viper

Release 0.1.0

Jan 13, 2020

Contents

CHAPTER 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.

CHAPTER 2

Dependency

Include the library in app level build.gradle of Telemesh

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

CHAPTER 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
```

(continues on next page)

(continued from previous page)

```
  >
</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"
                    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"
```

(continues on next page)

(continued from previous page)

```
                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.
     *
```

(continues on next page)

(continued from previous page)

```
     * @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

CHAPTER 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;
```

(continues on next page)

(continued from previous page)

```
            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;

            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

CHAPTER 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

}
```

CHAPTER 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