Proposal for GSoC 2019

Increase devices supported by device to device transfer app

About Me

BASIC INFORMATION

Name: Yizheng Huang
Email: huangyz0918@gmail.com
GitHub Account: huangyz0918
Website: huangyz.name
Timezone: +8:00 GMT|UTC

EDUCATION

ShanDong University, QingDao, China
Electrical Engineering & Computer Sciences, expected graduation July 2019.

INVOLVEMENT

- **ODK Collect** ***(15/134 with 18 commits)***
  - PR#2026 *(issue#1855)* • PR#2050 *(issue#1983)*
  - PR#2065 *(issue#1999)* • PR#2085 *(issue#1262)*
  - PR#2087 *(issue#1826)* • PR#2094 *(issue#1181)*
  - PR#2098 *(issue#1981)* • PR#2123 *(issue#2101)*
  - PR#2133 *(issue#2131)* • PR#2138 *(issue#2137)*
  - PR#2162 *(issue#1124)* • PR#2246 *(issue#2243)*
  - PR#2276 *(issue#2262, #2143)* • PR#2505 *(issue#2504)*
  - PR#2530 *(issue#2529)* • PR#2659 *(issue#2633)*
  - PR#2860 *(issue#2858)* • PR#2864 *(issue#2813)*
- **ODK Briefcase** ***(8/38 with 8 commits)***
  - PR#652 *(issue#651)* • PR#619 *(issue#610)*
- **ODK Skunkworks-crow** ***(8/17 with 2 commits)***
  - PR#211 *(issue#210)* • PR#229 *(issue#228)*
- **ODK Docs** ***(39/47 with one commit)***
  - PR#632 *(issue#631)*

RELATED EXPERIENCE

- Two-year Android development experience, proficient in Java; familiar with MVP structure and Git VCS, and Gradle/Maven for package dependencies management.
- Research Intern in Microsoft Research Asia (Built Microsoft Learn Chinese Android & iOS Apps & NLP models)
- An active member in ODK community, view profile.

Project

Project Name

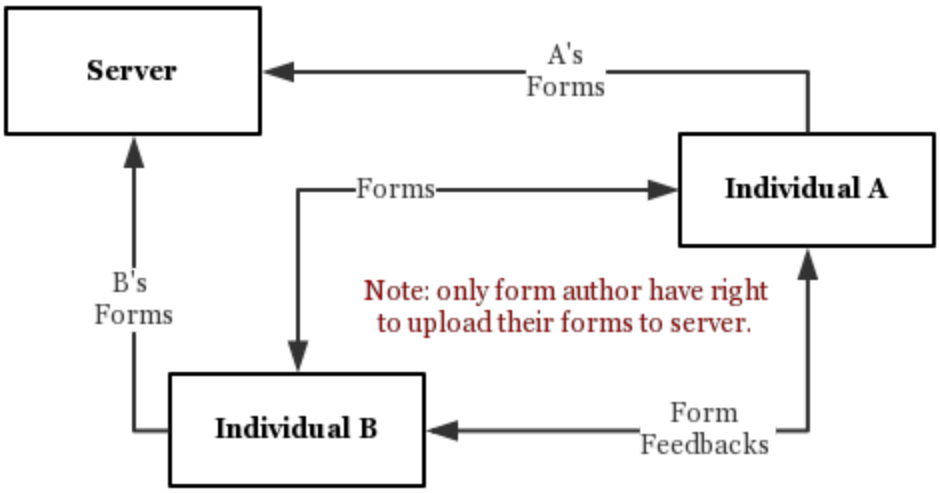
Increase devices supported by device to device transfer app

Project Description

ODK Collect aims to collect data from various fields, and offer a solution to replace paper forms with supports for images, audio clips and some other forms of data. It is designed to work without a cellular network and WiFi signal during the data collection. Once back in the network coverage, users can copy completed forms out of the device or send it to a server for analysis. And in some cases, we need a form-exchange among group members before uploading to server.

So here come some new demands, we need to transfer forms from one device to another without access to available network. Besides, we need a supervisor to collect forms from data-collectors, review them and send feedback. Now, we have skunkworks-crow, which is the GSoC 2018 project. In this year, we need to improve the project and develop a new strategy to address the highest-value limiting factors to broaden the base, and propose a strategy to validate behavior on devices we may not have access to.

Demand flow chart:



v

Outline

1. Why Current Approach
2. Limiting Factors
3. Strategy to Broaden the Base
4. Methods for Validating Behavior
5. UI/UX Improvements
6. Documents & Code Standard
7. Expected Timeline

Proposal

1. Why Current Approach

Currently, our skunkworks-crow implements no network forms transmission and it relies on WiFi hotspot technology. We compared several methods including bluetooth, wifi hotspot, NFC and WiFi P2P... And finally have a conclusion: using WiFi hotspot to transfer our data.

For those approaches listed above, their detailed advantages and disadvantages can be found here:

A. Bluethooth

Advantages:

• Bluetooth is a traditional data transferring way. Most devices have bluetooth module, so we don't have to worry about the universality if we use bluetooth to forms.

• Bluetooth APIs are easy to follow and implement.

• It's convenient for developers to create a bluetooth socket (similar to TCP socket) and handle the connection processes.

Disadvantages:

• Relatively speaking, its transmission speed is slow. (Wi-Fi Direct promises device-to-device transfer speeds of up to 250Mbps, while Bluetooth 4.0 promises speeds similar to Bluetooth 3.0 of up to 25Mbps)

• Signal coverage up to 20 meters at most.

B. WiFi Direct

Advantages:

• WiFi Direct is a software-based feature, initially called WiFi P2P, most devices have this feature.

• Faster transmission speed than bluetooth.

• WiFi Direct signal can cover a wider range (200m) than bluetooth.

• WiFi Direct supports WPA2 encryption.

• Devices can broadcast the services that they provide, which helps other devices discover suitable peers more easily.

Disadvantages:

• Require Android 4.0 (API level 14) or later devices. (WiFi P2P)

• Consume more battery than bluetooth.

C. WiFi Hotspot (Similar with **WiFi Direct**)

Advantages:

• Can run on most devices with WiFi module. (WiFiManager & ServerSocket)

• Pairing process is easier than WiFi Direct.

D. NFC*

Advantages:

• Simple operation to connect with others and transfer files.

Disadvantages:

• A number of devices don't support NFC.

• Can only transfer data within a very short distance (<0.1m).

From the methods listed above, **WiFi Transmission** is thought to be the main transfer approaches for the following reasons:

a. WiFi has a faster transfer speed than others.
b. WiFi has a wider transmission range.
c. Methods associated with WiFi in Android are easier to implement.
d. Transmission processes in WiFi are flexible and controllable, we can build some extra functions such as estimating the remaining time and calculating data transfer speed.
e. Most devices have a WiFi module, so we don't have to worry too much about the compatibility.

Compared with WiFi Direct and WiFi Hotspot, the hotspot is easier to implement than WiFi Direct. Also, WiFi Hotspot has a faster transfer speed than WiFi Direct (Tested in my Huawei Nexus 6P). So after the discussions and experiments, we decided to use **WiFi Hotspot** to transfer our forms and send feedback in our skunkworks-crow.

And we implemented the **Wifi Hotspot** method in skunkworks-crow mainly by following those processes:

a. Declare uses-permissions in **AndroidManifest.xml**.
b. Receiver turns on the **AP Hotspot**, as a Server to establish a Socket. Receiver has to wait Sender in some specified ports after that.
c. Sender connects to the Hotspot (Server) as a Client.
d. Sender selects the file to send, and submits a sending request.
e. Receiver responses and begin data transmission.
f. Check if files are complete.

2. Limiting Factors

We can simply classify these factors:

a. Hardware limitations for mobile devices.
b. Software restrictions for different Android versions and ROMs.
c. Issues caused by usage scenarios.

I will analyze each factor in detail.

2.1. Hardware Limitation

The biggest limitation from the hardware is **the support for WiFi**.
Currently, most mobile devices have hardware devices that support WiFi, so we don't have to worry about that. But for supporting WiFi Hotspot is different, some of device cannot establish an available hotspot, especially some old devices. If some device shows *'AP mode not supported'* when you trigger the hotspot, that means wireless adapter does not support Hosted Network.

Those factors can be listed like this:

1. Support for the WiFi.
2. Compatibility with wireless hotspots.

2.2. Software Limitation

The biggest limitations for the software comes from two aspects, the ROM versions and the API levels for the Android devices:

1. API Level (Android 6.0, 7.0 8.0, etc).
2. Android ROM Version (EMUI, MIUI, CM, ColorOS, etc).

The most possible factors come from Android API levels are different features or security levels designed by the Android system. It will be easy to adapt and fix according to the official documents.

We can test our application in Android Stock ROMs using devices like Google Pixel or Google Nexus (I have a Nexus 6P in API 28 which can be useful). Once the behaviors can be verified in the Android Stock ROMs, we can move forward to some third-party ROMs.

For different kinds of ROM, many limitations related to our adaptation work, that we can improve from our project aspect. Here is a simple list for most reasons causing by the ROM version:

a. ROMs don't have a support for the WiFi Hotspot (few).
b. ROMs required extra permissions to enable the WiFi Hotspot.
c. ROMs have a complex interaction logic to trigger the WiFi Hotspot (few).
d. ROMs is unstable for hotspot (need feedback).

2.3. Usage Scenario Limitation

As for the current implementation, we need to enable **access point** first. So any other access point can not be used at the same time, meaning that if user had existing connection through some access point, it will be terminated. This reason makes our transmission not very stable in the actual application process. Once there is a problem with the intermediary of different device connections, the process of transferring files will fail.

For addressing this issue, we can force user connect our hotspot if they turned our application on (by running a background service) or building another spare approach, I prefer using the second one, how to implement that will be documented in the *section #3.2*.

2.4. Some Test Results

In order to verify the expected behaviors (send & receive forms) in more devices as possible, I listed all the devices that I have tested before,

Device Name	Android Version	ROM Version	Test Result
Huawei Elate™	5.1	EMUI	Success
Hawei Mate 20 Pro	9.0	EMUI	Failed, Hotspot not turning on
Huawei Nexus 6P	6.0	Stock	Success
Huawei Nexus 6P	7.1	Stock	Success
Samsung Galaxy S10	8.0	AOKP	Failed, Hotspot not

			turning on
Google Pixel 3	9.0	CyanogenMod	Failed
Xiaomi 8	8.1	MIUI	Success
Huawei V10	9.0	EMUI	Failed, Hotspot not turning on

For those test results, I think we can use a connection listener to listener to our requests, if we get a timeout or connection failed, the application can change to the spare methods automatically to continue our data transfer.

3. Strategy to Broaden the Base

3.1. Proposal

As for the approaches related to WiFi technologies, we have already used the Wifi Hotspot (can be the best approach in our cases). So I don't think using WiFi Direct as our secondary approach is good. Also, WiFi Direct has a complex interaction process as WiFi Hotspot, the cost for our project is too much, we should not spend energy on developing a parallel function with similar properties.

For our project, I think **using Bluetooth as the secondary approach is the best**. Bluetooth is just complementary to our WiFi Hotspot technology.

The downsides for WiFi Hotspot:

- Needs Hotspot support (Bluetooth can work well on devices do not support WiFi).
- Relatively lacking in security performance (than WiFi Direct, it supports WPA2 encryption).

Let's take a look at bluetooth, it has many advantages to make up for the deficiency of WiFi Hotspot:

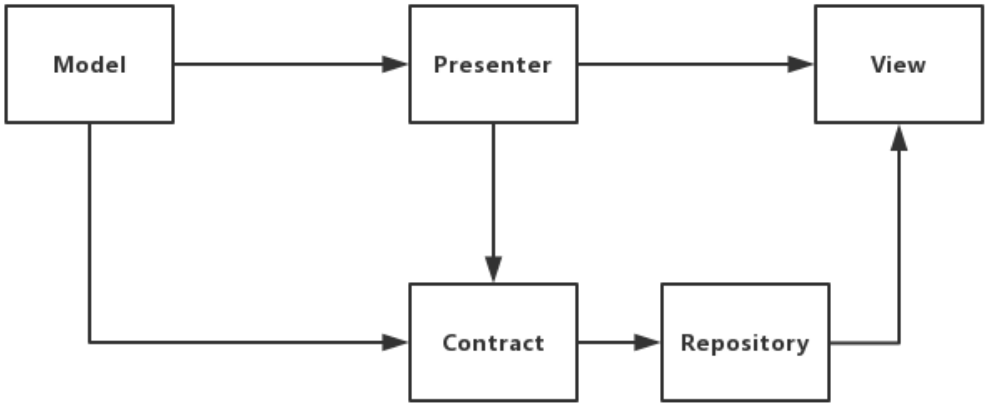
- Bluetooth is a traditional data transferring way, most devices have bluetooth module (so we don't have to worry about the universality if we use bluetooth).
- Bluetooth APIs are easy to follow and implement (Great for making the second option).
- Signal coverage up to 20 meters at most (is a downside, but it make close-range transmission will not receive interference from other wireless signals, more stable and safe).

In short, using bluetooth transmission is the best secondary choice for our project, which can get a good effect on broadening the base.

3.2. Implementation

I have a lot of experience in building android application requires the usage of bluetooth, since my major is EE, we often use bluetooth technology to create a host computer program to control microcontrollers and smart hardware.

In our project, I think **MVP structure** is suitable for the implementation of bluetooth.



According to this flow chart, we can see a classic MVP structure, The model layer offers data (from the other device) and we use a presenter to handle the received data and requests, and notify the view to update the data. since we don't have so many views, we don't really need a dependency injection framework or a MVVM structure.

The MVVM structure is not really suitable for our scenario, because the engineering benefits it brings are not as much as imagined. Similarly, some view injection frameworks and tools (such as ButterKnife) may make our view declarations simpler. However, its use is prone to unpredictable bugs in some devices and models (I have faced some issues in my previous developments). For us, actually we don't have many views in our application (most of the views are controlled by code and rendered by data), so I still recommend using android's original ***findViewById*** method to bind views. Anyway, we can discuss this when we are ready for doing the refactor work.

The part of **contract** is very useful, according to it, we can have a very clear view on our code structure. Developers can get the fastest understanding of the usefulness and structure of these codes. For example, if we have a UI structure looks like this:

```
bluetooth
├── BluetoothActivity.java
├── BluetoothContract.java
|       └── BluetoothView.class
├── BluetoothPresenter.java
└── BluetoothRepository.java
```

The **Bluetooth Contract** has an inner interface: **Bluetooth View**, which will be inherited by our **Bluetooth Activity**. So we can call the implemented method from our activity. The **Bluetooth View** gives other developers a very clear view on our functions,

```
BluetoothView
└── bindBluetoothData(ReceivedData data)
```

That method will give an encapsulated data instance to our view (extend BluetoothView) for view rendering. And the contract also has an inner presenter interface, which will be inherited in the **Bluetooth Presenter**. In the presenter, we need to send a request for getting bluetooth data (or sending data), presenter can call and pass a parameter to *BluetoothView#bindBluetoothData(*ReceivedData *data)* if we got the data successfully.

```
BluetoothPresenter
├─── senBluetoothData(ReceivedData data)
└─── receiveBluetoothData(DataRequest request)
```

In the activity, we can call *BluetoothPresenter#receiveBluetoothData()* to send a request and get data from our model layer. These designs are ideal for our bluetooth data acquisition and views rendering work, while decoupling views and logic, wrapping the APIs that get the data so that they are not directly related to the view and logic.

Similar in our view contract with WiFi Hotspot,

```
HotspotContract
└─── HotspotView
        ├─── bindReceivedData(ReceivedData data)
        └─── changeToBluetooth(DataRequest request)
```

We have an abstract method for switching transfer method to bluetooth, if we cannot send/receive data using Wifi Hotspot (or has a timeout), we will call the *HotspotView#changeToBluetooth(DataRequest request) to* send the original request using bluetooth automatically.

The repository class is used for data preprocessing. For example, we may need to do something related to data persistence and localization, or just making a wrapper of data before rendering to the views.

4. Methods for Validating Behavior

There are so many devices, we cannot test them all during our developments. So we need to build a well-performed feedback and validate system. We have three ways to do about this topic:

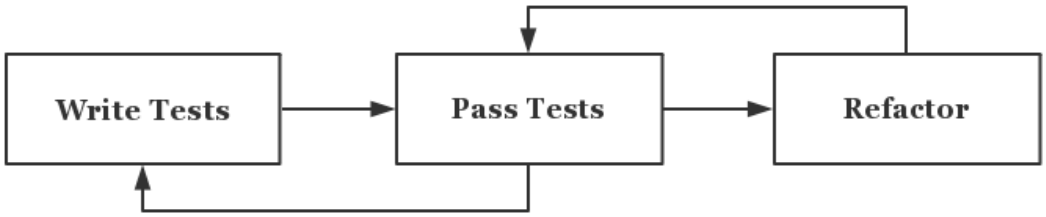
1. Unit tests (from the developer's side)
2. Feedback System (from the user's side)
3. Automatic crash reporter, Firebase Crashlytics (from the manager and developer's side)

4.1. Unit tests

Unit tests are the fundamental tests in our app testing strategy.
By creating and running unit tests against our code, we can easily verify that the logic of individual units is correct. Running unit tests after every build helps us to quickly catch and fix software regressions introduced by code changes to our implementation.

For our implementation with wifi hotspot or bluetooth, we can write unit tests to verify the life cycle of views and models. Both wifi and bluetooth have a scan result list for available devices, so we can build something like **ScanResultTest.java** to verify our behavior (The same as other functions).

In order to improve the efficiency of development and cost, I think we can use the TDD development model (Test-driven development), because our software needs to run in various environments and various devices, so the complete test will bring great benefits to our developments.



4.2. Feedback System & Firebase Crashlytics

Most of the time, we can't touch the user's point of view. If the user crashes, it is difficult for us to collect the user's specific crash information. Collecting such information can be divided into two main situations,

1. Collect feedback using some automatic tools, like Firebase Crashlytics, we can get a detailed view from Firebase Crashlytics web console, all we need to do is to copy the Firebase Crashlytics SDK into our project and trigger that in proper cases.
2. Another approach is to build a user proactive feedback system. Users can report the crashing details by filling a form, since we can use ODK Collect, I suggest using ODK Collect to collect those forms (we can offer them a crash report template).

User feedback system can also follow the structure described in *section#3.2*. I suggest we can use something like **BaseActivityWithLoadingUI.java** as a parent of our activities (need request). And put *ViewSubs* in those layout files. This design can make our life easier when we need to load a progress bar inside those views.

BaseActivityWithLoadingUI.java can follow an interface has these abstract methods:

a. onLoadSuccess()
b. onLoadFailed()
c. getRootView(View rootView)

When we need to call this method, all we need to do is to implement **BaseActivityWithLoadingUI.java** and return a root view with *ViewSubs* defined in our layout files. Details related to developments can be discussed later.

In summary, a detailed user report progress usually has those steps:

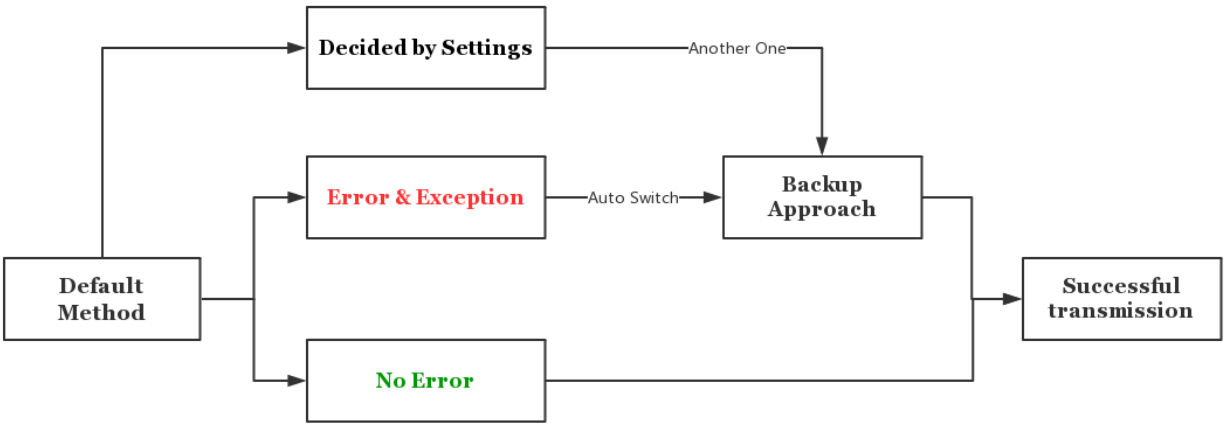
1. User launch **FeedbackActivity.java**. (or after a crash, program launch this activity automatically)
2. User fill an ODK Collect form offered by us.
3. User clicking something like *upload*, the form will be sent to our server.
4. If the user cannot reach the internet, we will store the form locally, users can choose a local form file to upload once they connected to the network.

5. UI/UX Improvements

5.1. User Interaction Process Changes

If we use bluetooth as our backup approach to transfer data, the UI/UX may need some refactor.

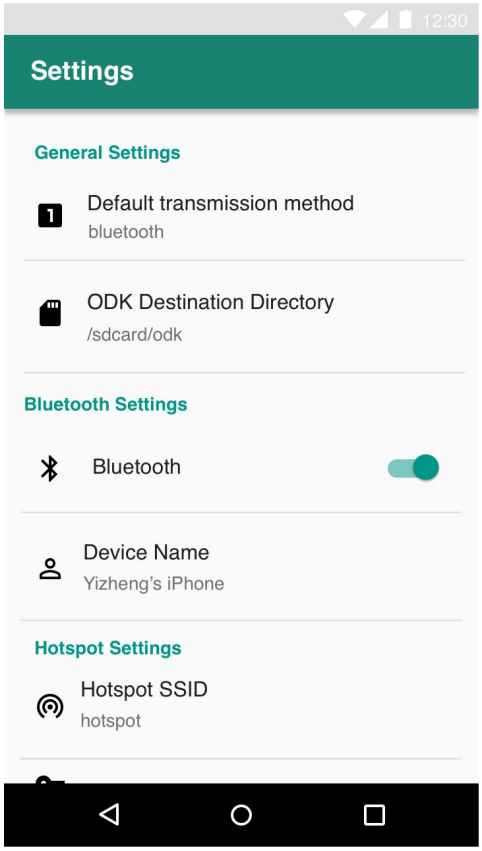
For the application process, we also have to change accordingly, WiFi Hotspot can be set as a default approach to transfer data, If the user encounters an unknown error or an exception (or timeout) while transferring data using the default mode (WIFI hotspot), our app will automatically switch the transfer method to bluetooth and jump to the new activity. All the user needs to do is to follow the instructions of the application to complete the data transfer step by step.



5.2. User Interface Changes

As some of our interaction processes have changed, our application interface has to change accordingly.

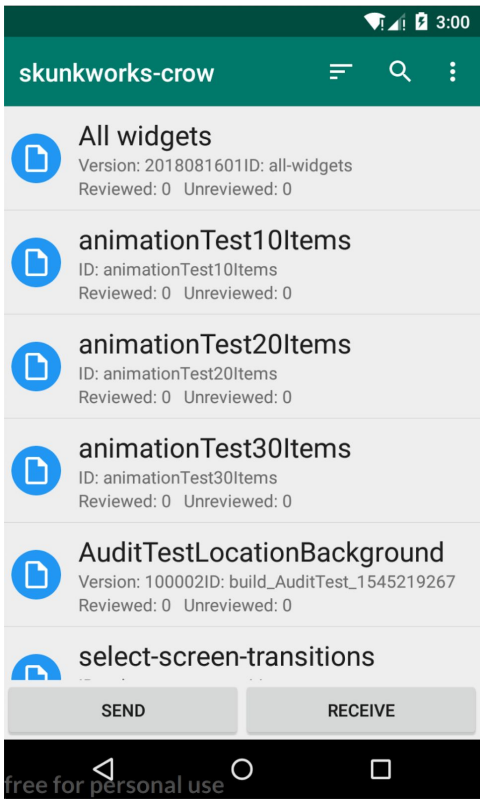
There are much work related to the UI/UX, some of them can be very time consuming, I drew some of the most important sketches.



5.2.1 Settings

The settings page needs to be optimized to isolate the different methods and add a *General Settings*.

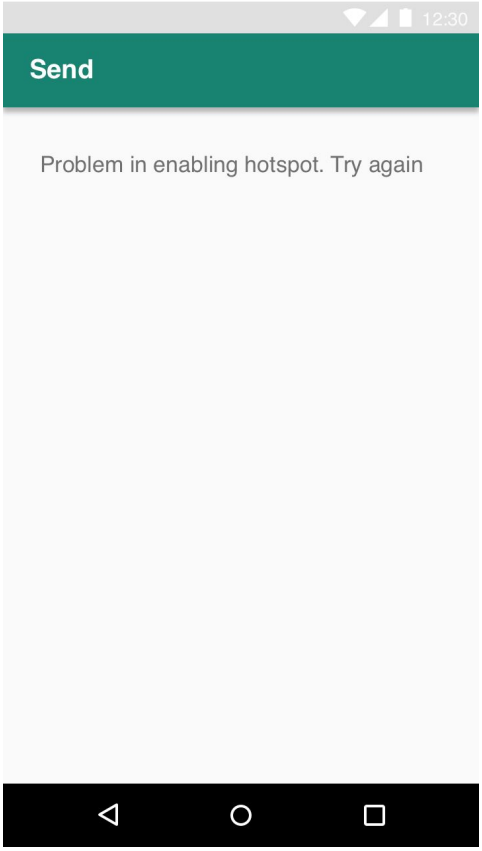
In this page, we can choose which approach is our default method, our application will offer two options, bluetooth and wifi-hotspot. user can choose one of them as the default. After that, another approach will automatically become the secondary method.



5.2.2 Main Page

Currently, when we launch our skunkworks-crow, the first thing is to choose which form we want to share with others, so I think this page does not require any improvement.

But when we click the *send/receive* in the main page, we will use the default method firstly.



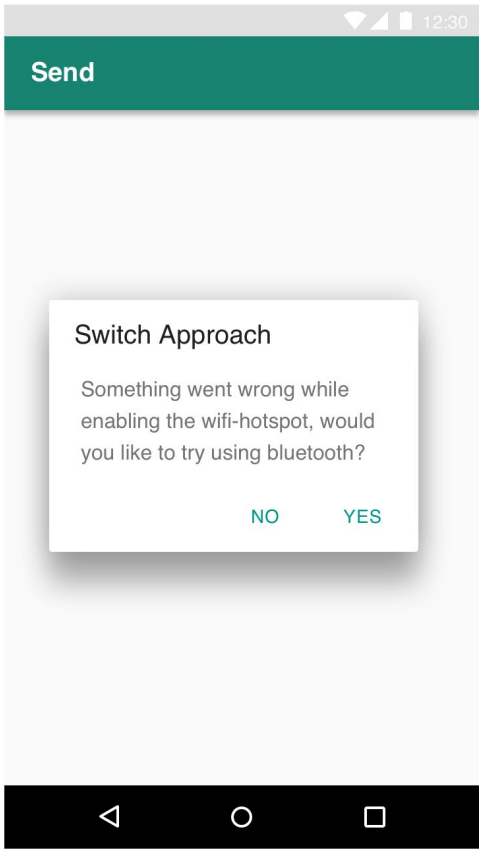
5.2.3 After the Send/Receive Failed

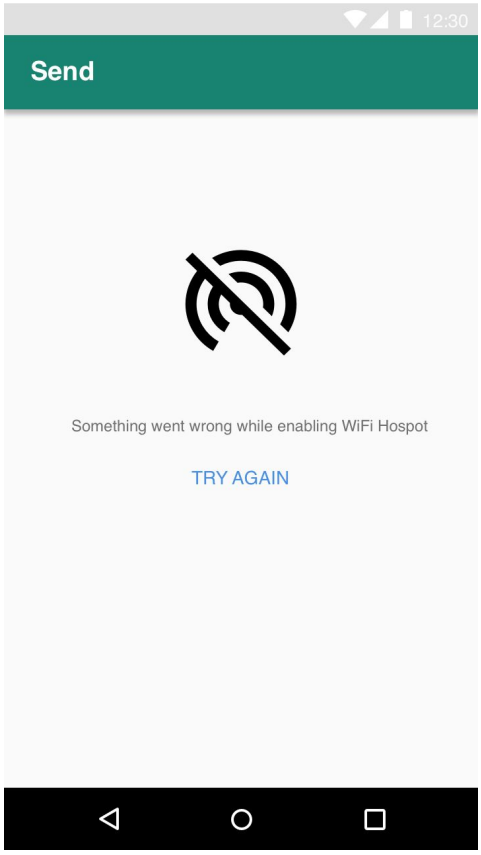
Currently, if we use the wifi-hotspot to transfer data, we will get an error message like this if we cannot establish an available hotspot connection.

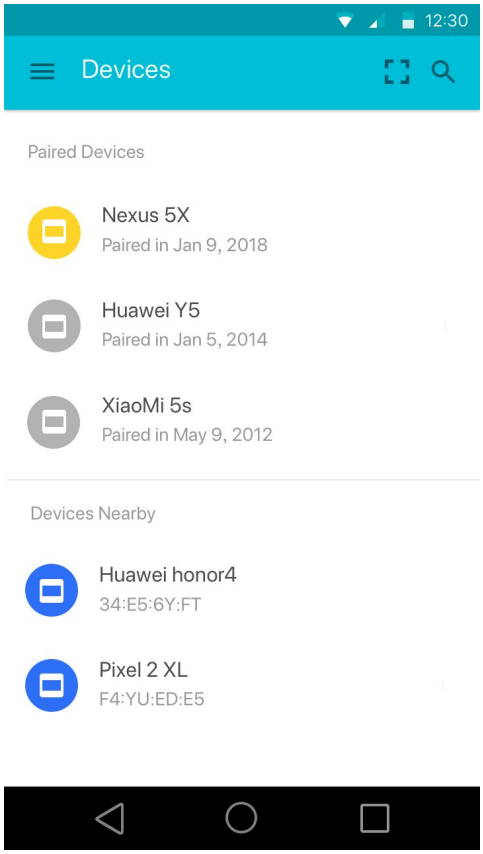
We can improve this page using a simple dialog, offering information and another choice for our user.

In the dialog, we have two buttons, *YES* and *NO*, if the user clicked *NO*, nothing will happen, the dialog will dismiss. We also need to refactor this page, add a *TRY AGAIN* button and make it more user-friendly.

Note, the *SEND* and *RECEIVE* should be refactored as two fragments under a parent activity.





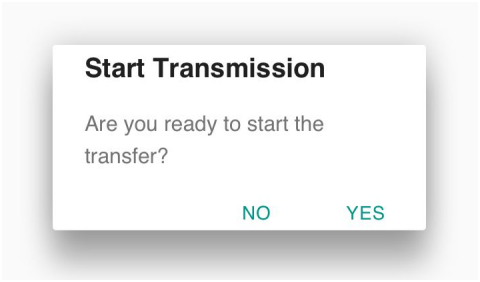


5.2.4 Bluetooth Device List

If user clicked the *YES* in the dialog (asking to switch approaches), we can launch to our bluetooth activity, there is a device list in this activity, something like this.

We can choose which device to pair with, (maybe we can add an option by scanning a QR code to pair to another device).

After successfully paired with another device, there will be a popup dialog. Asking if you are ready for a transfer.



6. Documents & Code Standard

5.1. Documents

For those tested devices, I think we should write the documents in our projects hosted in Github. Rather than using a README.md, I prefer using a **github pages**. Both of them can be combined, I mean, **we can use github README.md to generate a website hosted in Github**. I can build these since I have some experience in building web apps, it will be an easy task.

Besides, for recording the testing result and issues, we can build a mini data visualization program in our github website. These jobs seem complicated, but they are actually easy to implement. With some existing open source frameworks, we can build a better test and feedback system.

The link can be added into our ODK website too.
After the coding part is completed, a summary document will be written to describe our achievement. The document will include the introduction and some implement details about our projects.The document will be posted on my blog or wherever appropriate.

5.2. Code Standard

We need to pay more attention to code standard during the development. I have read Martin's *Clean Code* before and know the significance of a good coding style. And I have been using the Checkstyle Gradle plugin to improve the quality of my own code for over a year. Besides, we should strictly obey the development rules in open source community, for example: We should not commit unrelated changes to an issue and make sure pull requests have a clean and readable code style;

The project already has some static checks like PMD, checkStyle and findBugs, so I think we have few tasks in this part.

7. Expected Timeline

Date	Work
May 7 - 27	• Community bonding period. • Figure out the features and interaction logic inside our project. • Discuss with mentors about the implementation of the features. • Improve the GSoC proposal. • Get familiar with community rules. (e.g, how to ask questions, how to gather feedback from users.)

May 28 - June 25	• Coding before the first evaluations.
	• Refactor the code using new MVP structure (step by step).
	  • adapters (refactor to view type)
	  • structure using MVVM (should ask opinions in the community)
	• Fix bugs if has.
	• Test the previous functions (see the original features work well or not after refactoring) and write documents about this phase.
	• Build necessary documents for the project code.
	• Get down to building the bluetooth transfer feature (after the well refactor).

June 25 - 29	• First evaluations. • Write documents about the work during the first phase. • Discuss with mentors about the next steps.

June 30 - July 22	• Coding before the second evaluations.
	• Build features related to using bluetooth sending forms (the second approach to exchange data).
	• Do more tests with bluetooth sending feature.
	• Deal with forms received from others, use a presenter to handle these data and present to UI thread for a view-updating.
	• Test the sending/receiving features with bluetooth, and warp them into our model layer.
	• Finish the related UI work.
	  • UI work about the bluetooth feature.
	  • UI work about the change in previous transferring methods.
	• Build appropriate documents to present our test results.
	• Unit tests developments if time permits.

July 23 - 27	• Second evaluations. • Write documents about the work during the second phase. • Discuss with mentors about the next steps.
July 28 - August 20	• Coding before the final evaluations. • Build the review-feedback system feature. • Build the Github pages and auto publishing CI related to our tested results. • Test in different devices, address the adaptation problem. • Memory optimization and extend test cases if needed. • Improve the UI if time permits. • Test application and make it more robust. • Fix bugs.

August 20 - 27	• Students Submit Code and Final Evaluations. • Mentors review GSoC code and give feedback. • Write final documents about the work during the last phase. • Write a summary article throughout the project, and publish it to the ODK community and my personal website.

Extra Information

Working Time

I will be based in Beijing, China during this summer. Therefore, I will be working in GMT +8 timezone. I believe it will take about 10 weeks for me to complete the project. But before student projects will be announced in early May, I can have a head start with some early preparation.

As for the working time, I think it's ok for me to work 35~40 hours a week.

Reason for Participation

I dream to be a great developer since the first day I learnt programming. GSoC provides me a chance to make contributions to open source projects, with mentorship from great developers all over the world, it is really amazing. If I have the chance to participate in GSoC and work with ODK, I will try my best to complete this project.

Besides, I'm a part-time CTO in a non-profit organization which aims to improve the quality of charity in rural areas all over the world. The most important part of our work is to investigate the living conditions of rural left-behind children in poor villages, and try to help them. So I'm very excited when I saw ODK in the first time, because I think it offers a good approach to collect data and do a great help to those children. For these reasons, I would like to keep maintaining and improving ODK after the program ends. Before applying to GSoC, I'm a full-time research intern at MSR Asia, Microsoft internship experience has given me a deeper understanding of open source and project quality, and I believe that these experiences will be the basis for my better contribution to ODK.

As an Android developer, I think GSoC is a good chance to understand ODK and contribute to it. ODK has taught me a lot, and I hope I can get more from this progress in GSoC 2019. This is the only project I have applied this year, so I'm really looking forward to working on it!