SOFTWARE TECHNICAL DOCUMENTATION FOR THE HSAPPLICATION

ABSTRACT

The HSApplication is an android application project for the Hannah and Serge group limited company. The project is currently under the development process. The company already has a working website which a user can create an account, sign in, can access a dashboard where they can be able to see the current investment, loan balance and latest transactions. They can also invest their money into the HS group microfinance.

The objective of creating the HSApplication is to create a better experience for the customers, an application that will help the customers be able to apply the HS group loans from anywhere at an easy way. The application would be able to enable the customers be able to work with utilities as well such as the buying of airtime, paying TV cost and paying for water bills I the comfort of their zone.

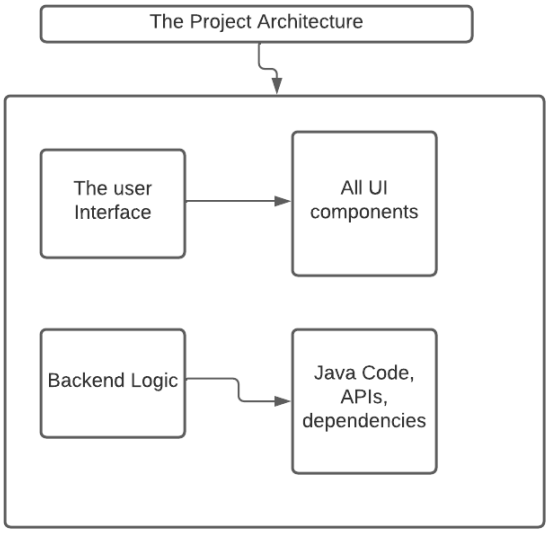
This document describes the technical part of the HSApplication, the user navigation through the application, the technological architecture, the user interface of the application and the user experience. These technical document is as a result of the collaborative recommendation between the developers of the Amba digital Company limited and the shareholders of the Hannah and Serge group. That is in coming up with the idea and suggesting a better UI strategy of developing the application. The normal users were also consulted in the process just to give their views.

Contents

PRODUCT ARCHITECTURE

The architecture of the HS Application consist of two main parts these includes the user interface which consist of the front end part where the user will interact with and the backend logic. The backend logic is what runs the whole application.

It is what enables the application to function as the user navigates through the application.



USER INTERFACE

The user interface is the front end part of the application where the users of the application interact with. It is normally colored with beautiful colors that is pleasing to the user. Each and every user Interface is designed in such a way that the user can have a nice user experience whenever they are using the application. A good user interface in most case has to be responsive in order to fit each every user device no matter the screen size.

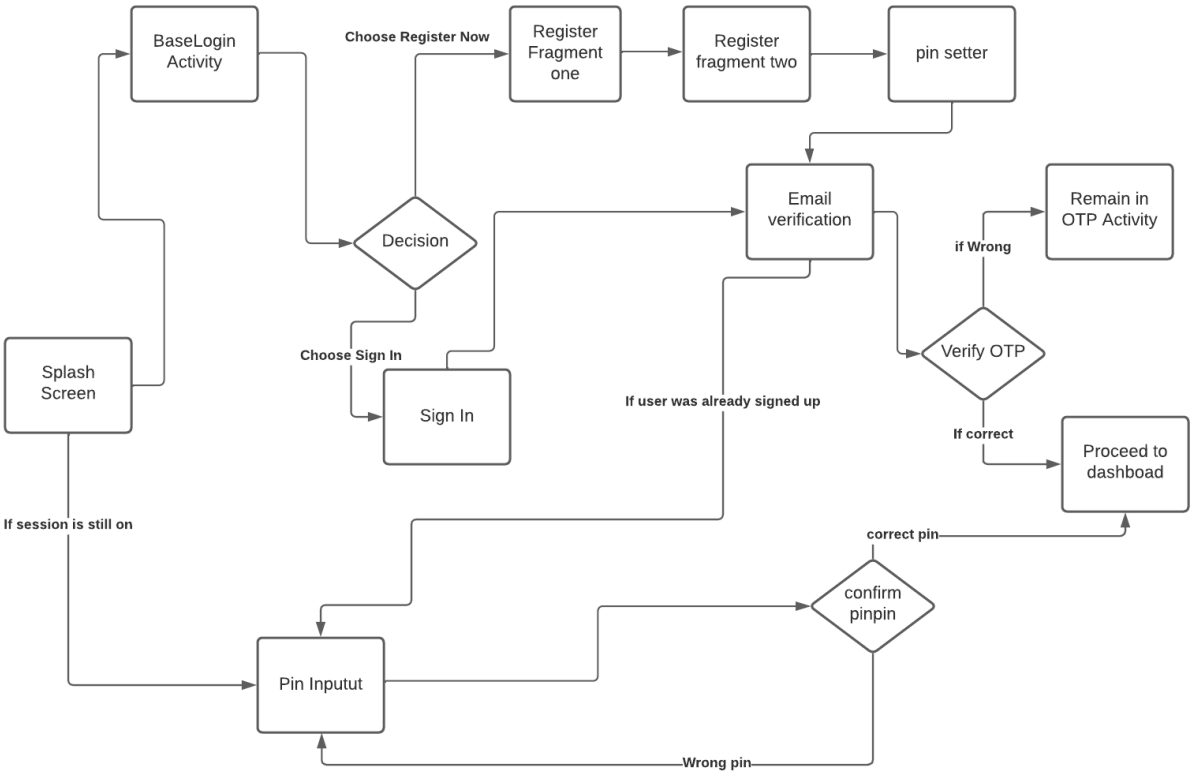
The HS Application has components from the android architecture such as navigation components, fragments, sweet alert dialogues, toolbars etc.

It also includes the android widgets such as button, text views, and Edit texts. The layouts used in the application includes menu, linear, constrain.

Navigation

The navigation component in the HS Application explains the path in which the user goes through as they use the android application. The following are the paths a user can navigate through in the application.

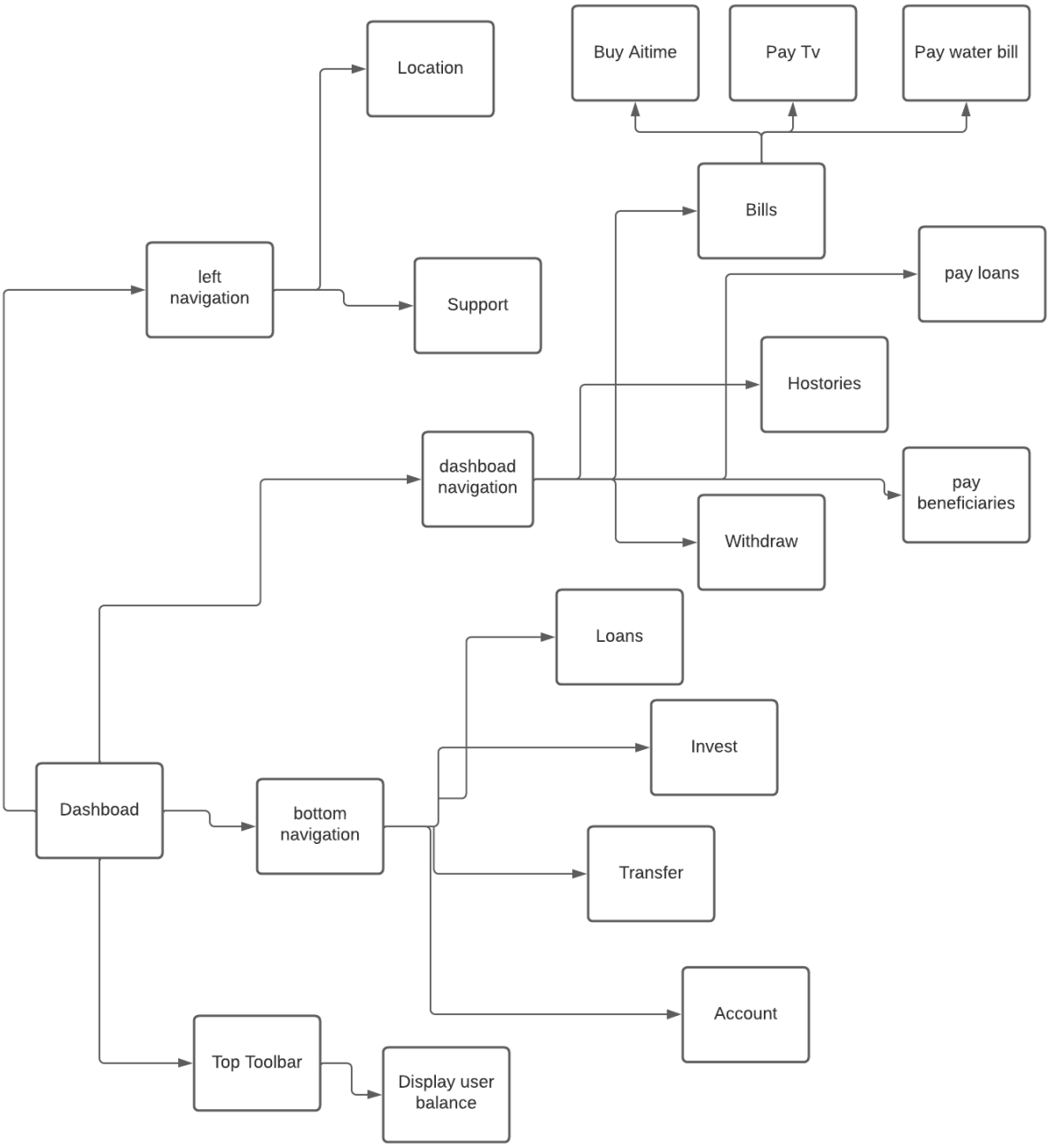
The authentication navigation route



The above diagram shows how the user navigates through the authentication activities. From the above diagram, a user can sign up that is registering as a new member in the Hs group and after entering all the details they should set pin and verify email before a successful sign up.

A user can also log in either after uninstalling the application or after logging out of the application. In both cases a user must verify his or her account through a onetime password. A user can furthermore access the application after exiting when the session still exists by inputting their pin. If the pin inputted is correct then the user can access the dashboard successfully.

After successful Authentication route after dashboard access.



On successful login, a user accesses the dashboard where they can be able to do as illustrated in the diagram above. There two common navigations that a user can access. The left navigation which enables the user to view the location of the companies' offices on google maps and the support that allows the user to raise an issue to the company's support team.

The bottom navigation allows a user to access the different types of loans offered by the Hannah and Serge group which in this case includes the logbook loans, Business loans, Kilimo loans, Title deed loans, Personal loans, Chama loans, and Bima loans. The invest tab is where a user can be able to invest their money into the HS group. Normally there are four types of investments that a customer can make. First is the Standard loan which ranges from 500 Shillings to 20, 000 Shillings. Second, there is platinum which ranges between 100, 000 Shillings to 250,000 Shillings. Finally, there is the gold loans which ranges from 250,000 shillings to 1,000,000 shillings. The transfer tab enables a user to be able to make transfers from within the HS Application and outside the HS Application. The internal transfers is the one that allows a user to transfer a funds to a bank account within the microfinance while the external transfers allows a user to transfer funds to other banks. The account tab will enable a user to view his or her account details starting from account number to pin and password.

The toolbar will allow the user to check his or her balance.

The dashboard navigation will also enable the user to be able to pay beneficiaries, access bills section where they can pay the different types of bills including buying airtime, paying TV, and paying water. With the dashboard navigation the user can also access the different types of transaction histories which includes the transaction history, Investment history, loan history and deposit history.

To understand more on the User Interface design visit this URL document.

https://github.com/Amba-Digital-Marketing/Test_Project_HS_Application/blob/main/HS%20Microfinance%20%5BAutosaved%5D%20(1).pptx

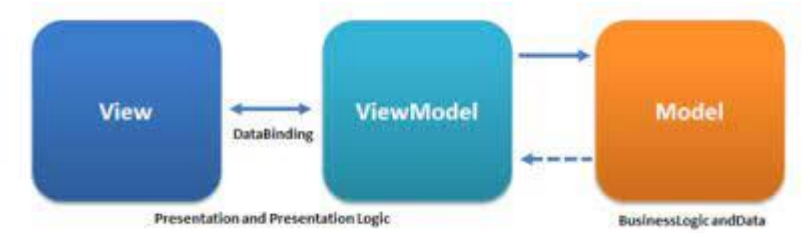
LOGIC INTERFACE

This section of the logical interface describes the backend architecture of the code that make up the HS Application, all the technologies used in the HS Application, all the structures on how the code is designed to ease your understanding on the application backend code. In this section we will go through all the APIs, integrations, code dependencies, Project dependencies, and the algorithms. This section explains clearly the functionalities in the HS Application comes to reality. The architecture used in the whole of the HS Application architecture is the MVVM model architecture. Let’s look in detail what MVVM architecture is. The MVVM architecture stands for model - view - view model.

It is a type of software architecture which separates the User Interface which is the view in this case from the business logic which is held in the view model files.

The model section will hold the class models that will model the data from storage sources e.g. the data from the database.

The view section contains the code e.g. validation of user input from the frontend such as fragments and activities. Any UI logic is handled in the view section before the data from the user is passed to the view model where the server business logic is handled. In most cases the business logic in the HS Application involves the API implementation where the end points are consumed in the application.



Another commonly used code structure is the usage of livedata and the MutableLivedata. The livedata is used in almost all the parts of the code base in the setting a value to a variable and in modifying the value of the variable.

The whole application also uses the shared preferences functionality in java in the assigning session's functionality. All the application sessions are placed in one file in the application called Timber.java. This file contains getter and setter functions that are used in setting and retrieving the session's variables.

There is the use of firebase dependencies that are used in the application to determine the track of user activity in the application. It is also used in the design of the support section of the application.

The network section is used in the application as the core part where all the retrofit callings are passed through. The APIService.java is a java interface class that it contains all the APIs endpoints that connect to the server. The retrofit instance is a java file that contains two functions that are called all through the application. These functions include the getRetroClient(contest) which takes the contest and the getRetroClientWithToken(token) which receives the token. Finally, there is the token interceptor file whose function is to call the token and intercept in all the calls. The token is an important key that is used in calling the rest of the APIs and is generated on the calling of the login endpoint.

Project tree structure

The project structure shows how the code of the HS Application is architectures in the android studio java code. How the folders are arranges right from the start. How it displays the MVVM architecture in the project, where the layout folder files are located and where the backend code

including the UI sections are located and the view model are located. To be able to see this tree structure visit the following GitHub URL link.

https://github.com/Amba-Digital-Marketing/Test_Project_HS_Application#readme

APIS

API stands for Application interface. It is a software intermediary that allows two applications to communicate with each other. In the Hs Application APIs are used all over especially in the view model section where the business logic is handled. It is in the business logic where the retrofit code is mostly implemented. This section discusses the different types of APIs used in the application and how they are important in the HS Application. This will better the understanding of the usage of APIs in the application for future maintenance.

Each and every API in the HS Application is made up of a base URL and the extension of the base URL which makes each endpoint unique. Like all the APIs, the HS Application APIs has got methods that defines the way data is being called. These methods include the GET, POST, PUT, DELETE methods that performs the normal CRUD operations in the server. The base URL is located in the retrofit instance file in network folder of the java backend files. The extensions of the base URL are located in the java interface file with the retrofit call back methods.

This section will explain both the full API endpoint and how they are interact with the interface class. It also explains the type of data that is passed to the API and the type of data that is received from the endpoint in the case of GET APIs.

APIs in the authentication Section

The APIs in this section includes those that are used in the sign in, sign up and the reset password and reset pin modules.

First of all we are going to start with the Sign up APIs. This is the API which is called with retrofit instance to insert user data into the server. The following is the API endpoint consisting of the base URL and the extension of the base URL located in the APIService.java file in the network package in the application.

```
https://www.member.hsgroup.tech/api/register
```

This end point is a POST API and it accepts application/Json as the headers and it is of type POST method. Its role is to pass the data from the user front end user interface including the name, email, phone number, id number, password, image, terms agreement condition and the pin.

All this endpoint does is to insert new data user into the database.

```
@Headers("Accept: application/json")
@Multipart
@POST("register")
Call<UserRegResponse> registerUser(
        @Part("name") RequestBody name,
        @Part("email") RequestBody email,
        @Part("phone_number") RequestBody phone_number,
        @Part("kra") RequestBody kra,
        @Part("password") RequestBody password,
        @Part("password_confirmation") RequestBody password_confirmation,
        @Part("image\"; filename=\"kra.png\" ") RequestBody photo,
        @Part("term_condition") RequestBody term_condition,
        @Part("pin") RequestBody pin
);
```

The OTP API is another API that is a part of the authentication system and its role is to enable the user verify their own email account. This API is responsible for sending the user a one-time password where they are support to retrieve it and input in the application and the application is support to confirm if that is the correct. The API is supposed to return a response class where the is an entity class called forgotPasswordResponse which will mold the response data and then the application will then check the data sent to the email and validate the OTP. The OTP will expire in a period of twenty seconds therefore the user is required to enter the OTP within the time frame. In case the OTP expires a user has the option of regenerating a new OTP again.

The API passes the email as a field in the API body.

```
https://www.member.hsgroup.tech/api/"user/otp
```

```
//resetPassword
@POST("password/forgot-password")
@FormUrlEncoded
Call<ForgotPasswordResponse> forgotpassword(
        @Field("email") String email);
```

The set user pin is another API in the authentication section whose role is to allow a user to set pin during the registration process. This API requires the user to pass his or her email and pin as well.

It takes the POST method which will store data into the server.

https://www.member.hsgroup.tech/api/set-user-pin

```
// update pin on login takes just new pin
@Headers("Accept: application/json")
@FormUrlEncoded
@POST("set-user-pin")
Call<String> updatepinOnLogin(
        @Field("email") String email,
        @Field("pin") String pin
);
```

The pin update API will enable the user to update his or her pin. This API takes the POST method which will store the new pin the server. In the body, It will take the updated pin and the current pin as the field parameters.

The following ids the full API endpoint.

https://www.member.hsgroup.tech/api/account/pin/update

This is the call of the same API in the interface class.

```
// update pin
@Headers("Accept: application/json")
@FormUrlEncoded
@POST("account/pin/update")
Call<PinChangeResponse> updatepin(
        @Field("updated_pin") String updated_pin,
        @Field("current_pin") String current_pin
);
```

HOME APIS

SIDE NAVIGATION APIs

The logout API is responsible for enabling the user to end his her activities in the application. With the logout API a user can end session in the server and also the local sessions. The following is the API endpoint.

https://www.member.hsgroup.tech/api/logout

This is the call of the logout endpoint in the interface class.

```
//Logout endpoint
@Headers("Accept: application/json")
```

```
@GET("logout")
Call<ResponseBody> logOutNow();
```

Another functionality in the side navigation component is the support section this contains a series of API that are used together to bring the support section into a reality.

The first one is the POST support API which enables the user to generate an issue to the support team.

These method takes title and comment from the user and passes into the API as the fields. These details are then sent to the server.

https://www.member.hsgroup.tech/api/support

```
// create a  support
@Headers("Accept: application/json")
@FormUrlEncoded
@POST("support")
Call<SupportResponse> createASupport(
        @Field("title") String title,
        @Field("comment") String comment
);
```

This other API gets one support item that will enable a user view only that particular item in an activity. This takes the GET method since it retrieves the item from the server.

https://www.member.hsgroup.tech/api/support/{id}

```
//getSupportId
@Headers("Accept: application/json")
@GET("support/{id}")
Call<SingleSupportItem> getOneSupportItemById(
        @Path("id") int id
);
```

This API will answer the most commonly asked questions. It returns the title and the answer of the commonly asked question. This API uses the GET method to retrieve the data from the server.

The following is the API endpoint.

https://www.member.hsgroup.tech/api/make-AI-ChatBot

```
//getSupportId
@Headers("Accept: application/json")
@GET("make-AI-ChatBot")
Call<SingleSupportItem> moreopenAIData(
        @Field("message") String title,
        @Field("engine") String comment
);
```

This API will enable the user to comment the support item. The method used is PUT method equivalent to update functionality. This API passes the id in the URL path section using the @path in the body.

https://www.member.hsgroup.tech/api/support/{id}

```
//Add comment SupportId
@Headers("Accept: application/json")
@FormUrlEncoded
@PUT("support/{id}")
Call<AddCommentSupportResponse> addCommenttoSupportItemById(
        @Path("id") int id, @Field("comment") String comment
);
```

To get all the support requests we use the following endpoint. This takes the GET method and the output of the data from the server is molded in an entity folder called SupportHistory.java file.

https://www.member.hsgroup.tech/api/user/get-all-support-requests

The following is the retrofit instance class code.

```
//get Support Items
@Headers("Accept: application/json")
@GET("user/get-all-support-requests")
Call<SupportHistory> getSupportItems();
```

Bills APIs

The bills section is a combination of many functionalities. These includes the buy airtime section, pay TV section, and pay water section. The following API endpoint is responsible for the purchase of airtime in the android application. The method used is the POST method which sends sender, biller name, amount, phone and account as the fields into the server.

https://www.member.hsgroup.tech/api/ipay/create/bill

In return the server returns an output response which is molded in the BillPayment.java file which is located in the entity folder in the HS Application.

The following is the retrofit instance class method.

```
//Buy airtime APi endpoint
@FormUrlEncoded
@Headers("Accept: application/json")
@POST("ipay/create/bill")
Call<BillPayment> buyAirtime(
        @Field("sender") String sender,
        @Field("biller_name") String biller_code,
        @Field("amount") String amount,
        @Field("phone") String phone,
        @Field("account") String account
);
```

The e deposit APIs allows the user to deposit funds into the HS Application. This endpoint uses the POST method which will write data into the server as soon as the transfer is complete. This API will take the amount and currency as the field parameters.

https://www.member.hsgroup.tech/api/edeposit/2/check

```
@FormUrlEncoded
@Headers("Accept: application/json")
@POST("edeposit/2/check")
Call<ResponseBody> eDeposit(
        @Field("amount") String amount,
        @Field("currency") String currency
);
```

Another way money can be deposited into the HS account is through the M-Pesa . With the following API, a user can be able to securely deposit funds into the HS account. The API tis a post method with amount and phone number as the field parameters.

https://www.member.hsgroup.tech/api/mpesa/stk/push

```
//deposit via mpesa
@Headers("Accept: application/json")
@POST("mpesa/stk/push")
```

```
@FormUrlEncoded
Call<MpesaResponseBody> depositMobileMoney(
      @Field("amount") int amount,
      @Field("phone_number") String phonenumber);
```

HISTORIES

The histories in the HS Application are meant to let the user track the history of their transactions. The histories that can be viewed include the investment history, transaction history, and the loan history.

The following API endpoint is meant to enable a user to view his or her investment history. The method used is the GET method.

https://www.member.hsgroup.tech/api/fixed-deposit-history

```
//loan history
@GET("fixed-deposit-history")
@Headers("Accept: application/json")
Call<InvestmentHistory> getAllInvestmentHistory();
```

The following GET API endpoint will enable a user to view all the transaction histories. This will display all the transaction histories from the server which will be displayed to the user in the UI.

https://www.member.hsgroup.tech/api/transaction/history

```
//transaction history
@Headers("Accept: application/json")
@GET("transaction/history")
Call<TransactionHistory> getAllTransactions();
```

The following GET API endpoint will enable a user to view all the loan histories. This will display all the loan histories from the server which will be displayed to the user in the UI.

https://www.member.hsgroup.tech/api/loan-history

```
//get loan history
@GET("loan-history")
@Headers("Accept: application/json")
Call<LoanHistories> getLoanhistory();
```

The following GET API endpoint will enable a user to view all the E-deposit histories. This will display all the E-deposit histories from the server which will be displayed to the user in the UI.

https://www.member.hsgroup.tech/api/edeposit/history

```
//Deposit history
@Headers("Accept: application/json")
@GET("edeposit/history")
Call<DepositHistory> getDepositHistory();
```

TRANSFERS APIS

Transfers in the HS Application involves the internal and external transfers.

The following API will enable the user to transact funds internally. This API takes amount, count number, and total amount as the field parameters. The method used is POST method.

https://www.member.hsgroup.tech/api/internal/hsgroup

```
//Internal Transfers (PAY BENEFICIARY)
@FormUrlEncoded
@Headers("Accept: application/json")
@POST("internal/hsgroup")
Call<ResponseBody> internalTransfer(
        @Field("amount") String amount,
        @Field("account_no") String account_no,
        @Field("total_amount") String total_amount
);
```

The following API will enable the user to transact funds to an external account. The external transfer operation works with the help of two APIs. The first one will get the details of the user from the server then these details will be of use in the second API which will finalize the transaction.

https://www.member.hsgroup.tech/api/transfer/otherbank/confirm

```
//other bank transfer
@Headers("Accept: application/json")
@FormUrlEncoded
@POST("transfer/otherbank/confirm")
Call<OtherBankTransferResponse> transferToOtherBanks(

        @Field("bank") int bank,
        @Field("currency") String currency,
        @Field("branch") String branch,
        @Field("account_no") String account_no,
        @Field("account_holder_name") String accountHolderName,
        @Field("amount") double amount
);
```

This API is the one that finalizes the transaction and it passes the bank Id and currency to the URL using the @path method and also it passes the bank id, amount, branch, account holder's name and account number as the field parameters.

https://www.member.hsgroup.tech/api/otherbank/transfer/{bankId}/{currency

```
//Other Bank transfers call
@FormUrlEncoded
@Headers("Accept: application/json")
@POST("otherbank/transfer/{bankId}/{currency}")
Call<ResponseBody> otherBankTransferFinalize(
        @Path("bankId") String bankId,
        @Path("currency") String currency,
        @Field("bank") String bank_id,
        @Field("amount") Double amount,
        @Field("branch") String branch,
        @Field("account_holders_name") String accountHolderName,
        @Field("account_no") String account_no
);
```

PROJECT DEPENDENCIES

All the project dependencies are always recorded in the gradle scripts in android studio particularly in the build gradle file. This section is going to discuss all the project dependencies that are used in this HS Application project.

Gradle dependencies code section:

```
dependencies {
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    implementation 'androidx.fragment:fragment:1.3.0'
    def nav_version = "2.3.5"
    def lifecycle_version = "2.3.1"

    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    //android navigation component
    implementation("androidx.navigation:navigation-fragment:$nav_version")
    implementation("androidx.navigation:navigation-ui:$nav_version")


    //country code picker flag
    implementation 'com.hbb20:ccp:2.5.0'
    //Recyclerview dependency
    implementation "androidx.recyclerview:recyclerview:1.2.1"
    implementation "androidx.recyclerview:recyclerview-selection:1.1.0"
    //retrofit2 and gson converter
    implementation 'com.squareup.retrofit2:retrofit:2.5.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
    //lifecycle dependency
    // ViewModel
    implementation "androidx.lifecycle:lifecycle-viewmodel:$lifecycle_version"
    // LiveData
    implementation "androidx.lifecycle:lifecycle-livedata:$lifecycle_version"

    // okhttp
    implementation("com.squareup.okhttp3:logging-interceptor:4.9.1")
    //image slider
    implementation "com.github.smarteist:autoimageslider:1.3.9"

    //picasso image loader
    implementation 'com.squareup.picasso:picasso:2.71828'
    //sweet alert dialog
    implementation 'com.github.f0ris.sweetalert:library:1.6.2'

    //Timber
    implementation 'com.jakewharton.timber:timber:5.0.1'
```

```
    // itextG dependecy
    implementation 'com.itextpdf:itextpdf:5.3.2'

    //jackson object mapper
    implementation 'com.fasterxml.jackson.core:jackson-databind:2.8.5'
    implementation 'com.fasterxml.jackson.core:jackson-core:2.8.5'
    implementation 'com.fasterxml.jackson.core:jackson-annotations:2.8.5'

    /*Glide*/
    implementation 'com.github.bumptech.glide:glide:4.12.0'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0'

    /*Lottie*/
    implementation 'com.airbnb.android:lottie:4.2.0'
    implementation 'com.stripe:stripe-android:18.2.0'

    implementation 'com.andrognito.pinlockview:pinlockview:2.1.0'

}
```

Lifecycles view Model and live data dependencies

These dependencies goes hand in hand in the MVVM model architecture. The lifecycle is a dependency that is used to help developers to understand the state in which activities goes through when a user navigates through the application.

The live data is an observable data holder class this works together with lifecycles. They are used to set and to observe and listen to data change all through the application. The view model dependency is used to enable the implementation of MVVM architecture where the view model files will hold the business logic. The main reason why developers prefer this type of architecture is to enable ease in code maintenance whenever the code is bulky.

Junit Maven dependency

The Junit maven dependency enabled the developers to write test runs for their code with the JUnut5, some of this types of tests include the logical tests and the UI tests.

Navigation and fragment dependency

This dependency is responsible in building the navigations all through the HS Application. The navigations are developed with the help of the toolbar in the activities and the fragments. This navigation helps in provision of back arrows that looks good and gives the user a nice experience when working with the HS Application. The navigation is completed with the help of explicit intents in switching from one activity to the next.

Country code picker flag

The country code picker is a library dependency that enables the user to be able to choose different country codes including the phone number codes and the flags of the different world's countries.

Retrofit and gson converter dependency

Retrofit dependency are used in the HS Applications to call all the APIs and Gson in converting JSON responses into java objects that can be mapped into the UI and also understood by the code.

OKHTTP interceptor dependency

OKHTTP interceptor is used to display the errors that occur in the retrofit responses. These errors are then displayed in the console for viewing with the help of timber library.

Picasso image loader

This library is used here to enable the smooth loading of image in the HS Application.

Sweet alert dialogue

Sweet alert dialogue is a dialogue building dependency that allows for different varieties of dialogue builders including the success types, warning type and error type. Almost all the responses in the HS Application are are inform

Timber dependency

Timber is a logging utility class built on top of android log class. This dependency is used to display errors in the console.

Glide dependency

Glide is a fast and efficient image loading library for android focused and smooth scrolling.

Mapper dependencies

This dependency provides functionality for writing and reading JSON. This is useful because most of the responses as a result of the calling of retrofit function which returns in terms of JSON.