are presented and highlighted the main actions of the app through FeatureDiscovery. When you tap the highlighted icon, the tutorial proceeds and explains the next feature. Alternatively, tapping outside the area dismisses the tutorial completely.

****

**Figure 2.** Typical usage of the application. The two left-most images are still images, and the right-most is a live camera view. Here you can see the effect of applying No Filter and a used created filter named “Filter 1” respectively.

Do note that the filter list is swipable, and when you swipe it immediately applies the filter to your image (or face if using live camera mode). Also note that the live camera view has a flip camera button, it is fully functional and the back camera has a higher resolution in most phones.

****

**Figure 3.** For my main screen floating action button, I opted for a speed-dial type. Shown here is the full dial with all its options.

Not shown when on live camera view

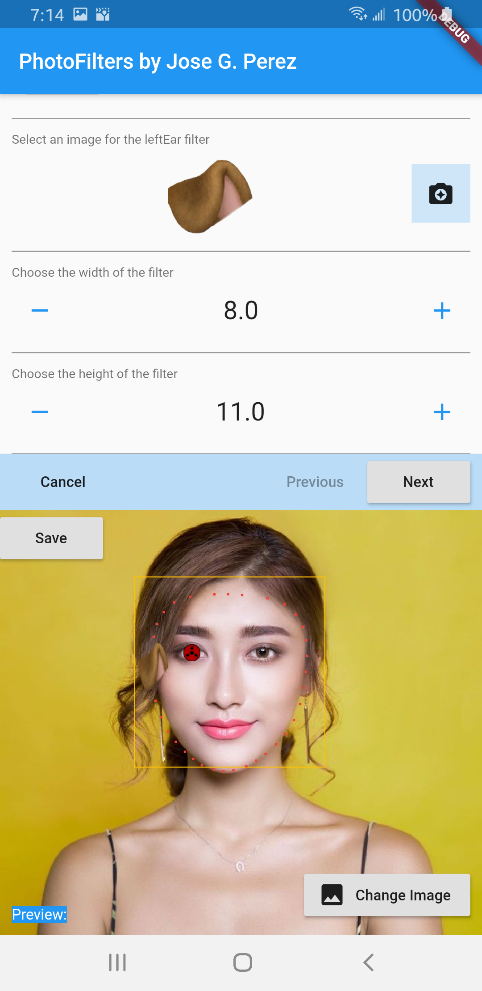
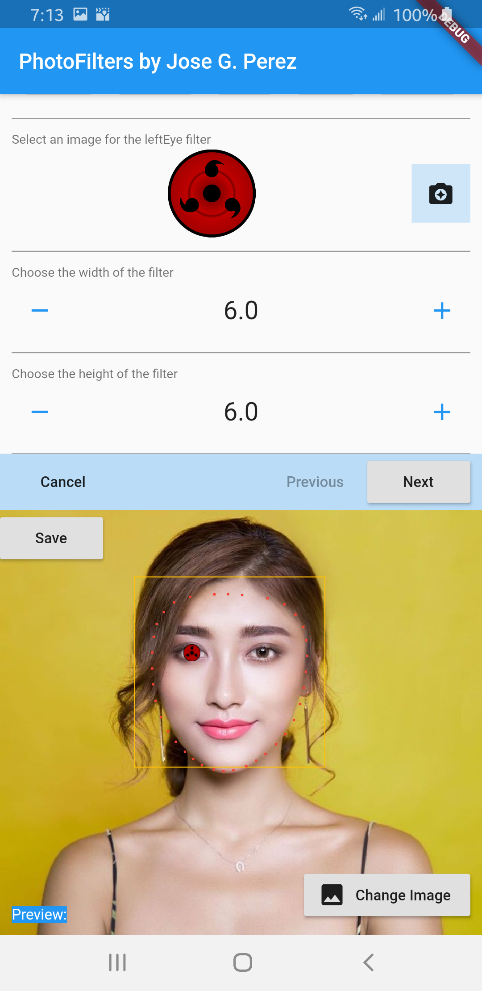
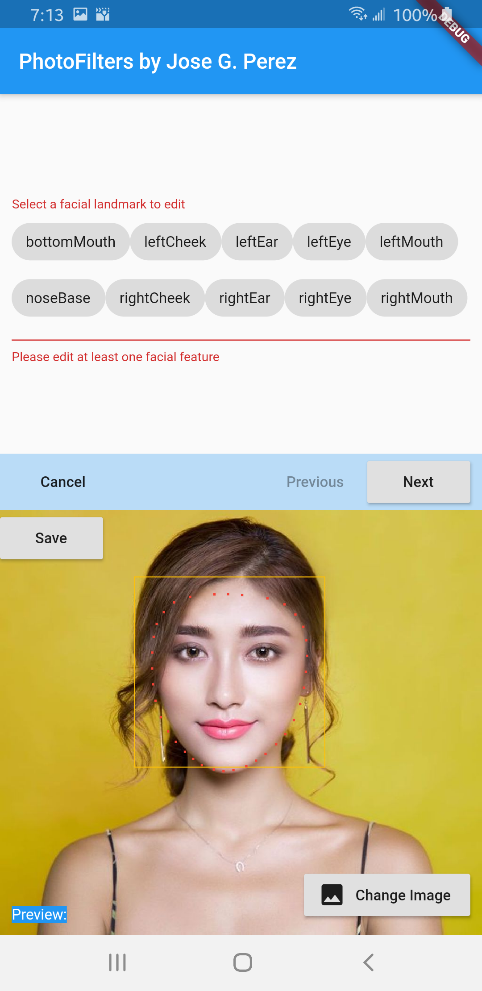
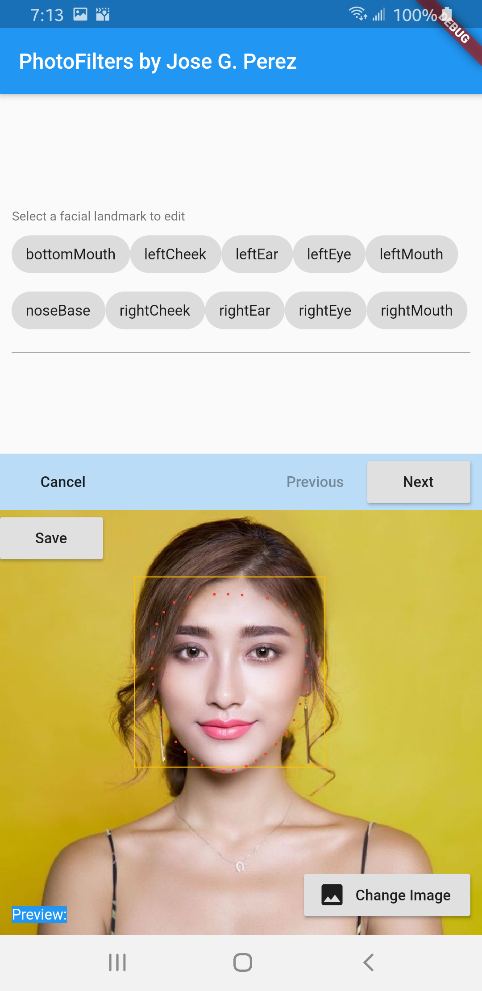
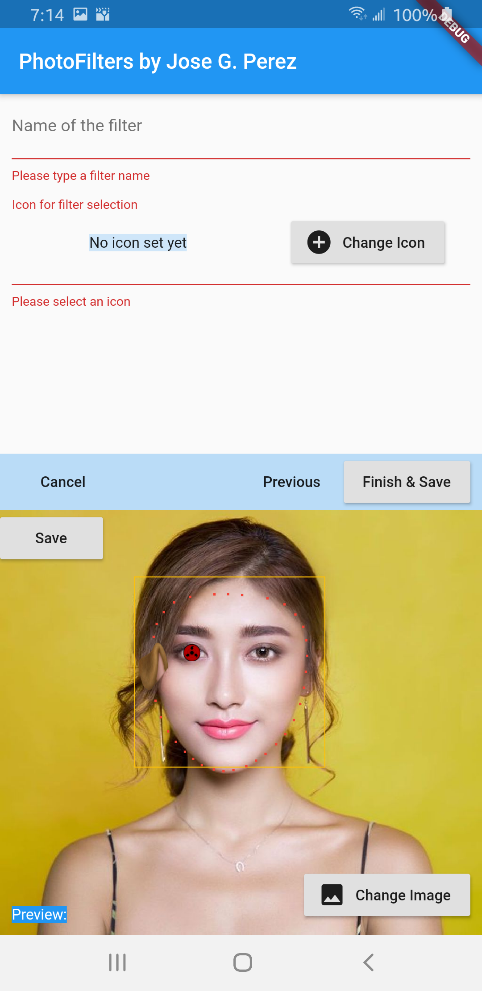
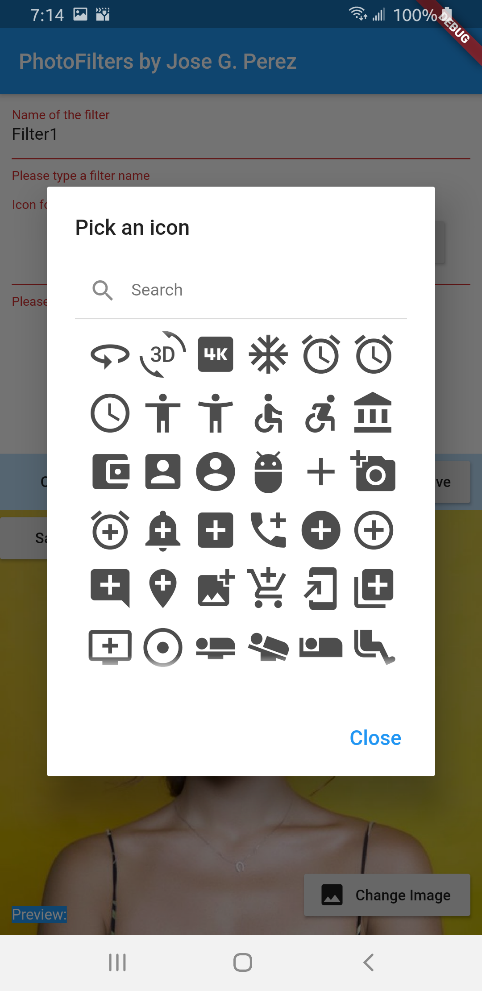
(it would be redundant)

Not shown for “No Filter”.

**Note** that the currently selected

filter name is shown

**Figure 3.** Filter creation and editing wizard. You begin by choosing a facial landmark to edit and are then presented the options. You can attach an image to a facial landmark and set the width and height (which are relative to the face width). The preview gets updated in real-time as you make the changes. Afterwards, you type the filter name and select an icon for the swipable list on the main screen. Validation is performed at every step of the wizard, as shown with the red errors.

********

**Figure 5.** When you select an image to perform processing on, it might take a while before Firebase can fully analyze and extract the faces from the image. You are updated on the progress of the loading state through a StreamBuilder as the state is streamed to the widget. This is so the app is still interactive and does not freeze while the image is being loaded. An example of one of the loading states is shown below. The states displayed to the user are (Start->Load Image -> Load Image Info -> Perform Machine Learning Recognition -> Clean-up).

