telemesh Documentation

Release 0.0.1

W3 Engineers

Nov 09, 2020

Contents

CHAPTER 1

Background

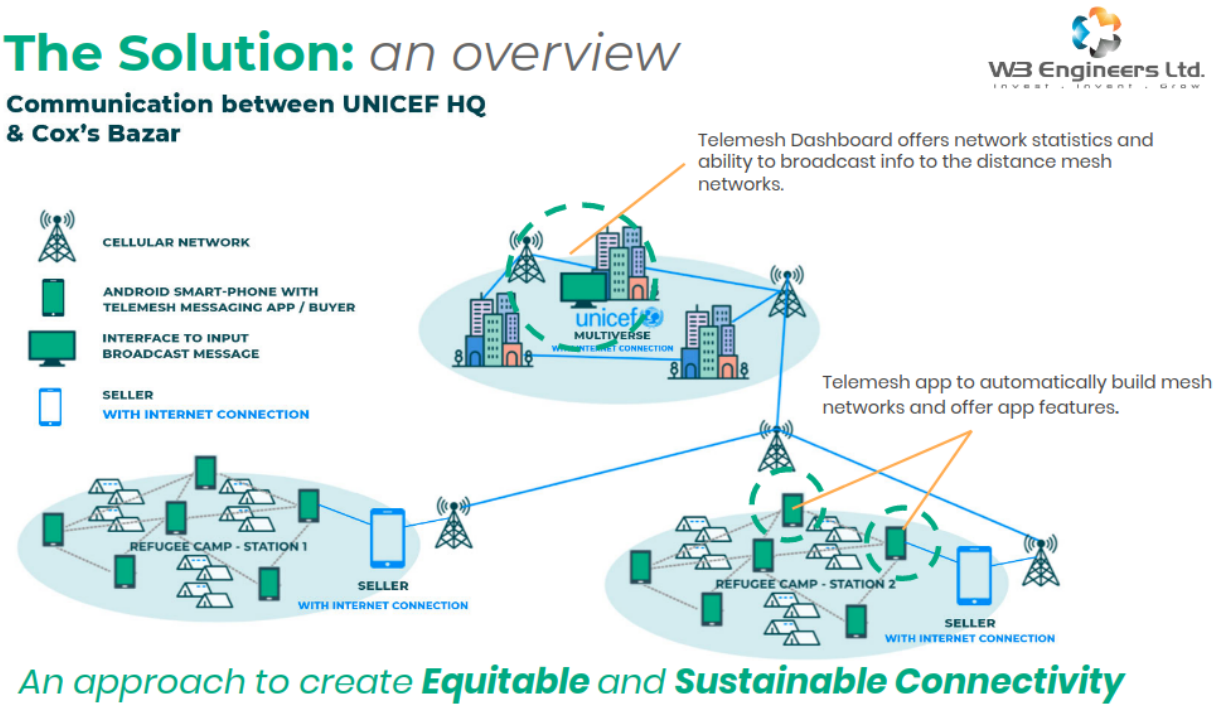
Globally, 68.5 million people are forcibly displaced at the time of writing this readme, and over 25.4 million are refugees. In Bangladesh, there are over 650,000 Rohingya refugees who have fled violence, mass killings and sexual abuse from neighboring Myanmar.2 Of those, nearly 60% are children, many of whom are orphaned Distributing information about humanitarian services to large numbers of refugees poses significant challenges for NGOs like UNICEF. While 40% of rural refugee households have smartphones, many are unconnected due to a lack of or poor telecommunications infrastructure or unaffordable cellular costs. The UNHCR believes that connecting refugees would ultimately transform humanitarian operations.

CHAPTER 2

Project's Goal

We intend to make use of mesh network. It allows for multi-hop, peer-to-peer connectivity directly between smartphones, instead of relying on internet and cell networks. Blockchain is used in the network to uniquely identify each node (smartphone) providing a trust layer to users without centralized signup. It also provides the infrastructure for users to connect multiple separate meshes by sharing an internet connection in exchange for ERC20 tokens. This offers an entirely new and unique method of information distribution not possible with existing technology.

For UNICEF, we plan to develop an open source messaging app to be tested in refugee camps, specifically, Cox's Bazar, Bangladesh. A broadcast channel would allow UNICEF to push vital information to smartphone users about services like vaccination clinics, maternity clinics and schools. The app would also allow refugees to message one another even if they do not have a SIM card or cellular data.


The Solution: an overview
WB Engineers Ltd.
Invest . Invent . Grow
Communication between UNICEF HQ & Cox's Bazar
Telemesh Dashboard offers network statistics and ability to broadcast info to the distance mesh networks.
CELLULAR NETWORK
ANDROID SMART-PHONE WITH TELEMESH MESSAGING APP / BUYER
INTERFACE TO INPUT BROADCAST MESSAGE
SELLER WITH INTERNET CONNECTION
unicef
MULTIVERSE
Telemesh app to automatically build mesh networks and offer app features.
REFUGEE CAMP - STATION 1
SELLER WITH INTERNET CONNECTION
REFUGEE CAMP - STATION 2
SELLER WITH INTERNET CONNECTION
An approach to create Equitable and Sustainable Connectivity

CHAPTER 3

Features

- User discovery in local mesh
- One to one messaging
- Message broadcast
- In app sharing
- Crypto wallet
- Mobile data buy and sell

CHAPTER 4

Project's Structure

```
.
|-- app
|-- src
|-- main
    |-- com.w3engineers.unicef
        |-- telemesh
            |-- data #local database, file, shared preferences etc.
            |-- ui #ui components
        |-- util
            |-- helper #Generic tasks like TimeUtil, NetworkUtil etc.
            |-- lib #third party library, component etc.
        |-- Application.java #Android Application class
|-- viper #W3Engineers wrapper module
|-- appshare #W3Engineers in app share module
|-- localserver #W3Engineers in app update module
|-- build.gradle
|-- settings.gradle
|-- versions.gradle
|-- gradle.properties
```

- **Alias** N/A
- **Commands** N/A

CHAPTER 5

Telemesh Deployment

After cloning the Telemesh github repo in your local machine it is effortless to build the Telemesh app.

Actually, The *Telemesh* app is using different `URLs` for different services. Preparing all those services is time-consuming as well as tedious too for outsiders.

Here if you want to simply deploy / build the *Telemesh* app, you don't need to worry at all about those tasks.

All the `URLs` and other permission related information are provided in a `.so` file as solid package to give you a staging environment the same as production for deployment purposes.

Just download the project and run.

But, we don't give the guarantee of continuous services for the staging environment, as it's only for deployment purposes.

After successful build, install the apk into any Android device (minimum android version `Lollipop - 5.0`)

and you need to download `TeleService` apk from Telemesh inside which will provide multihop mesh support into Telemesh.

5.1 Parse deployment

In Telemesh we are using `Parse` server to store analytics data from local mesh.

Please follow this Parse installation process in Android.

To configure parse with Telemesh, follow the steps are given below:

Step 1: Add parse server Android SDK version in `version.gradle`

Step 2: If you want to deploy your own parse server in any platform, you have to update parse server `URL` and parse server `APP-ID` in the Telemesh project.

Step 3. You have to add the `PARSE_URL` and `PARSE_APP_ID` in the GradleBuildValues interface that located in Constants class.

Parse Server installation Inside a Docker container

```
$ git clone https://github.com/parse-community/parse-server
$ cd parse-server
$ docker build --tag parse-server .
$ docker run --name my-parse-server -p 1337:1337  -d parse-server --appId␣
↪APPLICATION_ID --masterKey MASTER_KEY --databaseURI mongodb://mongo/test
```

If you need any help on any stage of your deployment and you want to communicate with us please join our Discord channel or communicate to our community manager through [media@telemesh.net] & [info@telemesh.net].

CHAPTER 6

Get started

Step 1: Clone repository: Navigate to directory where you want to keep source code. Open command prompt.

Execute below command:

```
git clone https://github.com/w3-engineers/telemesh.git
```

Here you will get how to install Git and start contribute to Open source

For more details please follow the *Telemesh Development Guideline*

Also you can use the following communication methods

- The `#get-help` channel on our Discord chat
- The mailing list [media@telemesh.net] for long term discussion.

Step 2: Sync and build: If everything is ok then sync and build should work as it should be.

Step 3: Test on device:

Minimum API: 21 (Lollipop - 5.0)

CHAPTER 7

Project Dependencies

Telemesh project has a dependency on wireless Mesh technology and Blockchain Technology

This project has core dependency on another open-source project Viper which will provide support for Mesh and Blockchain functionalities and expose set of api to ease development work. Please follow the *Viper Guideline*

CHAPTER 8

Development Guideline

8.1 Viper

8.1.1 Viper Introduction

`Viper` is the android library that acts as a communication bridge between Telemesh Android Application and Telemesh Service. It is responsible for coordinating the actions to the Telemesh Service. It can also perform some mappings to prepare the objects coming from the Android Application.

It provides few supports on `ui/ux` to ease developers daily development. Also it provides set of api for `mesh` support and `wallet` support. Currently `Telemesh` ui design use those supports from Viper.

8.1.2 Dependency

Include the library in app level build.gradle of Telemesh

```
dependencies{
     implementation 'com.github.w3-engineers:viper:<version_number>'
}
```

8.1.3 UI/UX Support

We all do many activities which are common to the most of the screens. Most famous are

- Design on Toolbar
- Recycler view clicklistener
- Manage empty view on recycler view

We don't think its wise to write all the component each time we want to design the screen since these components need to be consistent on all screen. Even if you able to achieve it by separate implementation consistency is difficult

to achieve and it's not good idea to write this for each screen. It's at this point where we can think of moving this implementation to the Base class/ BaseActivity.

Viper provides below supports

- Custom components (`BaseActivity`, `BaseFragment`, `BaseAdapter` etc.)
- Custom Widgets (`BaseButton`, `BaseRecyclerView`, `BaseEditText` etc.)
- `BaseToolBar` provides title to Toolbar and action for back button
- Close Coupled Behavior with Widget and Components
- Few configurable options (debugDatabase, Toasty etc.Still we are improving here)
- Enhanced support for Room (migration, creation of database, columns etc.)
- Necessary library added such a way so that developers can use without including in their gradle file (Timber, Multidex, Crashlytics, Debug Database etc.)
- `BaseSplashViewModel` provides time calculation facility and enforce ViewModel LiveData communication

BaseRecyclerView

BaseRecyclerView is a wrapper class of android RecyclerView

```
<RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/empty_layout"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:text="No data found"
            android:visibility="gone" />

        <com.w3engineers.ext.strom.application.ui.widget.BaseRecyclerView
            android:id="@+id/rv"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:brv_defaultAnimation="false"
            app:brv_emptyLayoutId="@id/empty_layout" // Empty View id. This is␣
↪mandatory field
            app:brv_viewMode="vertical" />

    </RelativeLayout>
```

- `app:brv_emptyLayoutId="@id/empty_layout"` This is compulsory filed if it does't set then you will get Runtime exception
- `app:brv_viewMode="vertical"` indicate how the RecyclerView scroll horizontally or vertically
- `app:brv_defaultAnimation="false"` Mark default animation enable or disable

BaseAdapter

BaseAdapter is a generic RecyclerView adapter which is capable to work with all types of data model.

Example

```
public class ExampleAdapter extends BaseAdapter<User> {
    @Override
    public boolean isEqual(User left, User right) {
        return false;
    }

    @Override
    public BaseAdapterViewHolder newViewHolder(ViewGroup parent, int viewType) {
        return null;
    }
}
```

Child class needs to implement *isEqual() and newViewHolder()* methods. No needs to override **onBindViewHolder()**

BaseToolBar

`activity_home.xml`

```
<com.w3engineers.ext.strom.application.ui.base.BaseToolBar
  android:id="@+id/home_toolbar"
  ...
  app:showHomeButton="true"            // this will show toolbar home button
  app:customTitle="@string/app_name"  // this will show toolbar title
  >
</com.w3engineers.ext.strom.application.ui.base.BaseToolBar>
```

`HomeActivity.java`

```
@Override
    protected int getToolbarId() {
        return R.id.home_toolbar;
    }
```

BaseButton:

BaseButton is a custom View class. You can design any types of Button with and without image, round corner and there are various properties with it.

-`app:bb_drawable="@drawable/button_gradient_blue"` is a mandatory field. If developer does not set this property it may causes Runtime exception

```
<com.w3engineers.ext.strom.application.ui.widget.BaseCompositeButton
                    android:id="@+id/btn_facebook_like"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginBottom="10dp"
                    android:padding="10dp"
                    android:textStyle="italic"
                    app:btn_borderColor="#FFFFFF"
                    app:btn_borderWidth="1dp"                        // Button border␣
↪width
                    app:btn_defaultColor="#3b5998"
                    app:btn_focusColor="#5577bd"                     // When click␣
↪show this focus color
                    app:btn_fontIconSize="15sp"
                    app:btn_iconPosition="right"                     // Icon position␣
↪(left, right, top, bottom)
                    app:btn_iconResource="@drawable/facebook"
```

(continues on next page)

(continued from previous page)

```
                    app:btn_radius="30dp"                                   // Button corner␣
↪radious
                    app:btn_text="Like my facebook page"
                    app:btn_disabledBorderColor="@color/colorAccent"
                    app:btn_disabledTextColor="@color/colorAccent"
                    app:btn_disabledColor="@color/colorAccent"
                    app:btn_textGravity="start"
                    app:btn_iconColor="@color/colorAccent"
                    app:btn_textColor="#FFFFFF" />
```

Till now nothing is mandatory, there are so many options here. This custom class will support for all types of button.

BaseEditText:

BaseEditText is a custom EditText wrapper, using this class it is possible to design EditText with and without label max, min char length and there are various options with it.

```
<com.w3engineers.ext.strom.application.ui.widget.BaseEditText
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:hint="Floating Label"
                app:bet_floatingLabel="highlight"
                app:bet_maxCharacters="10"                  // Max character size
                app:bet_minCharacters="2"                   // Min character size
                app:bet_autoValidate="true"
                app:bet_floatingLabelAlwaysShown="false"
                app:bet_checkCharactersCountAtBeginning="true"
                app:bet_baseColor="@color/colorAccent"
                app:bet_floatingLabelTextSize="20sp"
                app:bet_hideUnderline="true"
                app:bet_helperText="Helper"                // If it needs to help user␣
↪provide some example
                app:bet_helperTextAlwaysShown="true"
                app:bet_helperTextColor="@color/colorAccent"
                app:bet_primaryColor="@color/accent"/>
```

Use this class and its necessary properties.

BaseButton

```
<com.w3engineers.ext.strom.application.ui.widget.BaseButton
            android:id="@+id/btn_show_items"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="@string/show_data"
            android:padding="10dp"
            app:layout_constraintTop_toBottomOf="@+id/btn_add_item"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:bb_drawable="@drawable/button_gradient_blue"/>
```

BaseDialog

Base dialog is a custom dialog class, which force developer to set a layout file for custom design

```
protected abstract int getLayoutId();
protected abstract void startUi();
```

Are the two methods needs to child class implement.

DialogUtil

There are three overloading static methods here

```
public static void showDialog(Context context, String message, DialogListener␣
↪listener)
public static void showDialog(Context context, String title, String message,␣
↪DialogListener listener)
public static void showDialog(Context context, String title, String message, String␣
↪positiveText, String negativeText, final DialogListener listener)
```

Developer can call any one as his/her needs. It will show a default dialog

ItemClickListener:

```
public interface ItemClickListener<T> {
    /**
     * Called when a item has been clicked.
     *
     * @param view The view that was clicked.
     * @param item The T type object that was clicked.
     */
    void onItemClick(View view, T item);
}
```

Implement this interface in UI (Activity or Fragment) pass its reference to the Adapter

ItemLongClickListener

```
public interface ItemLongClickListener<T> {
    /**
     * Called when a item has been long clicked.
     *
     * @param view The view that was clicked.
     * @param item The T type object that was clicked.
     */
    void onItemLongClick(View view, T item);
}
```

For item long click listener implement this interface in UI (Activity or Fragment) and pass its reference to adapter

8.1.4 Mesh Support

To receive `EVENTS` from MeshService following Events are observed on Telemesh app end at `ViperUtil.java` class inside the package `com.w3engineers.unicef.util.helper`.

`ApiEvent.TRANSPORT_INIT` - After initializing mesh service this event provide mesh initialization state with own user/peer id

`ApiEvent.WALLET_LOADED` - After successfully wallet get loaded this event provide wallet status

`ApiEvent.PEER_ADD` - This event provide the new peer id when another user/peer get discovered through local mesh

`ApiEvent.PEER_REMOVED` - This event provide the remove peer id when user/peer get removed from local mesh

`ApiEvent.DATA` - This event provide the received data in byte array format

`ApiEvent.DATA_ACKNOWLEDGEMENT` - This event provide the send data/message acknowledgment status with message-id

`ApiEvent.USER_INFO` - This event sends the connected peer's info like peer name, peer image index, etc.

```
private void initObservers() {

        AppDataObserver.on().startObserver(ApiEvent.TRANSPORT_INIT, event ->
↪{
            TransportInit transportInit = (TransportInit) event;

            if (transportInit.success) {
                myUserId = transportInit.nodeId;

                onMesh(myUserId);
            }
        });

        AppDataObserver.on().startObserver(ApiEvent.WALLET_LOADED, event -> {
            WalletLoaded walletLoaded = (WalletLoaded) event;

            if (walletLoaded.success) {
                onMeshPrepared();
            }
        });

        AppDataObserver.on().startObserver(ApiEvent.PEER_ADD, event -> {
            PeerAdd peerAdd = (PeerAdd) event;
            peerDiscoveryProcess(peerAdd.peerId, true);
        });

        AppDataObserver.on().startObserver(ApiEvent.PEER_REMOVED, event -> {
            PeerRemoved peerRemoved = (PeerRemoved) event;
            peerDiscoveryProcess(peerRemoved.peerId, false);
        });

        AppDataObserver.on().startObserver(ApiEvent.DATA, event -> {

            DataEvent dataEvent = (DataEvent) event;

            dataReceive(dataEvent.peerId, dataEvent.data);
        });

        AppDataObserver.on().startObserver(ApiEvent.DATA_ACKNOWLEDGEMENT,␣
↪event -> {

            DataAckEvent dataAckEvent = (DataAckEvent) event;

            onAck(dataAckEvent.dataId, dataAckEvent.status);

        });

        AppDataObserver.on().startObserver(ApiEvent.USER_INFO, event -> {

            UserInfoEvent userInfoEvent = (UserInfoEvent) event;
```

(continues on next page)

(continued from previous page)

```
            UserModel userModel = new UserModel().setName(userInfoEvent.
↪getUserName())
                    .setImage(userInfoEvent.getAvatar())
                    .setTime(userInfoEvent.getRegTime());

            peerAdd(userInfoEvent.getAddress(), userModel);
        });

    }
```

To receive data from Viper to Telemesh Android app following abstract methods are used on Telemesh app end at `MeshDataSource.java` class inside the package `com.w3engineers.unicef.telemesh.data.helper`.

`protected abstract void onMesh(String myMeshId)` - When observer receive `ApiEvent.TRANSPORT_INIT` EVENT then this method get called.

`protected abstract void peerAdd(String peerId, byte[] peerData)` - When observer receive `ApiEvent.DATA` EVENT then this method get called.

`protected abstract void peerAdd(String peerId, UserModel userModel)` - When observer receive `ApiEvent.USER_INFO` EVENT then this method get called.

`protected abstract void peerRemove(String nodeId)` - When observer receive `ApiEvent.PEER_REMOVED` EVENT then this method get called.

`protected abstract void onData(String peerId, ViperData viperData)` - When observer receive `ApiEvent.DATA` EVENT then this method get called.

`protected abstract void onAck(String messageId, int status)` - When observer receive `ApiEvent.DATA_ACKNOWLEDGEMENT` EVENT then this method get called.

`protected abstract boolean isNodeAvailable(String nodeId, int userActiveStatus)` - To check whether the user/peer is currently active/online

8.1.5 Wallet Support

`public static WalletManager getInstance()` - returns the WalletManager singleton object.

`public boolean hasSeller()` - returns if user is connected to any seller type user.

`public String getMyAddress()` - returns user's wallet address.

`public int getMyEndpoint()` - returns user's current blockchain network endpoint value.

`public boolean isGiftGot()` - returns if user already has received the gift point and ether by airdrop.

`public void setWalletListener(WalletListener walletListener)` - set WalletListener from your wallet activity to receive various events

`public static void openActivity(Context context, byte[] picture)` - if you want to use the default wallet activity, calling this method will do that. @params: 1. Context: activity context, 2. byte[]: picture you walt to show in the default wallet page.

`public boolean giftEther()` - call this method to initialize point and ether gift process. If user is capable of getting gift, will get it.

`public void setEndpoint(int endpoint)` - call this method to set different blockchain network endpoint value (this is related to configuration file provided by application end initially)

`public void refreshMyBalance()` - call this method to refresh balance.

`public void getAllOpenDrawableBlock()` - call to withdraw pending balances stored in the channel.

`public LiveData<Double> getTotalEarn(String myAddress, int endPoint)` - observe this to get total earning live data by user

`public LiveData<Double> getTotalSpent(String myAddress, int endPoint)` - observe this to get total spent live data by user

`public LiveData<Double> getTotalPendingEarning(String myAddress, int endPoint)` - observe this to get pending earning(stored in microraiden channel) live data by seller

`public Flowable<List<NetworkInfo>> getNetworkInfoByNetworkType()` - observe this to get balance change, blockchain network information change.

`public void createWallet(Context context, String password, WalletCreateListener listener)` - This api is used to create wallet. Call this the user is totally new.

`public void loadWallet(Context context, String password, WalletLoadListener listener)` - This api is used to load wallet for a returning user, provided that wallet file already exists in the system.

`public void importWallet(Context context, String password, Uri fileUri, WalletImportListener listener)` - This api is used to import wallet, provided that user already has a wallet file of his/her own created from other source.

```
public interface WalletCreateListener {
    ``void onWalletCreated(String walletAddress, String publicKey)`` - called  when wallet is created.
    ``void onError(String message)`` - called when there is an error
}


public interface WalletLoadListener {
    ``void onWalletLoaded(String walletAddress, String publicKey)`` - called when wallet is loaded.
    ``void onError(String message)`` - called when there is an error
}



public interface WalletImportListener {
    ``void onWalletImported(String walletAddress, String publicKey)`` - called  when wallet is imported.
    ``void onError(String message)`` - called when there is an error
}


public interface WalletListener {
    ``void onGiftResponse(boolean success, boolean isGifted, String message)`` - called at various steps in ether and point gift process.
    ``void onBalanceInfo(boolean success, String msg)`` - called when refresh balance response received

}
```

8.1.6 Data plan Support

`public static DataPlanManager getInstance()` - returns DataPlanManager singleton object

`public int getDataPlanRole()` - returns user role

`public long getSellAmountData()` - returns amount of data user wnts to share.

`public int getDataAmountMode()` - returns whether shared data is limited or unlimited.

`public long getSellFromDate()` - returns timestamp from when data selling starts.

`public long getSellDataAmount()` - returns value of data amount what user set for limited data plan.

`public long getRemainingData()` - returns remaining amount of data shared by seller.

`public long getUsedData(Context context, long fromDate)` - returns used data amount from specific timestamp.

`public static void openActivity(Context context, int imageValue)` - call this method to open default dataplan activity.

`public static void resumeMessaging()` - call this method to resume seller side functionality to help buyer messaging.

`public void closeMesh(int role)` - call this method to stop mesh communication.

`public void roleSwitch(int newRole)` - call this method to switch user role

`public void setSellFromDate(long fromDate)` - call this method to set data selling starting timestamp.

`public void setDataAmountMode(int mode)` - call this method to set user choise for data sharing limited/unlimited, value 1 for limited and 0 for unlimited.

‘‘public void setSellDataAmount(Long sharedData) ‘‘ - call this method to set data sell amount in MB

`public void closeAllActiveChannel()` - call this method to close all active channel by seller.

`public void initPurchase(double amount, String sellerId)` - call this method to purchase data in MB from seller.

`public void closePurchase(String sellerId)` - call this method to close any purchased channel by buyer.

`public void processAllSeller(Context context)` - call this method to process the connected seller list in UI by buyer.

`public void setCurrentSeller(Context context, String sellerId, String currentSellerStatus)` - call this method to set status of the seller.

`public void precessDisconnectedSeller(Context context, String sellerId)` - call this method to process disconnected seller from list.

`public void setDataPlanListener(DataPlanListener dataPlanListener)` - set DataPlanListener from dataplan activity.

`public Flowable<List<Seller>> getAllSellers()` - observe this to get any change in connected seller list

8.2 Telemesh

1. Find the `versions.gradle` in the Telemesh root directory of the repo and any new support library reference should be added here.

2. Any support library on app-level `build.gradle` should be added in this way

```
implementation deps.support.app_compat
implementation deps.support.design
implementation deps.constraint_layout
implementation deps.support.recyclerview
implementation deps.support.cardview
```

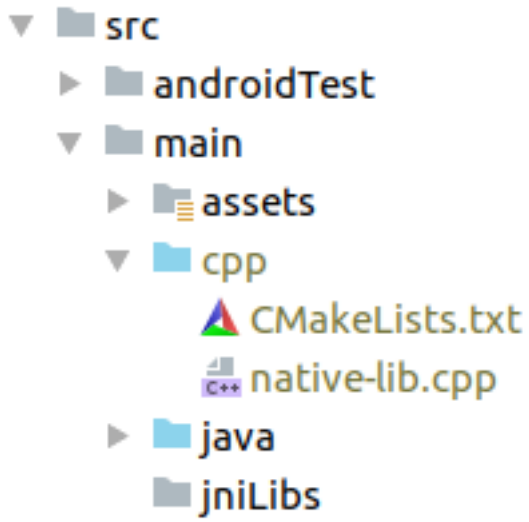
3. `Viper` dependency should be added in the same way inside the app-level `build.gradle`
4. The *Telemesh* app is using different `URLs` for different services. Preparing all those services is time-consuming as well as tedious too. But if you are a contributor to *Telemesh* app, you don't need to worry at all about those monotonous tasks. All the `URLs` and other permission related information are provided in a `.so` file as solid package to give you a staging environment the same as production for development purposes. Just download the project and run.

   But, we don't give the guarantee of continuous services for the staging environment, as it's only for development and testing purposes.

5. If you want to deploy your server machines and services you need to do a little more work. Assuming that you already have the services installed on the online machine(s), please follow the steps below.

   a. If android **NDK** and **CMake** are not installed on your development machine, please download and install them using **SDK manager**. For more help visit https://developer.android.com/studio/projects/install-ndk#default-version

   b. Create a directory named *cpp* under the *main* package of the *Telemesh* app.



   c. Put/Create the following two files into the *cpp* folder.

      1. CMakeLists.txt
      2. native-lib.cpp

   d. `CMakeLists.txt` build script can be prepared from the following link: https://developer.android.com/studio/projects/configure-cmake#create_script

   If you need any help on any stage of work and you want to communicate with us please join our Discord channel.

e. Any change or addition of credentials has to do under `native-lib.cpp` file. Copy the following code snippet and paste it into this file.

```
#include <jni.h>
#include <string>

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getBroadCastToken(JNIEnv *env, jobject) {
  std::string broadcast_token= "Here have to set your broadcast token";
  return env->NewStringUTF(broadcast_token.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getBroadCastUrl(JNIEnv *env, jobject) {
  std::string broadcast_url = "Here have to set your broadcast url";
  return env->NewStringUTF(broadcast_url.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getParseUrl(JNIEnv *env, jobject) {
  std::string parse_url = "Here have to set your parse server url";
  return env->NewStringUTF(parse_url.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getParseAppId(JNIEnv *env, jobject) {
  std::string parse_app_id = "Here have to set your parse app id";
  return env->NewStringUTF(parse_app_id.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getAuthUserName(JNIEnv *env, jobject) {
  std::string auth_user_name = "Here have to set your Authenticate name";
  return env->NewStringUTF(auth_user_name.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getAuthPassword(JNIEnv *env, jobject) {
  std::string auth_password = "Here have to set your authenticate password";
  return env->NewStringUTF(auth_password.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getFileRepoLink(JNIEnv *env, jobject) {
  std::string file_repo_link = "Here have to set your file download repo link
↪";
  return env->NewStringUTF(file_repo_link.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_w3engineers_unicef_telemesh_data_helper_AppCredentials_
↪getConfiguration(JNIEnv *env, jobject) {
```

(continues on next page)

(continued from previous page)

```
  std::string config_file = "Here have to set a Json file as string like␣
↪below";
  return env->NewStringUTF(config_file.c_str());
}
```

JSON file for configuration:

```
{
  "config_version_name":"0.0.1",
  "config_version_code":1,
  "token_per_mb":1.0,
  "default_network_type":2,
  "token_guide_version":0,
  "GIFT_DONATE_LINK" : "Here set your gift donate link",
  "wallet_rmesh_available": false,
  "network": [
    {
      "network_type":2,
      "network_name":"Kotti",
      "network_url":"Here set network url",
      "currency_symbol":"ETC",
      "token_symbol":"TMESH",
      "token_address":"Here set your token address",
      "channel_address":"Here set your channel address",
      "gas_price":25000000000,
      "gas_limit":800000,
      "token_amount":0,
      "currency_amount":0
    }
  ]
}
```

For more query please join us through Discord channel.

e. If any new credential is added have to add an API into AppCredentials.java class to access that credentials.

f. Delete the following two files from **jniLibs** package.

1. armeabi-v7a
2. x86

g. Find the externalNativeBuild {} tag from app-level build.gradle and uncomment this line: path `src/main/cpp/CMakeLists.txt`

h. Execute Gradle sync

6. Now check the `ViperUtil.java` class and find the constructor `ViperUtil` where we use the above credentials

```
protected ViperUtil(UserModel userModel) {
  try {
    context = MainActivity.getInstance() != null ? MainActivity.
↪getInstance() : TeleMeshApplication.getContext();
    String appName = context.getResources().getString(R.string.app_name);


    String AUTH_USER_NAME = AppCredentials.getInstance().getAuthUserName();
```

(continues on next page)

(continued from previous page)

```
    String AUTH_PASSWORD = AppCredentials.getInstance().getAuthPassword();
    String FILE_REPO_LINK = AppCredentials.getInstance().getFileRepoLink();
    String PARSE_APP_ID = AppCredentials.getInstance().getParseAppId();
    String PARSE_URL = AppCredentials.getInstance().getParseUrl();
    String CONFIG_DATA = AppCredentials.getInstance().getConfiguration();


    SharedPref sharedPref = SharedPref.getSharedPref(context);
    String address = sharedPref.read(Constants.preferenceKey.MY_WALLET_
↪ADDRESS);
    String publicKey = sharedPref.read(Constants.preferenceKey.MY_PUBLIC_
↪KEY);
    String networkSSID = sharedPref.read(Constants.preferenceKey.NETWORK_
↪PREFIX);

    initObservers();

    if (TextUtils.isEmpty(networkSSID)) {
      networkSSID = context.getResources().getString(R.string.def_ssid);
    }

    viperClient = ViperClient.on(context, appName, context.getPackageName(),␣
↪networkSSID, userModel.getName(), address, publicKey, userModel.getImage(),
↪ userModel.getTime(), true)
    .setConfig(AUTH_USER_NAME, AUTH_PASSWORD, FILE_REPO_LINK, PARSE_URL,␣
↪PARSE_APP_ID, CONFIG_DATA);

  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

7. For wallet design currently, we are using default design from `Viper`
8. In Telemesh we are using `Parse` server to store analytics data from local mesh.

   Please follow this Parse installation process in Android.

   To configure parse with Telemesh, follow the steps are given below:

   Step 1: Add parse server Android SDK version in `version.gradle`

   Step 2: If you want to deploy your own parse server in any platform, you have to update parse server `URL` and parse server `APP-ID` in the Telemesh project.

   Step 3. You have to add the `PARSE_URL` and `PARSE_APP_ID` in the GradleBuildValues interface that located in Constants class.

   The sample Parse model (Table) structure is

   ```
   ParseObject parseObj = new ParseObject("table_name");
   parseObj.put("column_name","value");
   ............
   parseObj.saveInBackground();
   ```

   The parse server table structure and save/update process located in `parseapi` package.

   Parse Server installation Inside a Docker container

```
$ git clone https://github.com/parse-community/parse-server
$ cd parse-server
$ docker build --tag parse-server .
$ docker run --name my-parse-server -p 1337:1337  -d parse-server --appId␣
↪APPLICATION_ID --masterKey MASTER_KEY --databaseURI mongodb://mongo/test
```

Happy Coding :)

After successful build, install the apk into any Android device (minimum android version `Lollipop - 5.0`)

and you need to download `TeleService` apk from Telemesh inside which will provide multihop mesh support into Telemesh.

CHAPTER 9

Create branch

Everyone should create their own branch from development branch with convention as *feature/TICKET_NO_TASK_HINT* e.g. *feature/TEL-123_image_change*. Soon he/she finish his task, he/she should push and request for merge with development branch.

For release a well-tested production ready app should marge from development to master branch. Android keystore for app release should pass to top level management via email. Make sure you have putted a TAG for each release on git.

For bugs, a hotfix branch should create first from the release branch with format of *hotfix/TICKET_NO_TASK_HINT* and sync between development and master branch.

We are following the GitFlow standards

CHAPTER 10

Local development environments

You will be glad to know that you can start Telemesh Android application development on either of the following operating systems

- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.

Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web.

Following is the list of software's you will need before you start your Android application programming.

- Java JDK 8 or later version
- Android Studio 3.3 or later version

CHAPTER 11

Development Prerequisites

11.1 Android Core

- Understand Object Oriented Programming (OOP) and various constraints of Java programming language
- Know how to build and run an Android app
- Understand the Android activity lifecycle
- Understand the `MVVM architecture` of the Android system
- Be able to display a message outside your app's UI using `Notifications`
- Understand how to localize an app
- Be able to schedule a background task using `JobScheduler`
- Be able to use RxJava 2 and basic understanding of various design patterns like `Singleton`, `PubSub` and so on..

11.2 User Interface

- Be able to create an `Activity` that displays a Layout
- Be able to use `Data Binding` for view binding
- Be able to construct a UI with `ConstraintLayout`
- Understand how to display items in a `RecyclerView`
- **Be able to bind local data to a RecyclerView list using the `Paging library`** and so on..

11.3 Data Management

- Understand how to define data using `Room entities`
- Be able to access Room database with data access object `(DAO)`
- Know how to observe and respond to changing data using `LiveData`
- Understand how to use a Repository to mediate data operations
- Be able to read and parse raw resources or asset files and so on..

11.4 Testing

- Thoroughly understand the fundamentals of testing
- Be able to write useful local `JUnit` tests
- Understand the `Espresso` UI test framework
- Know how to write useful automated Android tests and so on..