# Trialchain-Patient

**About the application**

The mobile application is a wallet for the patients to store, manage and share their health records with clinics. The application also serves as a means for the patients to authorize or revoke a clinic’s access of their clinical trial data.

The application is built to run on Android smartphones, and currently can run on Android devices running Android 4.3 (Jelly Bean) and up. Running with Hyperledger Fabric as the backend, the application in conjunction with the web interface provides a robust, auditable trail of health records.

The application primarily provides the following functionality:

* Authenticate the patient onto the web interface by scanning QR codes
* The patient can add new records to the system once they have authenticated themselves on the web interface.
* Authorize a clinic to access patient’s data
* Revoke a clinic’s access to the patient data so that the clinic is no longer able to use the trial data of the patient.
* **Classes designed:**
* *AliasActivity*: An activity presented to the user to take alias input
  + Has a text field and a button. On button press, a key pair is generated and is associated with the alias. Then the alias and the key pair are stored in a SQLite database.
* *DBHelper*: A helper class that manages all transactions (read and write) with the database. This has methods for adding new key pairs, retrieving keys based on alias, deleting keys and so on.
* *RecordListAdapter*: This is the adapter class for the list view on the record list. This adapter is responsible for populating the list with the data from the SQLite database. The adapter is also responsible for instantiating the layout (initializing the text labels and filling them with alias and public keys) for each row of the *listview*.
* *Record*: A plain java class that bundles alias, public key and secret keys together. This is designed so that handling all three becomes easier. There is no functionality provided here, the class just declares data items and has getters and setters.
* *LoginActivity*: This is the launcher activity of the application. This presents the user with a text field where the user is supposed to enter their PIN. The activity, when first launched searches for existing credentials. If no stored PIN is found, the user is redirected to a register activity, where they can set up their PIN.
* *Dashboard*: This activity is the primary dashboard activity of the patient application. This activity encompasses a pager view that puts up two fragments. First one as a record list and second as a messaging interface. This activity is largely a navigation controller, the data is controlled by the fragments.
* *RecordListFragment*: This is a fragment class for the listview. This fragment holds the floating + button for adding new records. Hitting the + button brings up a QR code scanner. Now the patient app scans a QR code that has data such as clinic ID embedded in it. Then the patient gives a name (like clinic name) to the record and hits save to add a new record successfully.
* *ResponseHandler:* A class that implements AsyncHttpResponseHandler, an interface that has *onSuccess* and *onFailure* call back methods. Which are called by the application when the HTTP request is successful and unsuccessful respectively.
* *TimestampHelper*: A helper class that has class level methods to provide timestamp in ISO 8601 format.
* *QRCodeDisplayActivity*: This is invoked when the user selects an item from the *listview*. The select item’s public key is then passed to this activity as an argument and a QR Code is prepared for it. Lastly, an *imageview* on this activity displays the generated bit map.
* *QRCodeScannerActivity*: This brings up the camera to scan a QR Code. This activity reads a QR Code and just makes a toast of it at the moment.
* *RegisterActivity*: So this is the activity where the user can setup their PIN on first run of the application. It writes the PIN to a preferences file, and is stored as a key value pair.
* *Token*: This is the token that will be signed and posted to the URL embedding in the QR code on the web interface. Again, it just comprises of a public key and a timestamp, and getters and setters for it. Once the web UI has a QR code interface up, I may modify the class as needed.
* *TokenPOSTAsyncTask*: A background service that is responsible for preparing and sending a HTTP POST request from the patient application to the URL scanned from the QR code.
* **Functionalities**:
  + Running the application: We need to authenticate the user locally first.

There are two possibilities:

* + - Running first time and has no existing PIN : You’ll need to enter your name and a PIN. The PIN is essentially a number of your choice that you’ll be using to login to the application each time you run it.
    - Running the application with existing PIN on device: If you are already using the application, you probably have a PIN setup. The Login Activity expect`s you to enter your numeric PIN and hit the login button.

The Login Activity’s *onCreate* method checks if there exists a PIN. (The PIN is stored in a preferences file, which is essentially a XML file for storing key value pairs). If a PIN is not found, it automatically redirects the user to a Register Activity to enter a name and a PIN. The register activity then generates a new XML file storing the credentials)

No communication with the back end happens because the authentication is happening locally.

* + Manage your record list: After a successful login attempt, you will be presented with a list of your records in the system. You may select any one of them to bring up a QR code interface. This interface must be presented to the clinic to share your record data. Press and hold the record entry for extended duration (about 3 seconds) to delete the record from the system.

The list view has a onLongClickListener to delete the corresponding record from the application.

* + Adding new record: To add a new record, simply hit the + button on the lower right corner of the screen that holds the list of records. You will be presented with a camera interface to scan a QR code. Now, scan a QR code from your clinic’s display to obtain clinic’s ID. You will now be presented a dialog to enter a name. Please enter a name here that you wish to use to identify your record with. Hit the save button to successfully add a record. You will see the name of this new record in the list.
  + Storing the records: The records you add are stored in a SQLite Database, managed by the application.

A helper class (DBHelper.java) is in place for controlling all transactions with the database. This class is instantiated for adding keys to the database (there exists just one table called keys). The class is initialized as follows:

*DBHelper <obj name> = new DBHelper(<context>)*

The context may take a reference of the activity where this is being initialized. To get a list of all key pairs, simply call *<helper-object>.getAllKeyPairs()* . This will return a *List<Record>* object, containing all Key Pairs and their associated alias in the database. The keys are residing as plain strings at present.

* + Sharing your record: To share your record data with a clinic, simply select the record item you would like to share with the clinic from the list. You should see a new screen bearing a QR Code. Have the clinic scan this QR Code to share your record.

A library called *Zxing* is being used for generating a QR Code. It involves generating a bitmap and setting it to an *ImageView*. The library provides a class called *Bitmatrix*. This is a matrix that holds binary values of each pixel of the QR code(set or not). To generate this matrix, we use another class called *QRCodeWriter.* This class has a method called encode that takes arguments as a string to be encoded, format(barcode or QR code) and resolution of the image. Having obtained this we are ready to inflate this bitmap to the imageview.

* + Authenticate yourself on the web client: To authenticate yourself on the web client, you need to scan a QR code from the web interface. For that, scan a QR code from the web interface. To bring up the scanner, select the record you want to authenticate yourself with and you will see a button saying “Scan QR Code” just below the QR code. Once you see the camera interface, scan a QR code from the web page to authenticate yourself onto the server.

The *Zxing* library is used for scanning QR codes. Initialize an *IntentIntegrator* object to be able to scan. Starting android M and up, permission to use camera must be granted at run time. To initiate a scan, call the method of the *IntentIntegrator* class names *initiateScan().* The control is then redirected to a method of the activity life cycle method named *onActivityResult*. Here collect the contents of the QR code just scanned in an *IntentResult* object. Retrieve it as

*IntentResult qrScan = IntentIntegrator.parseActivityResult(requestCode,resultCode,data);*

Where *requestCode*, *resultCode* and data are parameters of the *onActivityResult*. The *onActivityResult* is an overridden method of the android activity lifecycle and the *zxing* library method, that is redirecting us to this method sets them. Obtain the scanned content as

*Object scanned-content = qrScan.getContents().*

Patient app update: (04/28): The application now supports live messaging. It is now capable of:

1. Receiving messages from clinics even in background.
2. Respond to these messages either directly from notification tray. \*

Messages appear as push notifications in the notifications tray on your phone. To respond to messages, select the notification to bring up the dialog box. The dialog box will provide an interface( a text field, and buttons like authorization and dismiss) to respond to the messages.

The kind of messages we intend to handle via live messaging are mainly authorization requests. As the patients log in using the QR code, the application listens for messages for the duration of their session. Once the session expires, the patient needs to log in again before they can get any new messages.

Messaging is accomplished using the following classes in the application:

1. MessageListenService: This is a background service, that runs even when the patient app is put away or the phone is locked and not being used actively. This service initiates  a subscriber using ZeroMQ API and listens for messages from a publisher. The publisher is a component of web-ui and it is responsible for sending messages to the subscriber. This service broadcasts an intent whenever it gets a new message. A message recevier(see below for more info) recevies the intent and triggers a push notification to notify the user. The foreground activity is responsible for starting this service in its lifecycle. At the time of writing this document, the service connects as a subscriber to the publisher as:
   1. `subscriber.connect("tcp://trial-chain.ml:30303");
2. MessageBroadcastRecevier: This recevier is responsible for generating push notification for the user upon receipt of a broadcasted intent. The service listens for messages, and upon receiving a message, sends an intent. The foreground activity, is responsible for registering the recevier so that it can be called when an intent is received. The notification can be used to reply/respond to the incoming messages. Selecting the notification brings up a dialog that prompts the user for their response. The user may respond to messages like authorization requests via this dialog box.
3. IntentAction: This is a class that holds custom Actions for intents. Currently only one action is defined.