

Project Report

Smart Notebook

Introduction

I developed an Android app that aims to improve the note taking experience for students who take notes on paper. It will allow them to link digital content to their notes by associating the desired file with a number that they will write on the page, and retrieve it pointing their smartphone's camera at it, in a way similar to how QR codes work. The app uses OCR technology to recognize the number and open the appropriate file, that had been previously uploaded to the app. This way they can create smarter notes without giving up the pleasure of handwriting and flipping through a physical notebook.

My project aims to build a bridge between paper and digital world by making smartphones interact with notebooks, adding some useful features that are only available on computers and smart devices. I think that the most useful ones that are missing on regular paper notebooks compared to even the most basic word processing software on a computer are the ability to insert multimedia content and easily share the files. In this project I tried to solve the first problem, with the intention of improving the app in the future adding scanning capabilities that will allow the user to create a digital library and share the notebooks with classmates.

As a student, I have been looking for ways to make studying easier and having well-organized notes plays a big part in that. I have found that I prefer studying on handwritten notes while sketching flow charts and diagrams, but jumping from textbook to notes to lecture slides and extra resources shared by the professor kills productivity. In the past few years, taking notes on tablets with high-quality styluses like Microsoft's Surface and Apple's iPad has become increasingly popular, which is a testament to the fact that a lot of students still like handwriting but don't want to give up on some of the useful features that computers have. My app is an alternative to these expensive devices at a much lower cost, so hopefully more and more students who switched to typing notes on their computers will go back to handwriting, which according to research boosts memory and the ability to understand.

I chose to develop an Android app because I know Java but I had no experience with mobile app development, so I thought it would be an interesting challenge for my senior project, and also because I have an Android phone, so I can test the app on a physical device and not just a virtual one.

Background

I haven't found any notetaking apps that perform a similar task to mine, but I used GoodNotes as inspiration for the UI and studied some QR and barcode reader apps to decide how to implement the action of retrieving a file from some type of code on paper. The most simple solution is, in fact, using QR codes, but that is not practical for the type of problem that I was trying to solve because it requires the user to print a unique code for each file, which takes the same time as printing the actual file and putting it in the notebook.

The first idea I came up with was using different shapes like triangles, squares, circles, and other polygons in different colors that the user could draw on the page, but that kind of code

allows a very limited amount of choices. So I started looking into the performance of APIs for character recognition on handwritten text, and upon finding out that it was good I decided to use numbers that would indicate the unique identifier of the file in the database.

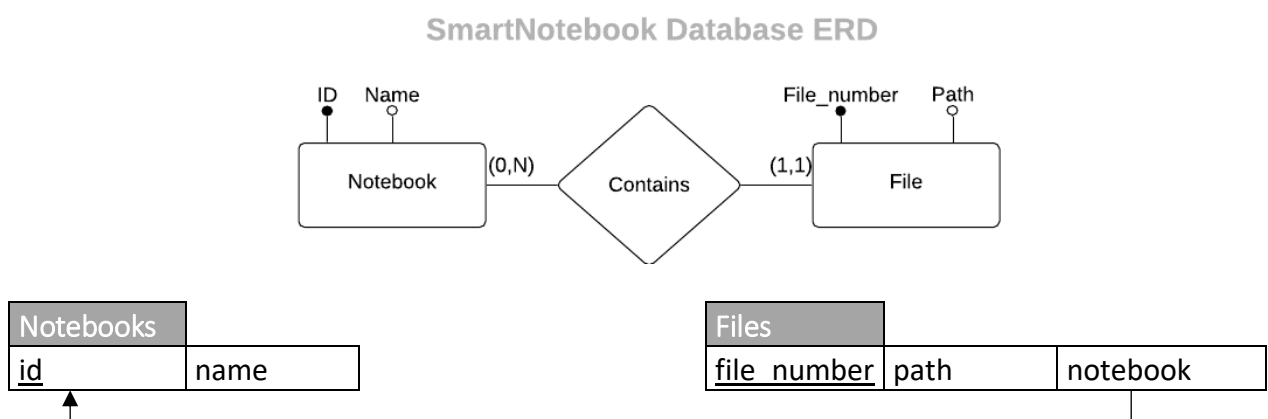
An added advantage of using the OCR approach is that then I already have the basic structure I will need to expand the functionality of my app with the scanner function, for capturing all of the text in a page and digitalizing the notes that can be stored in the app and shared with other people.

Methods

I chose to use Android Studio as IDE even if I was already familiar with Eclipse because it's Google's official IDE for Android development and Google has dropped support for Eclipse ADT a few years ago, so even if it can still be used I thought the experience would be better with Android Studio. I used the book "Android 6 for programmers: An App-Driven approach" of the Deitel Developer Series [1] to learn the basics of Android Studio and Android development, experimenting with the apps provided as examples before starting to work on my project.

Since the app has two main functions to perform, adding files to different notebooks and reading a number with the camera, I decided that the best structure for it was with two tabs, one for the list of notebooks and the other containing the camera preview, with a swipe motion to switch between them. I implemented it with a ViewPager [2], that has built-in functions that manage swipe gestures to switch between Fragments, and added buttons on the page containing the list of notebooks to allow creation, renaming, and deletion of a notebook and upload of a file connected to a notebook. The list of notebooks that is displayed in this page is made with RecyclerView, which according to Google is "the best way to display a scrolling list of elements based on large data sets (or data that frequently changes)" [8].

I had intended to use Google's Camera2 API and Firebase ML Kit for OCR, but because I hit a huge roadblock when trying to connect Camera2 with the OCR because of different and incompatible API levels required for the methods I had used I decided to switch to Camera1 and Google's Mobile Vision, which are the previous versions of those APIs and have much better documentation [3,4]. The OCR performance is very good even if the text is written on graph or ruled paper, but it won't recognize single letters or numbers because it's programmed to discard them as noise. This is a problem when the number that represents the file is a single digit, but it can be easily fixed by writing a 0 in front of it. I noticed that sometimes in this case the OCR would read O instead of 0, so I'm discarding all the letters on the string received from the OCR activity to accept it anyway. I have implemented a graphic overlay that shows the preview of all the recognized text in real time on the screen, and it's clickable so the user can select the part he needs. If it had been done differently, for example taking a picture and then processing it with OCR, it would have been problematic to distinguish between the number that indicates a file and other text on the page.



For the database I used SQLite with Room Persistence Library, which is an abstraction layer on top of SQLite that allows more fluent database access [5,6,7]. The database contains two tables, one for the notebooks, with columns for autogenerated unique ID and name of the

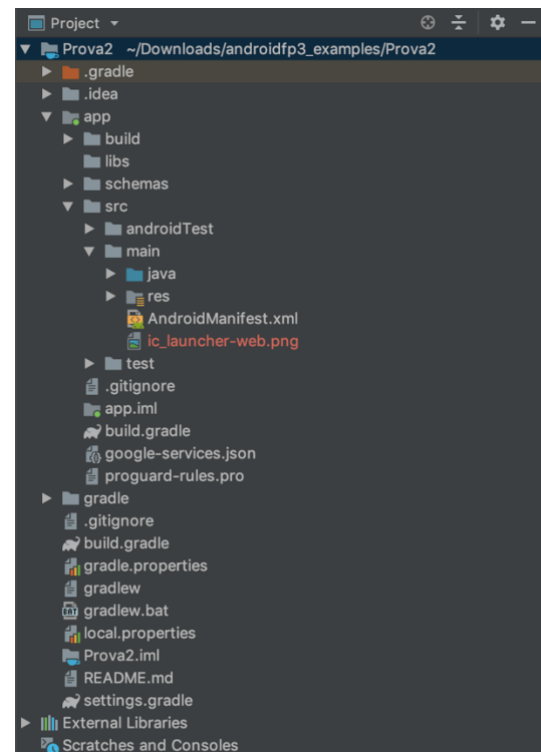
notebook, and the other for the files, that are identified by an autogenerated ID (which is the number that has to be written on the page in order to retrieve the file with OCR), have a reference to the ID of the notebook they belong to, and the file path that can be used to find them in storage. The file ID is primary key for the table, so the numbers in a given notebook do not follow a continuous sequence. I initially intended to have separate numerations for each notebook but it would have meant having to check if the scanned number existed in more than one notebook and prompting the user to choose which one they were referring to, but I decided it was more of an inconvenience for the user to have to go through one more step when trying to open the file than it is to potentially have big numbers to write.

The files are stored in Firebase Cloud Storage, at the moment I'm only supporting image formats (JPEG, PNG) [9]. When the user requests a file by scanning a number with the camera, the click on the recognized text preview triggers a query in the database to get the file path corresponding to the file with that number as ID, then the file path is used to request the picture from the cloud storage, and the byte stream received from there is passed to the ImageDisplay activity that decodes it with a Bitmap and displays it to the user in full screen mode.

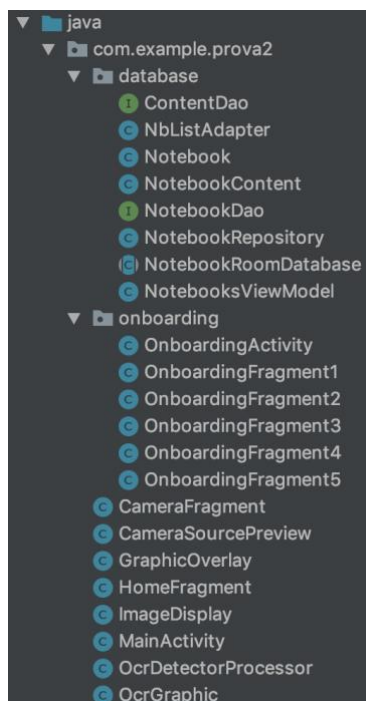
Instead of expanding the list of supported file formats I preferred to implement a walkthrough that explained how to use the app to a first-time user [10], since most of my beta testers needed me to explain it to them because it was not intuitive, and they said they would mostly use it for pictures anyway. The walkthrough is implemented using a ViewPager like in the main activity but without the tab indicators, and it is shown automatically only at the first use of the app, then it can be displayed again by clicking on the "Help" button in the menu.

Project files description and UML

There are 3 fundamentals parts in an Android project: the java source code and the XML files for the elements of the UI, that are contained in the src folder, in the java and res subfolders respectively, and the gradle files, that contain information for building the project, including versions of the APIs used, lowest Android version supported and target Android SDK, as well as a reference to the google-services.json file, that contains the necessary information to connect to the Firebase services used.



The structure of my project's source code is as follows:



- Database: contains classes Notebook and NotebookContent, which represent the objects “notebook” and “file” and at the same time contain the information necessary to autogenerate the database tables using NotebookRoomDatabase (table name, column names, primary key designation, etc.). NotebookDao and ContentDao define the possible interactions with the database by mapping database queries to method calls, that are implemented in NotebookRepository. NotebooksViewModel is a class whose role is to

provide data to the UI and survive configuration changes: it's a step in between the

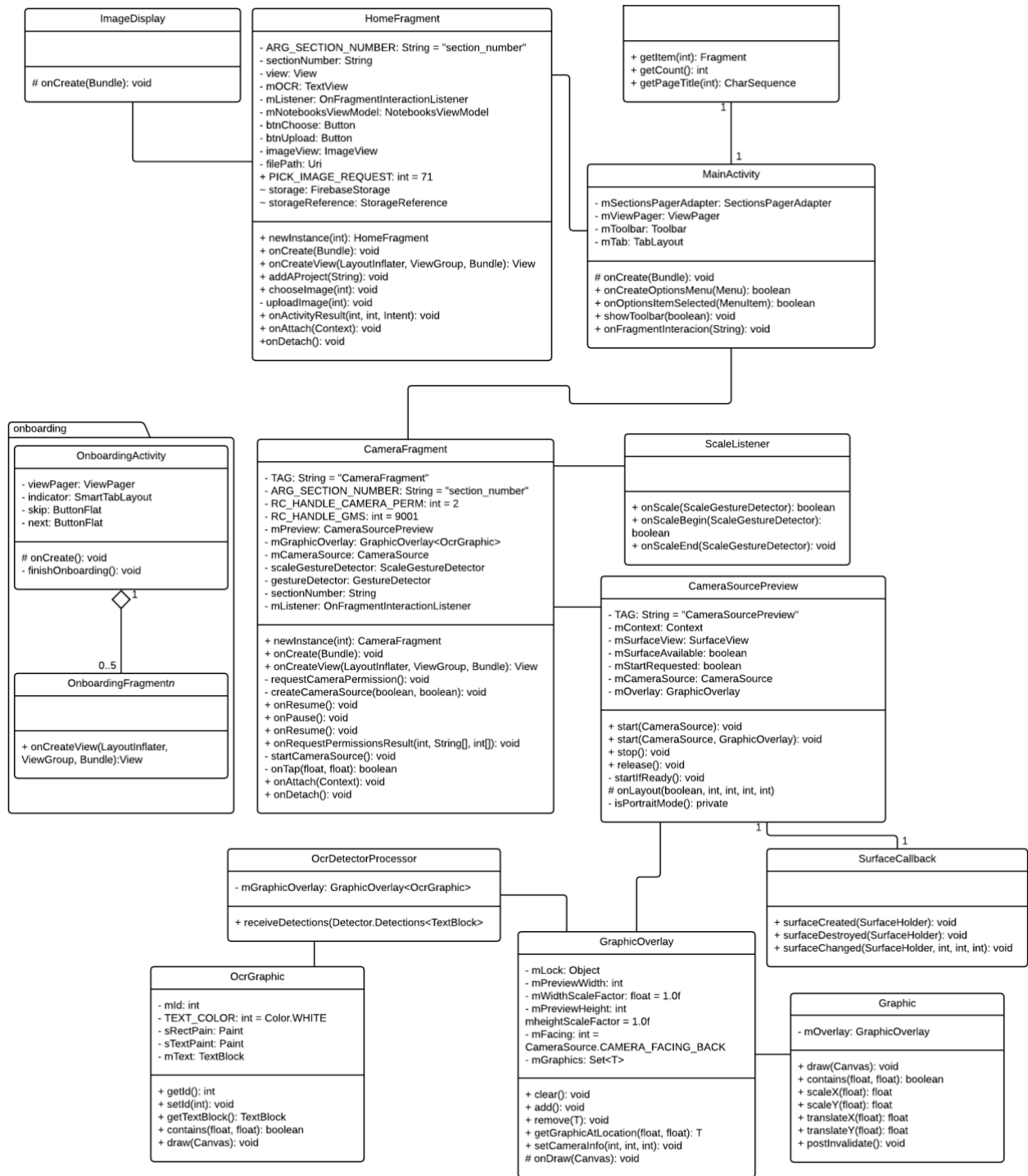
Daos and the NotebookRepository. NbListAdapter is the class that defines how a single

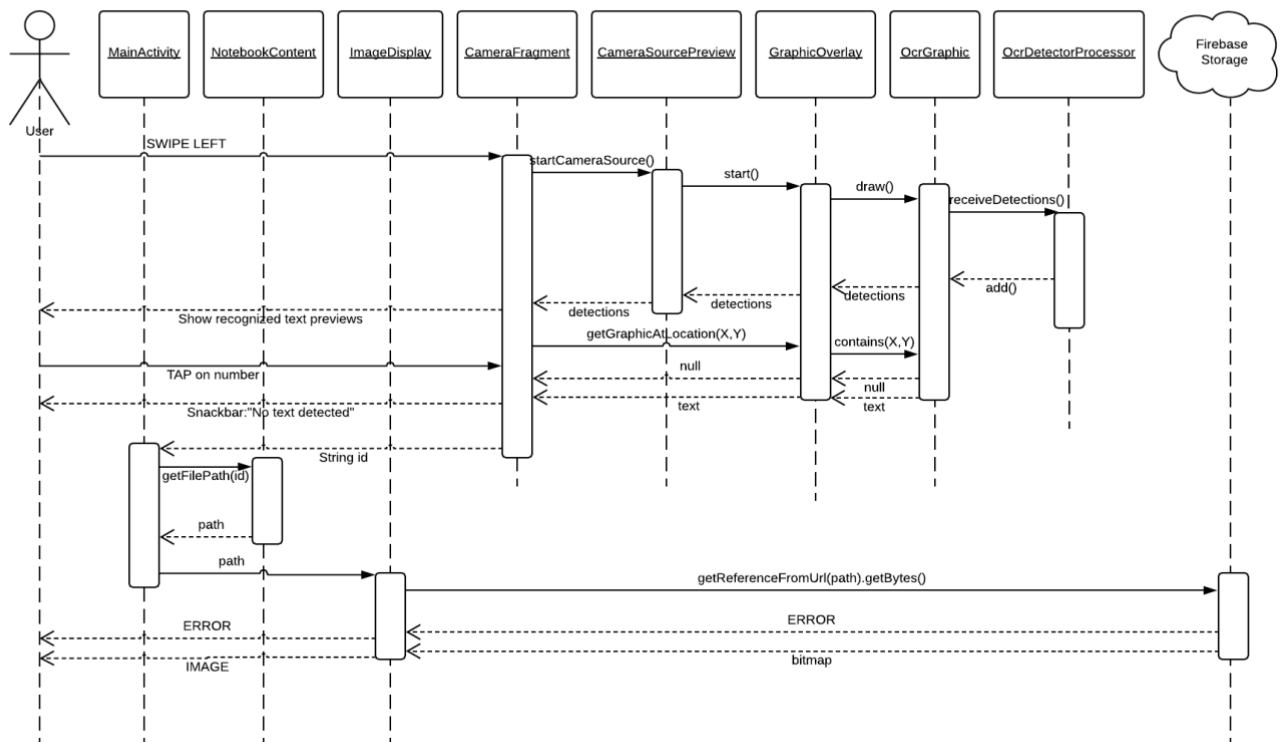
item (row) of the database will be represented graphically (calling on the XMLs for each element) and manages onClick actions on each element of the design by submitting the request to its parent HomeFragment (creating a notebook, modifying the name or deleting a notebook, inserting a file, etc.).

- Onboarding: the package contains the main onboarding activity and the 4 fragments that compose it, each of which simply inflates the layout for a page of the onboarding activity.
- HomeFragment is one of the 2 main pages of the app, the other being the fragment that hosts the camera (CameraFragment). It shows the database contents as a list, where each element (recyclerview item) is created according to how it was defined in NbListAdapter, and manages communications with the database and Firebase and interactions with the user (dialogs, pick image from storage, etc).
- CameraFragment manages user interaction with the camera, like gestures and the onTap action, which returns the text shown with GraphicOverlay tapped by the user. It shows the user the frames received by the camera (what the camera is seeing) through CameraSourcePreview, on top of which the GraphicOverlay adds the recognized text received from OcrGraphic through OcrDetectorProcessor.
- MainActivity is the basic activity that manages the entire workflow of the application. It implements a FragmentManager with HomeFragment and CameraFragment as the two sliding pages, calls OnboardingActivity the first time the app is opened or when

“help” in the menu is clicked, and it’s responsible for calling the ImageDisplay activity (that displays the image linked to the number scanned by the user) when requested by CameraFragment.

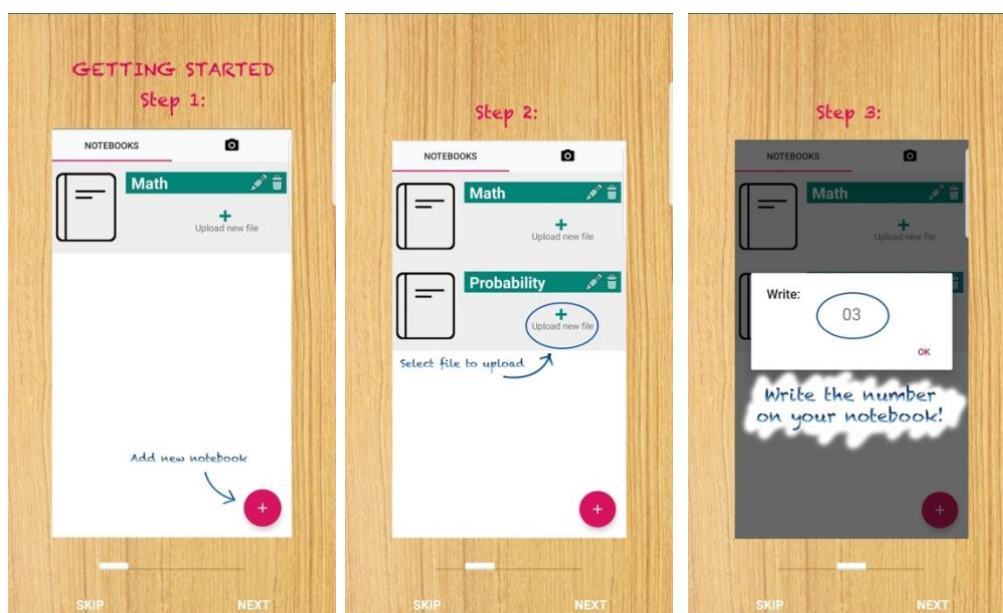
Class Diagram

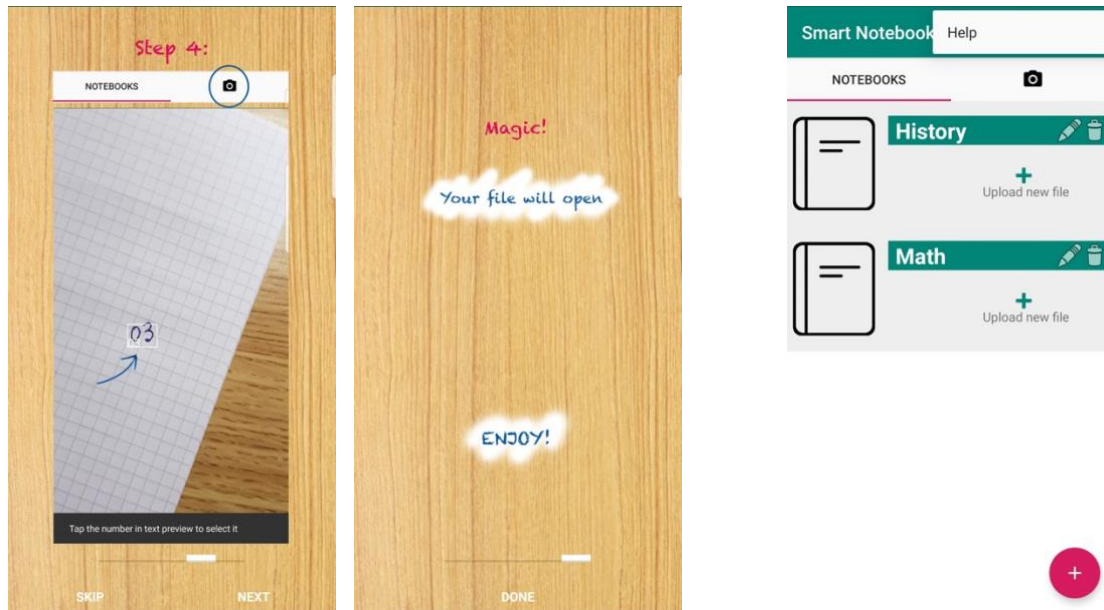




Results

The first time the app is opened it shows a guide that explains how to use it. The walkthrough is also linked to the “Help” button in the menu.





The main activity is made of two main pages, one that shows the collection of notebooks that have been created in the app, and the other is the camera with integrated OCR. The home page is the one that contains the notebooks [Figure 1] and the camera fragment can be reached by swiping left or clicking on the tab with the camera icon [Figure 2].



Figure 1

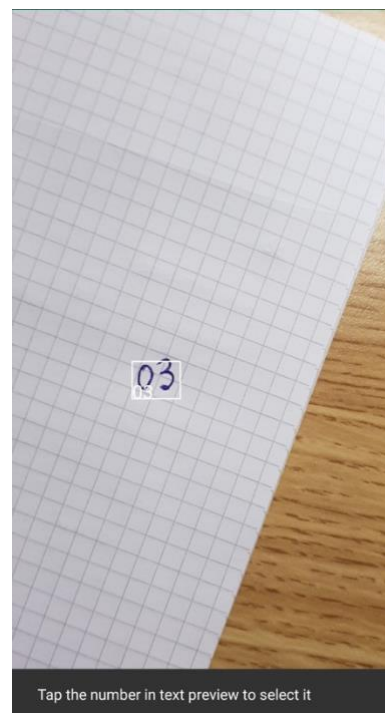


Figure 2

The Floating Action Button in the bottom-right corner is used to add a new notebook, which is created after the user writes a name for it in the dialog that opens when the button is clicked [Figure 3]. To the right of the names of the notebooks there are the buttons for renaming [Figure 4] and deleting the notebook [Figure 5].

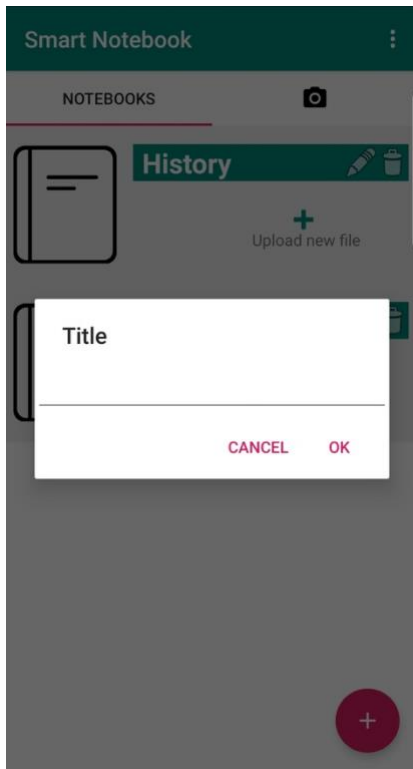


Figure 3

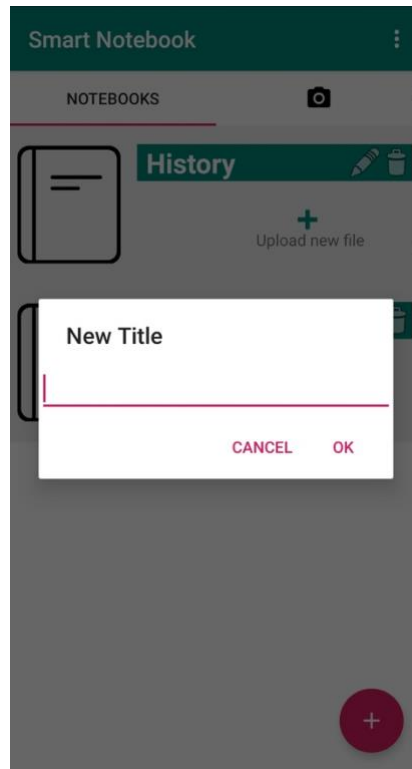


Figure 4

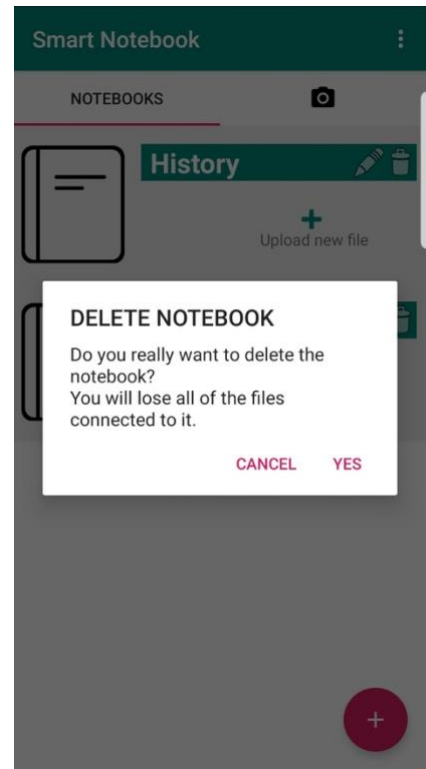


Figure 5

The “Upload new file” button opens a dialog asking the user to confirm the action [Figure 6] and if the answer is affirmative it opens the device’s file explorer so the user can select a picture [Figure 7]. After the file has been uploaded a dialog shows the unique ID that has been generated for it, which is the number that has to be written in the notebook [Figure 8].

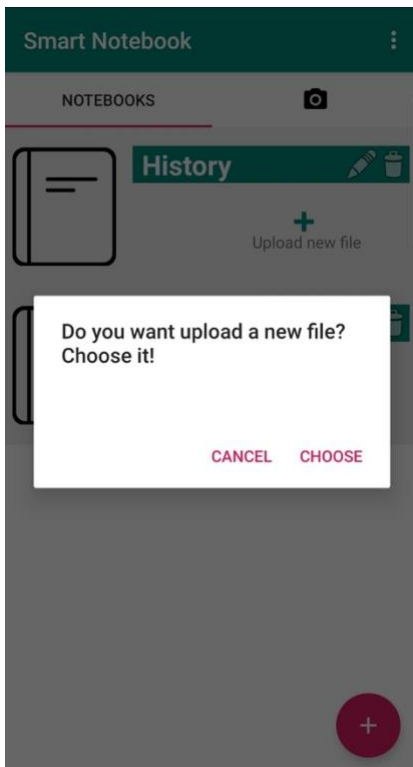


Figure 6

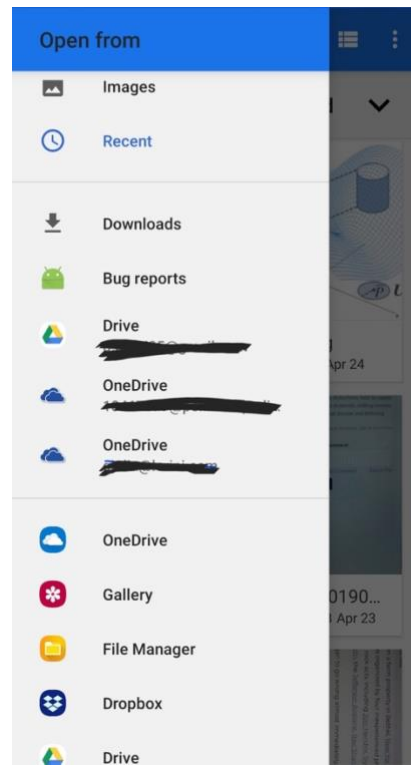


Figure 7

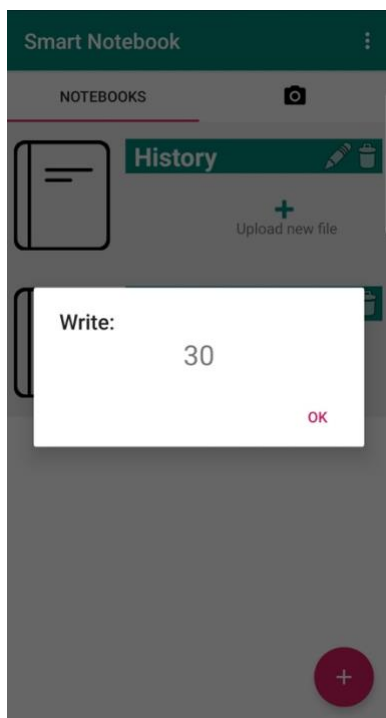
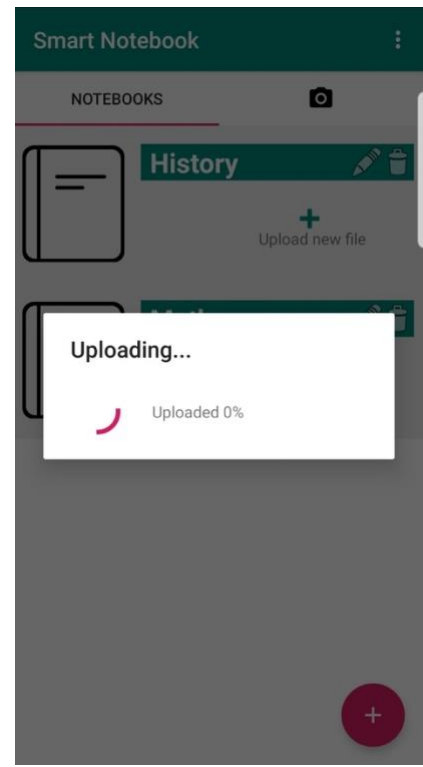


Figure 8

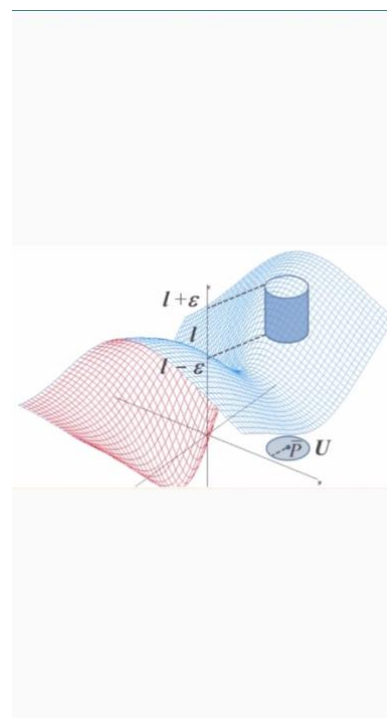
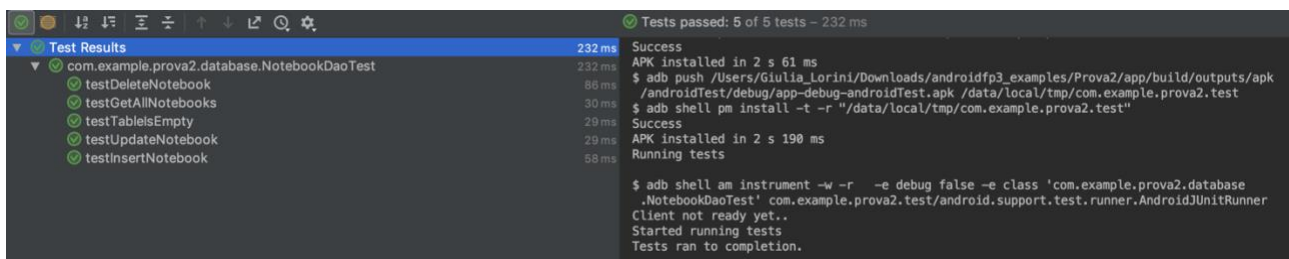


Figure 9

When the user wants to retrieve a file that they have linked to their notebook they will swipe left from the home screen to open the camera, point it at the number and click on the text preview showing the recognized number that appears on top of the camera preview [Figure 2], and the picture will open [Figure 9].

Tests

I tested the application with both white box and black box approaches: I used the JUnit4 framework, already integrated with Android Studio, to test the database, which I believe is the most critical part of the application and thus worth the time investment for such a time consuming type of test. It was particularly challenging because of the implementation of the database with Room, which adds a layer of abstraction and many autogenerated files to build the database, and on top of that there was the added difficulty of testing LiveData, which is used by Room to ensure that the data presented to the user is always up-to-date [11]. Example test result:



Black box testing was implemented both using the Firebase Test Lab offered by Firebase, which performs automated tests by randomly navigating the app [12], and with user testing performed by myself throughout the development of the app and by 6 other people, 2 “expert” users (computer science students) and 4 “regular” users, starting from the beta release. User testing was crucial because it was the only way to test OCR, which is a big part of the application. Multiple devices and API Levels were simulated with the Test Lab, I simulated a Nexus 5X API 28 with Android Studio during my initial virtual device tests, and the physical devices used were Galaxy S8, Galaxy S5, Google Pixel, Huawei P20, OnePlus 6. The final release doesn’t show any bugs, except for a problem with the camera icon on the tab in the homepage, which doesn’t render in API versions below 26, so this would suggest limiting compatibility to devices with later API versions in case of future publishing of the product on Google Play, or using text instead of the icon to label the tab. In the early stages of user testing, the users reported difficulties with understanding how

to navigate the app, which made me decide to add the walkthrough that is automatically shown upon first use of the application, and later reachable by means of the “help” button in the menu. OCR works well with any type of paper and isn’t negatively affected by grids or lines in the background, and it has good performance with numbers written in a way that most resembles the “printed” character.

Example test reports from Firebase Test Lab:

Robo test, 4/26/19, 6:07 AM ⓘ

Failed

0

Flaky

0

Passed

1

Skipped

0

Inconclusive

0

View screenshot clusters →

Test execution

Pixel, API Level 26

Locale

English (United States)

Orientation

Portrait

← Robo test, Samsung Galaxy S9+ (US), API Level 26 ⓘ

Passed

4/26/19, 6:17 AM

1 min 32 sec

Portrait

Italian (Italy)

Test results ⓘ

Robo

Logs

Screenshots

Videos

Performance

Crawl duration

1 min 26 s

Crawl stats ⓘ

Actions

61

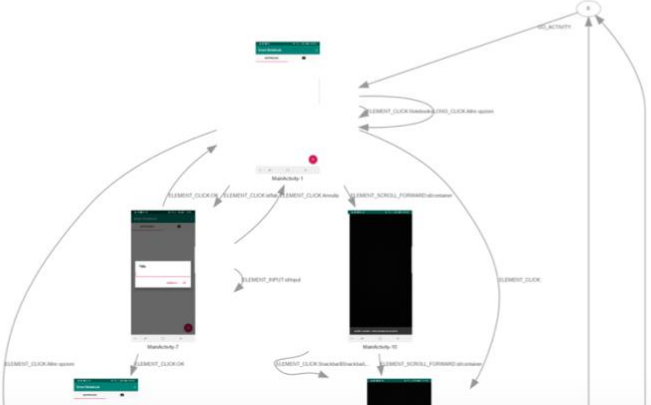
Activities

2

Screens

11

Crawl graph



Conclusion

Overall, I consider my results a success because I was able to implement all of the features I had planned to, except for the support of different media files. I don't think it would be very hard to do at this point, but I think it was a good choice to prioritize the walkthrough over adding a minor feature because if users don't understand the app they won't use it. If I had more time I would add that and implement the scanner to capture all of the text in a page and transform it into a typed document, as well as a share function to send the digitalized notes to other users. I think I will keep working on the app in the future to add these functions so I can continue to expand my knowledge of Android development pushed by the desire to improve a product that I will definitely use.

References

- [1] P.J. Deitel, Harvey M. Deitel, and Alexander Wald. 2016. *Android 6 for programmers: An App-Driven approach* / Paul Deitel, Harvey Deitel, Alexander Wald, Buenos Aires: Prentice Hall.
- [2] Anon. ViewPager with FragmentPagerAdapter. Retrieved March 10, 2019 from <https://guides.codepath.com/android/viewpager-with-fragmentpageradapter>
- [3] Google. 2018. googlesamples/android-vision. (May 2018). Retrieved March 15, 2019 from <https://github.com/googlesamples/android-vision>
- [4] Google. See and Understand Text using OCR with Mobile Vision Text API for Android. Retrieved March 15, 2019 from <https://codelabs.developers.google.com/codelabs/mobile-vision-ocr/#0>
- [5] Google. Android Room with a View - Java. Retrieved March 29, 2019 from <https://codelabs.developers.google.com/codelabs/android-room-with-a-view/#0>
- [6] Google. Defining data using Room entities | Android Developers. Retrieved March 30, 2019 from <https://developer.android.com/training/data-storage/room/defining-data>
- [7] Google. 2019. googlesamples/android-architecture-components. (March 2019). Retrieved March 30, 2019 from <https://github.com/googlesamples/android-architecture-components>
- [8] Google. Create a List with RecyclerView | Android Developers. Retrieved April 2, 2019 from <https://developer.android.com/guide/topics/ui/layout/recyclerview>

[9] Anon. How to Upload Images to Firebase from an Android App. Retrieved April 4, 2019 from <https://code.tutsplus.com/tutorials/image-upload-to-firebase-in-android-application--cms-29934>

[10] haroon47. 2018. haroon47/MySlider. (May 2018). Retrieved April 14, 2019 from <https://github.com/haroon47/MySlider>

[11] Kapil Bakshi. 2018. Testing the Un-Testable With Android Architecture Components - Room Queries. (February 2018). Retrieved April 18, 2019 from <https://proandroiddev.com/testing-the-un-testable-and-beyond-with-android-architecture-components-part-1-testing-room-4d97dec0f451>

[12] Google. Get started with Firebase Test Lab from the Firebase Console | Firebase. Retrieved April 26, 2019 from <https://firebase.google.com/docs/test-lab/android/firebase-console>