SmartPin SDK - Android

Prerequisites

- You need to have an SmartPin account

To initialize you need to have a client secret, and a one time password. This one time password can only be used once to create a accesstoken. In the SmartPin app you can also revoke tokens, when you revoke a token the accesstoken will become invalid.

1. Get an accesstoken

1.1 Retrieve client secret and One Time Password

- Login in the SmartPin app as the contract owner
- Go to settings
- Go to 'Options for developers'
- Enable 'Allow API access'
- Copy the client secret
- Go to Manage access tokens
- Create a new One Time Password

1.2 Retrieve an accesstoken

In Postman (https://getpostman.com) or Paw (https://paw.cloud) create a new POST request with the following content:

The `token_inactivity_timeout` is a double, you can enter a number of days the token is valid when there is no activity on this token. The minimum 1 day, maximum 90 days. This field is optional, the default value is 7 days.

Create a basic authorization header, the username is your contractnumber, the password is your client_secret.

```
POST https://production.rspin.nl/api/v1/oauth/token
Content-Type: application/json
Accept: application/json
Authorizaton: [Basic authorization with contract and client_secret]

Content:
{
  "grant_type":"client_credentials",
  "otp":"[One Time Password]",
  "access_token_validity":7
}
```

With the call you retrieve an access_token, you can use this to initialize the SDK.

2. Initialize the SDK

To initialize the SDK you need to call:

```
 SmartPinManager.init(sandbox, token, object : SPInitListener {
        override fun onReady() {
          //SDK successfully initiated
        }

        override fun onNotReady(error: SPError) {
          // Error while initializing SDK. See the error object.
        }
})
```

`sandbox: Boolean` When `true`, the SDK will communicate with the sandbox server. Use this for development. Dont't forget to set this to `false` when releasing the final version of your app.
`accesstoken: String`
`listener`: SPInitListener

In the init call the SDK validates the SDK and access token. When the SDK is invalid (e.g. there is a newer version, and older versions are blocked) or the access token is invalid, you'll recieve an SPError.

3. Get orders

3.1 Get orders in a range

You can retrieve orders for your contract within a range of two dates. You can get the orders by calling:

```
SmartPinManager.getOrders(startDate, null, object : SPOrdersListener {
    override fun onOrdersReceived(orderList: List<SPOrder>) {
       // Retrieved the orders
    }

    override fun onOrdersFailed(error: SPError) {
        // Error retrieving the orders, see the error object
    }
})
```

`startDate: Date`
`endDate: Date` (Optional). If endDate is not present, you'll retrieve the orders up to and including today.
`listener`: SPOrdersListener

3.2 Get single order based on id

You can get a single order bases on a id by calling:

```
SmartPinManager.getOrder(orderId, object : SPOrderListener {
    override fun onOrderReceived(order: SPOrder) {
      // Retrieved the order
    }

    override fun onOrderFailed(error: SPError) {
      // Error retrieving the order, see the error object
    }
})
```

`id: Long` The id of the order you want to retrieve
`listener`: SPOrderListener

4. Get sites

You can retrieve all the sites within the contract by calling:

```
SmartPinManager.getSites(object : SPSitesListener {
    override fun onSitesReceived(sites: List<SPSite>) {
        // Retrieved the sites within the contract
    }

    override fun onSitesFailed(error: SPError) {
        // Error while retrieving the sites, see the error object
    }
})
```

`sites`: List The ID of the order you want to retrieve
`listener`: SPSitesListener

5. Get terminals

To retrieve all terminals within the SmartPin contract you call:

```
SmartPinManager.getTerminals(object : SPTerminalsListener {
    override fun onTerminalsReceived(terminalsList: List<SPTerminal>) {
        // Retrieved the terminals
    }

    override fun onTerminalsFailed(error: SPError) {
        // Error while retrieving the terminals, see the error object
    }

}, siteId)
```

`siteId: String` (Optional) The external id of the site. Leave empty if you want to retrieve all the terminals in the contract.
`listener`: SPTerminalsListener

6. Do a connection test with the terminal

You can do a connection test with the terminal to make sure the terminal is connected to the right environments. You can do this by calling:

```
SmartPinManager.testConnection(object : SPTestListener {
    override fun onTestSuccessful() {
        // Terminal connected and valid
    }

    override fun onTestFailed(error: SPError) {
        // Terminal not connected or not valid, see the error object.
    }
})
```

`listener`: SPTestListener

7. Check terminal availability

You can check if there is an available terminal:

```
val terminalConnected = SmartPinManager.isExternalDeviceAvailable()
```

8. Transactions

Implement SPPaymentListener to receive updates about transactions.

8.1 SPPaymentListener

```
interface SPPaymentListener {
    fun onPaymentSuccess(order: SPOrder)
    fun onPaymentUnknown(error: SPError)
    fun onPaymentDeclined()
    fun onPaymentError(error: SPError)
}
```

Assign this listener to `SmartPinManager.paymentListener` to receive callback events.

`order`: SPOrder
`error`: SPError

8.1.1 `fun onPaymentError(error: SPError)`

This function is called when the payment has errored. This could be for several reasons:

- The SDK is blocked
- The amount is too high or too low. (Max = 9999999, Min = -9999999)
- The payment type is invalid
- There is no terminal connected when performing an pin payment

- The connected terminal is invalid, or it does not belong to your contract.
- There can be more errors, these will be explained in the SPError.

8.1.2 `fun onPaymentSuccess(order: SPOrder)`

This function is called when the payment is approved. You'll retrieve the SPOrder back.

8.1.3 `fun onPaymentDeclined(error: SPError)`

This function is called when a payment is declined. This can happen for several reasons, for example:

- Transaction cancelled on the terminal
- Connection error halfway through the transaction

8.1.4 `fun onPaymentUnknown(error: SPError)`

Called when we don't know the status of the payment. This happens mostly when we retrieve a unknown status from the terminal.

When this happens, the payment will be synced in the backend. After a while you'll get the correct status of this payment in the 3. Get orders call.

8.2 Cash transactions

You can start a cash payment by providing the payment type CASH [SPPaymentType] and an amount

```
SmartPinManager.startPayment(
    SPPaymentType.CASH,
    amount,
    reference
)
```

`type`: SPPaymentType
`amount: BigDecimal`
`reference: String` (Optional)

Once then transaction is successfully stored, `onPaymentSuccess()` will be called.

When there is an error, `onPaymentError(error: SPError)` will be called. This can happen when there is a network issue. In this case the payment is **not** stored in the SmartPin backend. It could be that the payment was successful, but the transaction was not stored. You'll retrieve the error `paymentNotStored`. The customer has succesfully paid, and the transaction will be stored in the SmartPin backend later.

8.3 Pin transactions

You can start a cash payment by providing the payment type PIN [SPPaymentType] and an amount

```
SmartPinManager.startPayment(
    SPPaymentType.PIN,
    amount,
    reference
)
```

`type`: SPPaymentType
`amount: BigDecimal`
`reference: String` (Optional)

Once the payment is finished one of the delegate functions described in SPPaymentListener will be called.

8.4 Retour transaction

When you want to do a retour/refund transactions, make the amount send in the `startPayment()` call negative.

9. Listeners

9.1 SPInitListener

```
interface SPInitListener {
    fun onReady()
    fun onNotReady(error: SPError)
}
```

`error`: SPError

9.2 SPOrdersListener

```
interface SPOrdersListener {
    fun onOrdersReceived(orders: List<SPOrder>)
    fun onOrdersFailed(error: SPError)
}
```

`orders`: List A list with order objects
`error`: SPError

9.3 SPOrderListener

```
interface SPOrderListener {
    fun onOrderReceived(order: SPOrder)
    fun onOrderFailed(error: SPError)
}
```

`order`: SPOrder
`error`: SPError

9.4 SPSitesListener

```
interface SPSitesListener {
    fun onSitesReceived(orders: List<SPSite>)
    fun onSitesFailed(error: SPError)
}
```

`order`: List A list with site objects
`error`: SPError

9.5 SPSiteListener

```
interface SPSiteListener {
    fun onSiteReceived(site: SPSite)
    fun onSiteFailed(error: SPError)
}
```

`order`: SPSite
`error`: SPError

9.6 SPTerminalsListener

```
interface SPTerminalsListener {
    fun onTerminalsReceived(terminals: List<SPTerminal>)
    fun onTerminalsFailed(error: SPError)
}
```

`terminals`: List A list with terminal objects
`error`: SPError

9.7 SPTerminalListener

```
interface SPTerminalListener {
    fun onTerminalReceived(terminal: SPTerminal)
    fun onTerminalFailed(error: SPError)
}
```

`terminal`: SPTerminal
`error`: SPError

9.8 SPDeviceListener

```
interface SPDeviceListener {
    fun onConnecting()
    fun onConnected()
    fun onDisconnected()
}
```

9.9 SPTestListener

```
interface SPTestListener {
    fun onTestSuccessful()
    fun onTestFailed(error: SPError)
}
```

`error`: SPError

10. Objects

10.1 SPError

```
data class SPError(
    var code: ErrorCode = ErrorCode.GENERIC,
    var message: String = "" // The description of the error if available
)
```

For now there are 9 cases the ErrorCode can contain. Keep in mind that in update of the SDK, new codes can be added.

10.2 SPOrder

```
class SPOrder(
    var id: Long? //The ID of the order
    var date: String? //The date and time of the transaction
```

```
    var terminalDate: String? //The date and time of when transaction reached the
terminal
    var siteId: String? //The id of the site where this transaction was done
    var terminalId: String? //The id of the terminal that handled the transaction
    var type: SPPaymentType //An enum containing pin or cash
    var paymentMethod: String? //The payment method of this transaction (e.g.
maestro)
    var amount: BigDecimal //The amount of the transaction in cents
    var transactionId: String? //The unique transaction id if this order
    var aid: String? //The AID of the pin transaction
    var pan: String? //The truncated pan of the card holder
    var authCode: String? //The authorizationcode of the transaction
    var status: SPPaymentStatus //The status of the transaction, (Pending, failed
or success)
    var responseCode: Int? //The responsecode of the transaction
    var externalId: String? //The unique external id of this transaction
    var reference: String? //The reference of the order
)
```

10.3 SPSite

```
class SPSite (
    var id: Long?
    var externalId: String //The external id of the site
    var name: String //The external id of the site
    var terminals: List<SPTerminal> //A list of terminals belonging to this site
)
```

10.4 SPTerminal

```
class SPTerminal (
   var externalId: String
)
```

10.5 SPPaymentType

```
enum class SPPaymentType {
    PIN,
    CASH
}
```

10.6 ErrorCode

```
enum class ErrorCode {
    GENERIC, //A generic ErrorCode, see the message what the error is.
    BACKEND_OFFLINE, //The backend of SmartPin (Or internet conncetion) is
offline
    AMOUNT_INVALID, //The amount send in the payment is invalid
    REFUND_DISABLED, //You are doing a refund transaction, but refund payments
are disabled in your SmartPin contract
    CONNECTION_ERROR, //There is a connection error with the terminal
    SDK_BLOCKED, //This SDK version is blocked, please update your SDK
    INVALID_TOKEN, //The token submitted is invalid
    PAYMENT_NOT_STORED, //The payment was successful, but could not be stored in
the backend. It will be synced later
    BACKEND_OFFLINE, //There was a error in the backend, or with the internet
connection. You can see the statuscode of the network call.
}
```