Onemoney SDK Integration Guide

Version 0.1

Contents

SDK Integration

To integrate Onemoney SDK in any existing app, we have to do the following steps:

- Add Onemoney SDK as a module in the project by importing the aar file.
- Declare **Module Dependency** on Onemoney SDK.

Add Onemoney SDK as a module in the project by importing the aar file:

1. Open up the project structure by right-clicking on your project and choose "**Open Module Settings**" or choose "**File**" then "**Project Structure**".
2. Select **Module** and Click the "**+**" button in the top left to add Onemoney SDK (.aar).
3. Choose "**Import .JAR or .AAR Package**" and click the "**Next**" button.
4. Click on the folder icon to locate *<**onemoneysdk**>.aar* file to be included.
5. Finally click on **Finish**.

Declare **Module Dependency** on Onemoney SDK:

Dependency can be added by adding the below line to the **'dependencies'** section of your app level **build.gradle**.

```
implementation project(path: ':onemoneysdk')
```

Alternatively, follow the steps:

1. Open up the project structure by right-clicking on your project and choose "**Open Module Settings**" or choose "**File**" then "**Project Structure**".
2. Select **Dependencies** from the left side menu, select **app** in the **Modules** section and Click on the "**+**" button in the Declared Dependencies section and select **Module Dependency**.
3. Select onemoney and click on **OK**. Then click on Apply and **OK**.
4. Done.

Add transitive Dependencies (Dependencies of Onemoney SDK - aar):

Dependency to be added to the '**dependencies**' section of your app level **build.gradle**.

```
implementation 'androidx.core:core-ktx:1.3.1'
implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.72"
implementation 'com.squareup.retrofit2:retrofit:2.6.0'
implementation 'com.squareup.retrofit2:converter-moshi:2.3.0'
implementation 'com.squareup.okhttp3:okhttp:3.12.0'
implementation "com.squareup.okhttp3:logging-interceptor:3.12.0"
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.1.1'
```

As we are adding the aar file directly as dependency to the project, the onemoney SDK dependencies are not taken care of automatically. So, we need to include them explicitly.

SDK Initiation

Instantiate Onemoney with the parameters that you receive from the developer portal.

Code Snippet:

```
Onemoney onemoney = Onemoney.init(context, organisationId,
client_id);
```

Params:

- **context** [String]: Pass context (instance of Context)
- **organisationId** [String]: The "**Organization ID**" when you register for **Onemoney** at Onemoney Developer Portal.

  example: **AWE0258**

- **client_id** [String]: To obtain this, you need to generate the **App API Keys** from the Settings section on Onemoney Developer Portal. This is an authorization parameter.

  example:

  client_id = fp_test_6562c491a05e76602eac50582fa907c20ba7045f

Once Onemoney instance is initialized, we are ready to make the API calls, by passing appropriate parameters(if required).

Onemoney APIs

List of available Onemoney APIs for consumption:

1. **User Registration - Send OTP:**

```
registerUser(name: String, mobileNo: String, termsAndConditions: Boolean, listener: UserDetailsListener)
```

2. **Verify OTP for User Registration:**

```
verifyOtpToRegisterUser(mobileNo: String, otp: String, listener: UserDetailsListener)
```

3. **Set User VUA:**

```
setVUA(vua: String, mobileNo: String, listener: UserDetailsListener)
```

4. **Initialize Session - (Obtain SessionId):**

```
initializeSession(vua: String, listener: ResultListener)
```

5. **Corporate User Registration - (Obtain SessionId):**

```
registerCorporateUser(mobileNo: String, pan: String, listener: RegisterUserListener)
```

6. **User Login - Send OTP:**

```
loginUser(mobileNo: String, listener: SessionListener)
```

7. **Verify OTP to Login User - (Obtain SessionId):**

```
verifyOtpTologinUser(mobileNo: String,  otpReference: String, code: String, listener: ResultListener)
```

8. **Get User Profile Data:**

```
getUserProfile(listener: ProfileDetailsListener)
```

9. **Get User Dashboard Data:**

```
getUserDashboard(listener: DashboardDetailsListener)
```

10. **Get all the Linked Accounts:**

```
getLinkedAccounts(listener: AccountDetailsListener)
```

11. **Get all the Discovered Accounts:**

```
getDiscoveredAccounts(listener: AccountDetailsListener)
```

12. **Account Discovery for Linking:**

```
discoverAccounts( identifiers: List<InputIdentifier>, fipId: String,
listener: AccountDetailsListener)
```

13. **Send OTP to Link Account:**

```
sendOtpToLinkAccount(account: Account, listener: AuthSessionListener)
```

14. **Verify OTP to Link Account:**

```
verifyOtpToLinkAccount( referenceNumber: String, authToken: String,
listener: ResultListener)
```

15. **Get Consent Details:**

```
getConsentDetails(consentHandles: List<String>, listener:
ConsentDetailsListener)
```

16. **Send OTP to Update (Approve/Reject) Consent:**

```
sendOtpToUpdateConsent(actionType: String, identifierType: String,
identifierValue: String, listener: ResultListener)
```

17. **Reject Consent Request:**

```
rejectConsent(consentHandle: String, otp: String, listener: ResultListener)
```

18. **Approve Consent Request:**

```
approveConsent(consentHandle: String, otp: String, accounts: List<Account>,
listener: ResultListener)
```

19. **Logout User:**

```
logoutUser(listener: ResultListener)
```

20. **Verify VUA:**

```
verifyVUA(vua: String, listener: ResultListener)
```

21. **Get list of FIPs:**

```
getFipList(listener: FipListListener)
```

User Registration

We have the following 4 steps to quickly onboard a regular user:

1. Make the `registerUser` API call with required parameters to send OTP.
2. Verify OTP by calling `verifyOtpToRegisterUser` API.
3. Set the VUA for the user by making `setVUA` API call.
4. Initialize user session by calling `initializeSession` API call.

For quickly onboarding a corporate user, make the `registerCorporateUser` API call. This will register and initialize a user session.

User Registration - Send OTP

API Signature:

```
registerUser(name: String, mobileNo: String, termsAndConditions: Boolean,
listener: UserDetailsListener)
```

Implementation:

```
onemoney.registerUser("userName", "mobileNumber", true, new
UserDetailsListener() {
   @Override
   public void onSuccess(@Nullable ResponseBody result) {
       /**TODO: MAKE "verifyOtpToRegisterUser" API call**/
   }

   @Override
   public void onFailure(@NotNull OnemoneyError onemoneyError) {
       /**TODO: Handle Error*/
   }
});
```

The purpose of this API is to register a new user with Onemoney with their name and mobile number.

Note: The user registration process completes only when you register, verify with OTP and then setvua.

Verify OTP for User Registration

API Signature:

```
verifyOtpToRegisterUser(mobileNo: String, otp: String, listener:
UserDetailsListener)
```

Implementation:

```
onemoney.verifyOtpToRegisterUser("mobileNumber", "123456", new
UserDetailsListener() {
   @Override
   public void onSuccess(ResponseBody result) {
       /**TODO: MAKE "setVUA" API call**/
   }

   @Override
   public void onFailure(OnemoneyError onemoneyError) {
       /**TODO: Handle Error*/
   }
});
```

The purpose of this API is to verify the customer's mobile number by validating OTP sent to the customer's mobile. It is one of the steps in customer's registration with Onemoney where Onemoney has to verify the customer's identity.

Set User VUA

API Signature:

```
setVUA(vua: String, mobileNo: String, listener: UserDetailsListener)
```

Implementation:

```
onemoney.setVUA(vua, mobileNumber, new UserDetailsListener() {
   @Override
   public void onSuccess(ResponseBody result) {
       /**TODO: MAKE "initializeSession" API call**/
   }

   @Override
   public void onFailure(OnemoneyError onemoneyError) {
```

```
        /**TODO: Handle Error*/
    }
});
```

The purpose of this API is to set the VUA of the customer after the customer's mobile number is verified successfully by Onemoney. Unless this API is called, the profile of the customer will not be created in Onemoney.

Initialize Session - (Obtain SessionId)

API Signature:

```
initializeSession(vua: String, listener: ResultListener)
```

Implementation:

```
onemoney.initializeSession(vua, new ResultListener() {
    @Override
    public void onSuccess(boolean result) {
        /**TODO: User session successfully obtained**/
    }

    @Override
    public void onFailure(@NotNull OnemoneyError onemoneyError) {
        /**TODO: Handle Error*/
    }
});
```

The purpose of this API is to initialize a user session, through the SDK.

Corporate User Registration - (Obtain SessionId)

API Signature:

```
registerCorporateUser(mobileNo: String, pan: String, listener:
RegisterUserListener)
```

Implementation:

```
onemoney.registerCorporateUser(mobileNumber, "CBDEQ1557A", new
RegisterUserListener() {
    @Override
    public void onSuccess(@Nullable UserRegistrationDetails result) {
```

```
        /**TODO: User session successfully obtained**/
    }

    @Override
    public void onFailure(@NotNull OnemoneyError onemoneyError) {
        /**TODO: Handle Error*/
    }
});
```

This API is intended for a consent manager to allow a corporate user to register with Onemoney

User Login and Session management

User session management is done by Onemoney SDK. In order to make user related API calls, to get user data (like User Profile Data, Dashboard Data, Linked Accounts, DIscovered Accounts, Account Linking etc.), user session needs to be initialized.

Once the user session is initialized, sessionId is stored in the SDK and used internally to make other API calls.

User Login - Send OTP

API Signature:

```
loginUser(mobileNo: String, listener: SessionListener)
```

Implementation:

```
onemoney.loginUser(mobileNumber, new SessionListener() {
   @Override
   public void onSuccess(String ref) {
       /**TODO: MAKE "verifyOtpTologinUser" API call with ref**/
   }

   @Override
   public void onFailure(@NotNull OnemoneyError onemoneyError) {
       /**TODO: Handle Error*/
   }
});
```

The purpose of this API is to validate login for a user session, through the SDK.

Verify OTP to Login User - (Obtain SessionId)

API Signature:

```
verifyOtpTologinUser(mobileNo: String,  otpReference: String, code: String,
listener: ResultListener)
```

Implementation:

```
onemoney.verifyOtpTologinUser("mobileNumber", ref, "OTP", new
ResultListener() {
```

```
    @Override
    public void onSuccess(boolean result) {
        /**TODO: Session initialized Successfully**/
    }

    @Override
    public void onFailure(@NotNull OnemoneyError onemoneyError) {
        /**TODO: Handle Error*/
    }
});
```

The purpose of this API is login otp validation for a user session, through the SDK.

Logout User

API Signature:

```
logoutUser(listener: ResultListener)
```

Implementation:

```
onemoney.logoutUser(new ResultListener() {
    @Override
    public void onSuccess(boolean result) {
        /**TODO: Session terminated Successfully**/
    }

    @Override
    public void onFailure(@NotNull OnemoneyError onemoneyError) {
        /**TODO: Handle Error*/
    }
});
```

This API is intended for a consent manager to enable a user to logout of their session.

Account Discovery and Linking

This process involves the following steps in sequence:

1. Get the list of FIPs by making the `getFipList` API call.
2. Discover user accounts using identifiers for a particular FIP. And allow the user to select accounts to link by making the `discoverAccounts` API call.
3. Send OTP for account linking by making the `sendOtpToLinkAccount` API call.
4. Verify OTP using `verifyOtpToLinkAccount` API.

Get list of FIPs

API Signature:

```
getFipList(listener: FipListListener)
```

Implementation:

```
onemoney.getFipList(new FipListListener() {
   @Override
   public void onSuccess(@Nullable FipList result) {
       /**TODO: get list of FIPs, fipId is needed in AccountDiscovery**/
   }

   @Override
   public void onFailure(@NotNull OnemoneyError onemoneyError) {
       /**TODO: Handle Error**/
   }
});
```

This API is intended for a consent manager to get the list of FIPs (Financial information providers) participating in the AA ecosystem .

Account Discovery

API Signature:

```
discoverAccounts( identifiers: List<InputIdentifier>, fipId: String,
listener: AccountDetailsListener)
```

Implementation:

```
InputIdentifier identifier = new InputIdentifier();
identifier.setCategory("STRONG");
identifier.setType("MOBILE");
identifier.setValue("1999999999");
List<InputIdentifier> identifiers = new ArrayList<>();
identifiers.add(identifier);
onemoney.discoverAccounts(identifiers, fipId, new
AccountDetailsListener() {
   @Override
   public void onSuccess(List<Account> result) {
       /**TODO: Handle List of discovered accounts based on
Identifiers**/
   }

   @Override
   public void onFailure(OnemoneyError onemoneyError) {
      /**TODO: Handle Error**/
   }
});
```

The purpose of this API is to discover a customer's account from the selected FIP based on the User's identifiers. At least one strong identifier is mandatory for account discovery. The Identifiers are categorized as STRONG, WEAK or ANCILLARY. Some of the identifier types are mobile number, CRN, account number, PAN and folio number.

Send OTP to Link Account

API Signature:

```
sendOtpToLinkAccount(account: Account, listener: AuthSessionListener)
```

Implementation:

```
onemoney.sendOtpToLinkAccount(account, new AuthSessionListener() {
   @Override
   public void onSuccess(@Nullable AuthSessionParams result) {
      /**TODO: Handle Auth Session Params and Pass result.refNumber in
"verifyOtpToLinkAccount" API Params**/
   }
```

```
    @Override
    public void onFailure(@NotNull OnemoneyError onemoneyError) {
        /**TODO: Handle Error**/
    }
});
```

This API is intended to generate a token (OTP) to initiate account linking request for customer's discovered accounts.

Verify OTP to Link Account

`API Signature:`

```
verifyOtpToLinkAccount( referenceNumber: String, authToken: String,
listener: ResultListener)
```

`Implementation:`

```
onemoney.verifyOtpToLinkAccount(refNumber, "123456", new
ResultListener() {
    @Override
    public void onSuccess(boolean result) {
        /**TODO: Successfully Linked Account**/
    }

    @Override
    public void onFailure(@NotNull OnemoneyError onemoneyError) {
        /**TODO: Handle Error**/
    }
});
```

This API is intended to authenticate the token (OTP) for account linking. If the token (OTP) is verified successfully the account gets linked.

Consent Management

These APIs will help to get consent request details and approve/reject consent requests. In the case of accepting consent, allow the user to link at least one account.

Get Consent Request Details

API Signature:

```
getConsentDetails(consentHandles: List<String>, listener:
ConsentDetailsListener)
```

Implementation:

```
List<String> consentHandles = new ArrayList<>();
consentHandles.add("be7acb9e-251e-449a-88cd-d94005c0ee5e");

onemoney.getConsentDetails(consentHandles, new ConsentDetailsListener()
{
   @Override
   public void onSuccess(@Nullable List<? extends ConsentData> result) {
       /**TODO: Handle Consent details**/
   }

   @Override
   public void onFailure(@NotNull OnemoneyError onemoneyError) {
       /**TODO: Handle Error**/
   }
});
```

This API is used by consent managers to get the consent details of a particular consent request for the user to either approve or reject the consent.

Get all the Linked Accounts

API Signature:

```
getLinkedAccounts(listener: AccountDetailsListener)
```

Implementation:

```
onemoney.getLinkedAccounts(new AccountDetailsListener() {
```

```
    @Override
    public void onSuccess(List<Account> result) {
        /**TODO: Handle The list of linked accounts. Allow users to link
accounts to approve a consent request.**/
    }

    @Override
    public void onFailure(OnemoneyError onemoneyError) {
        /**TODO: Handle Error**/
    }
});
```

This API is intended to return the list of linked accounts of the customer.

Send OTP to Update (Approve/Reject) Consent

API Signature:

```
sendOtpToUpdateConsent(actionType: String, identifierType: String,
identifierValue: String, listener: ResultListener)
```

Implementation:

```
onemoney.sendOtpToUpdateConsent("VERIFY_IDENTIFIER", "MOBILE",
"mobileNumber", new ResultListener() {
    @Override
    public void onSuccess(boolean result) {
        /**TODO: Make consent approve/reject API call**/
    }

    @Override
    public void onFailure(@NotNull OnemoneyError onemoneyError) {
        /**TODO: Handle Error**/
    }
});
```

This API is used for the consent manager to request the Onemoney AA service to send an OTP to authenticate and authorize the user's consent management (approval/rejection) action.

Reject Consent Request

API Signature:

```
rejectConsent(consentHandle: String, otp: String, listener: ResultListener)
```

Implementation:

```
onemoney.rejectConsent("e64ee795-dd3e-47bc-89c4-ac6dc4ad89a8", "otp",
new ResultListener() {
   @Override
   public void onSuccess(boolean result) {
       /**TODO: If result is true, the consent is rejected
successfully**/
   }

   @Override
   public void onFailure(@NotNull OnemoneyError onemoneyError) {
      /**TODO: Handle Error**/
   }
});
```

The purpose of this API is for the consent manager to reject the consent request.

Approve Consent Request

API Signature:

```
approveConsent(consentHandle: String, otp: String, accounts: List<Account>,
listener: ResultListener)
```

Implementation:

```
onemoney.approveConsent("be7acb9e-251e-449a-88cd-d94005c0ee5e", "otp",
       accounts, new ResultListener() {
           @Override
           public void onSuccess(boolean result) {
               /**TODO: If result is true, the consent is approved
successfully**/
           }

           @Override
           public void onFailure(@NotNull OnemoneyError onemoneyError) {
```

```
            /**TODO: Handle Error**/
        }
    });
```

The purpose of this API is for the consent manager to approve a consent request by linking one or more linked accounts with the consent.

User Details and Dashboard data

To obtain user profile information, the following APIs to be used.

Get User Profile Data

API Signature:

```
getUserProfile(listener: ProfileDetailsListener)
```

Implementation:

```
onemoney.getUserProfile(new ProfileDetailsListener() {
    @Override
    public void onSuccess(ProfileData result) {
        /**TODO: Handle Profile Data**/
    }

    @Override
    public void onFailure(OnemoneyError onemoneyError) {
        /**TODO: Handle Error**/
    }
});
```

This API is intended for a consent manager to fetch the user profile of an existing Onemoney user.

Get User Dashboard Data

API Signature:

```
getUserDashboard(listener: DashboardDetailsListener)
```

Implementation:

```
onemoney.getUserDashboard(new DashboardDetailsListener() {
    @Override
    public void onSuccess(DashboardData result) {
        /**TODO: Handle Dashboard Data (Includes list of Pending, Active,
Inactive, Rejected etc. and list of linked accounts.)**/
    }
```

```
    @Override
    public void onFailure(OnemoneyError onemoneyError) {
        /**TODO: Handle Error**/
    }
});
```

This API is intended to allow a consent manager to get a list of all consents (pending approval / active /rejected/ expired / revoked / paused) and all accounts of the user (linked / delinked).

Get all the Discovered Accounts

API Signature:

```
getDiscoveredAccounts(listener: AccountDetailsListener)
```

Implementation:

```
onemoney.getDiscoveredAccounts(new AccountDetailsListener() {
    @Override
    public void onSuccess(List<Account> result) {
        /**TODO: Handle List of Discovered accounts**/
    }

    @Override
    public void onFailure(OnemoneyError onemoneyError) {
        /**TODO: Handle Error**/
    }
});
```

This API is intended to return the list of discovered accounts.

Verify User VUA

The purpose of this API is to verify for a given mobile number whether any VUA exists or not. If VUA exists the output of the API is a status that VUA is verified else VUA not found.

Verify VUA

API Signature:

```
verifyVUA(vua: String, listener: ResultListener)
```

Implementation:

```
onemoney.verifyVUA("1999999999@onemoney", new ResultListener() {
   @Override
   public void onSuccess(boolean result) {
      /**TODO: if result is true, the VUA is valid**/
   }

   @Override
   public void onFailure(@NotNull OnemoneyError onemoneyError) {
      /**TODO: Handle Error**/
   }
});
```