Wandz.ai Android SDK Deployment Guide

Welcome to Wandz.ai!

We're thrilled to help you unlock the full potential of our adaptive AI platform. This guide will walk you through the deployment process to ensure a smooth integration with Wandz.ai.

Please follow these steps carefully for optimal results.

wandz.ai

Android SDK Version 1.0.37+

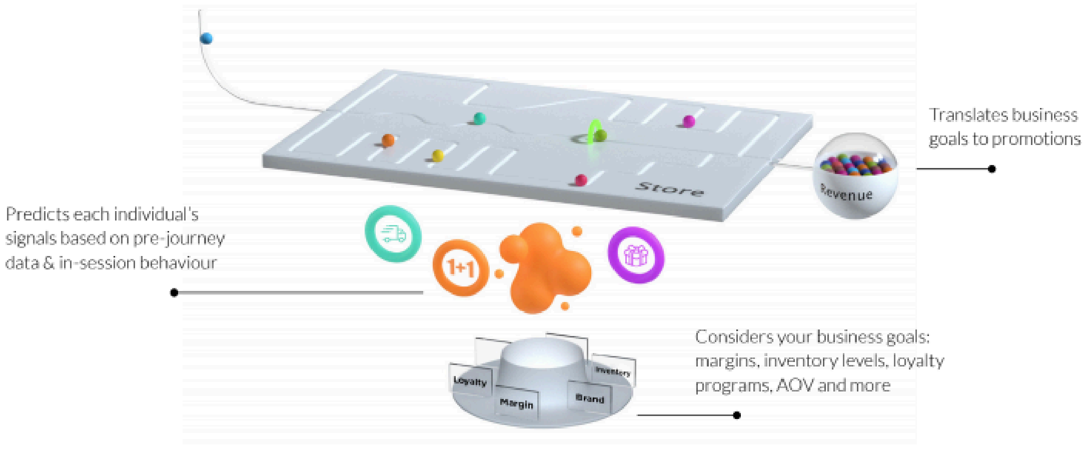
Overview

Wandz Real-Time Predictive AI

Our Predictive AI platform enhances the customer journey for numerous global online brands. Wandz.ai's innovative technology leverages in-session data, consumer-side signals, behavioral analytics, and prediction algorithms to personalize every interaction, guiding each customer smoothly to their tailored destination.

Introducing Wandz

The Wandz.ai SDK library for Android empowers developers to integrate advanced predictive AI capabilities into mobile applications effortlessly. By leveraging real-time data and machine learning models, the SDK enables personalized, in-session predictions tailored to each user's behavior. This ensures that your app delivers the right content, offers, and experiences at the optimal moment, enhancing user engagement and conversion rates.
Designed for flexibility and ease of use, the Wandz.ai SDK allows developers to incorporate predictive analytics seamlessly without extensive coding or technical overhead. The SDK integrates smoothly with existing Android applications, offering robust features to track and analyze user interactions in real time. With built-in support for GDPR and CCPA compliance, Wandz.ai ensures that user privacy is maintained while delivering high-impact, data-driven insights to optimize every step of the customer journey.


Store
Revenue
Translates business goals to promotions
Predicts each individual's signals based on pre-journey data & in-session behaviour
1+1
Loyalty
Margin
Brand
Inventory
Considers your business goals: margins, inventory levels, loyalty programs, AOV and more

Integration

The integration and role of the Wandz SDK are as follows (see Figure 1):

1. The application initializes the SDK with an API call (WandzClient.start) to verify and gather information about the associated account.
2. Once initialized, the Wandz library tracks the user behavior automatically and reports back to the Wandz AI backserver for more precise predictions.
   Note: Information that cannot be retrieved by the SDK must be passed to Wandz using the appropriate API calls that the library provides.
3. The Wandz SDK returns an outcome that predicts the user's future actions or characteristics. Additionally, if the Wandz Interactions library is enabled and display criteria that are set via the platform are met, an interactive popup will become visible to the user.

Note: Programmatic calls can be made to the SDK to start an AI prediction calculation (Example: requestPrediction).

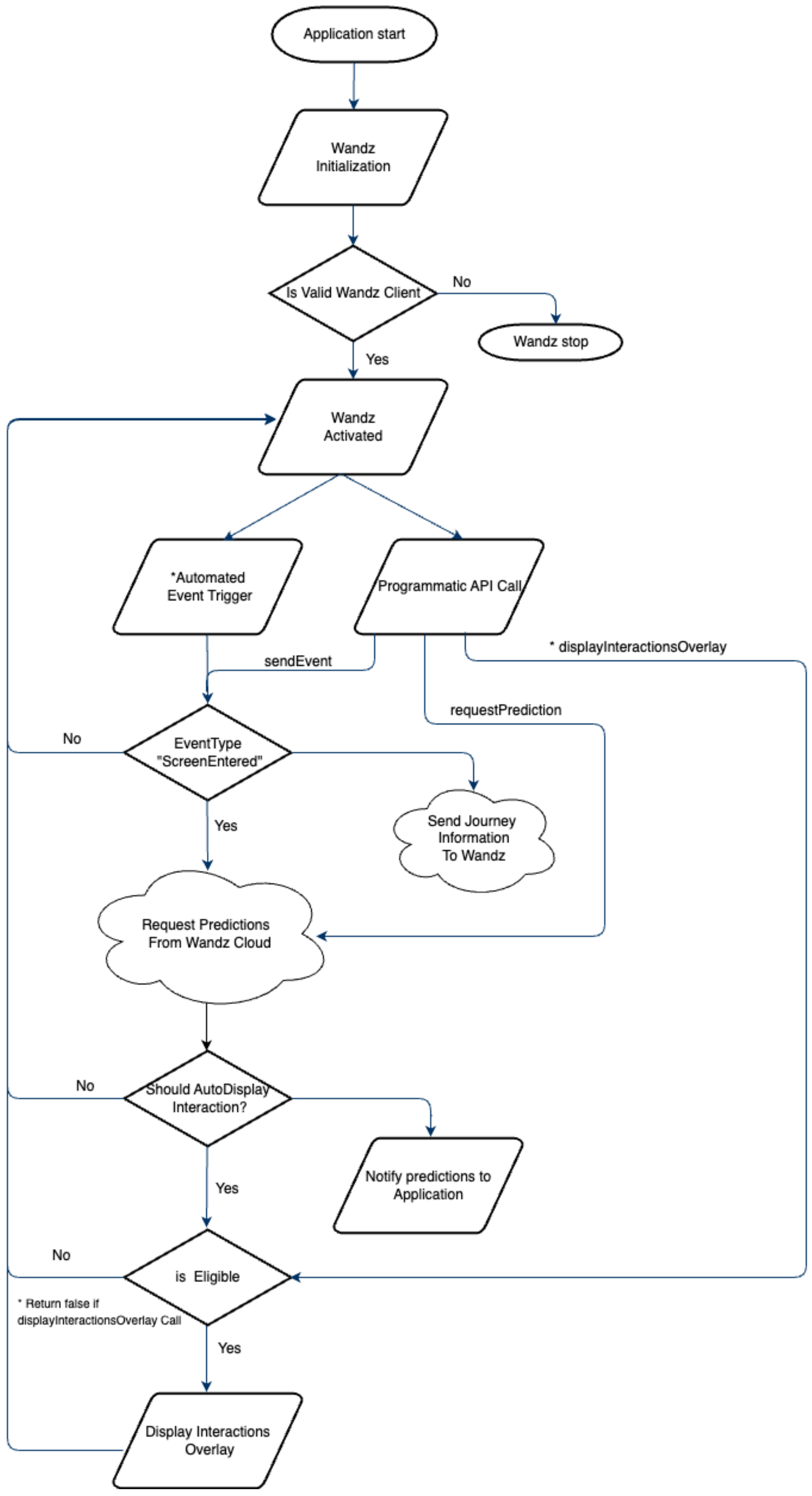

Application start
Wandz Initialization
Is Valid Wandz Client
No
Wandz stop
Yes
Wandz Activated
*Automated Event Trigger
Programmatic API Call
sendEvent
* displayInteractionsOverlay
requestPrediction
No
EventType "ScreenEntered"
Send Journey Information To Wandz
Yes
Request Predictions From Wandz Cloud
No
Should AutoDisplay Interaction?
Notify predictions to Application
Yes
No
is Eligible
* Return false if displayInteractionsOverlay Call
Yes
Display Interactions Overlay

Figure 1. Flow

Setup and installation

Basic requirements

In order to successfully install and use the Wandz SDK, you must follow these requirements:

- Minimum Android SDK version: 21
- Dependencies:
    - androidx.appcompat:appcompat:1.6.1
    - com.squareup.retrofit2:retrofit:2.9.0
    - com.squareup.retrofit2:converter-gson:2.9.0
    - com.google.code.gson:gson:2.9.1
- A Wandz account

    If you do not have a Wandz account, contact your Customer Success Manager or contact us via [our website](). Wandz provides you with an account ID (`Client Tag ID`) *to activate the SDK.*

The following instructions will help you to integrate Wandz into your application.

Adding the Wandz dependency

In Android Studio, navigate to build.gradle file at the app level and add the Wandz core dependency:

```
Android {
      defaultConfig {
            minSdkVersion 21
      }
}
dependencies {
       implementation 'ai.wandz.android:core-wandz:1.0.36'
}
```

Wandz SDK has two additional complementary libraries that can be used, “AutoCapture” & “Activate”.
The AutoCapture library captures journey events automatically, without the programmer needing to report them. These events include lifecycle, click & affinities. The second library is Activate, which holds user interaction capabilities, can display popups designed using the Wandz Platform and following the required display rules.

```
dependencies {
        implementation 'ai.wandz.android:autocapture-wandz:1.0.15'
        implementation 'ai.wandz.android:acticate-wandz:1.0.0.18'
}
```

Library definitions

The Wandz SDK has a primary object named WandzClient for interacting with the Wandz library. All interactions between the application and the Wandz library should be via the WandzClient object. When using a Wandz sub-library (activate, for example), the primary object of that library should be used (WandzActivate object, for example)

Mandatory Requirements

The following MUST be called/implemented for proper functionality:

- API entities
  - WandzClient.start
  - WandzClient.reportScreenEntered
    Cart / Checkout / Confirmation screen entered events must be reported. This provides vital information for the Wandz AI mechanism to produce predictions.

Initialization

The host application must register with a Wandz Account ID. The ID must be located under the application tag in the AndroidManifest file. For example:

```
<manifest>
    <application>
        <meta-data android:name="ai.wandz.sdk.client_id" android:value="ACCOUNT_ID"/>
        ...
    </application>
</manifest>
```

ACCOUNT_ID	The Customer ID of your Wandz account. Type: String Required

You have to initialize the Wandz SDK in the onCreate() method of your Application. Only one instance of the Wandz will be instantiated.

```
WandzClient.start(Context applicationContext);
```

In case your project uses additional Wandz libraries, they must be initialized as well in the same manner as the Core SDK.

```
WandzActivate.start(Context applicationContext);

WandzAutoCapture.start(Context applicationContext);
```

APIs

Session Properties

Setting End User IDs

In order to align your journey data with Wandz, you can set your unique end-user & session IDs with setEndUserIds api. The call should be done after a successful login via the following method.

```
WandzClient.setEndUserIds(String userId, String sessionId);
```

userId	The App internal user ID. Type: String Optional: Default Value: null
sessionId	Represents the app internal session ID. Update if changes for any reason. Type: String Optional: Default Value: null

Reporting App Referrer

If available, report to the Wandz system the package referrer & its URI. They are important in order to make more accurate predictions.

```
WandzClient.reportReferrer(String referrer, String referrerUri);
```

referrer	The package name of the 3rd party app that initiated the launch of the session. Type: String Optional: Default Value: null
referrer Uri	The Uri that was used by the 3rd party app to start the first session. Type: String Optional: Default Value: null

Listeners

The Wandz SDK uses the observer pattern listeners to push specific events to your application. If you wish to be notified, you must implement and register one of the below listeners on classes you wish to receive Wandz notifications. Once the class no longer needs to receive these notifications, you must unregister them. (In Activity/Fragment, “add” will be done in OnResume and “remove” in OnPause)

Register Prediction Listener

The values of prediction are calculated asynchronously, so in order to receive prediction information you will need to register a listener. The calculation is triggered automatically on every screen-entered event or programmatically using the getPrediction method.

```
WandzClient.registerPredictionListener(IWandzPredictionListener listener);
```

IWandzPredictionListener	The listener class must implement the predictionsCallback method that receives PredictionModel arrayList. Type: Interface

Register Audiences Listener

The audience values are calculated asynchronously therefore in order to receive the user-associated list of audiences a listener is required.

```
WandzClient.registerAudiencesListener(IWandzAudiencesListener listener);
```

IWandzAudiencesListener	The listener class must implement the audiencesCallback method that receives the Audience ArrayList. Type: Interface

Register AI Features Listener

The AI Features' values are continuously calculated asynchronously (screen/session/device levels), therefore, in order to receive their values, a listener is required. The listener will be triggered on any AI Feature value change according to the featuresFilter keys parameter.

```
WandzClient.registerAiFeaturesListener(IWandzAIFeaturesListener listener,
ArrayList<String> featuresFilter);
```

IWandzAiFeaturesListener	The listener class must implement the aiFeatureCallback method that receives key & value parameters. Type: Interface
featuresFilter	A positive list of AI feature keys you wish to listen to their values. If set to null, then all AI Features values are sent. Type: ArrayList<String> Optional: Default Value: null

Unregister Listeners

Remove listeners that were registered from the SDK. Use this method when the listeners are no longer needed. Only non-null values will be unregistered.

```
WandzClient.unregisterListeners(IWandzPredictionsListener,
IWandzAIFeaturesListener, IWandzAIFeaturesListener);

//OR

WandzClient.unregisterAllListeners();
```

Track Client Journey

The Wandz AI requires information about the client journey in order to calculate predictions. Some events are detected automatically, but additional events should be triggered manually.

Report Event

The SDK sends journey events automatically, the `reportEvent` API is used to programmatically report events. It is encouraged to use the EventType enum to notify the system about the event that took place. In the case that there is no matching event found, a custom String can be used instead.

```
WandzClient.reportEvent(EventType eventType);

//for custom event types
WandzClient.reportEvent(String eventType);
```

Parameters:

eventType	An enum representing the possible events Type: EventType / String Required

Report Screen Entered Event

The SDK requires screen change events to produce predictions. Each start of screen (activity/fragment) should report it to the library.

```
WandzClient.reportScreenEnteredEvent(ScreenType screenType, String
shortDescription, Activity currentActivity);

//for custom screen types
WandzClient.reportScreenEnteredEvent(String screenType);
```

Parameters:

screenType	An enum or String representing the event type Type: ScreenType / String Required
shortDescription	A short screen descriptor of the screen. Used in cases of ambiguous screens like category/product pages, for example. Type: String Optional

Note:
The checkout & Order Confirmation screen types are the most important to report.

Report App Paused / Resumed

Sending the application status (paused/resumed) will help create more accurate predictions.

```
WandzClient.reportAppPaused();
WandzClient.reportAppResumed();
```

Set Custom AI Feature

The SDK can receive custom AI Features On which to calculate its predictions. Once set a value the prediction mechanism will incorporate the custom AI features that were programmatically set to produce better predictions tailored to your application.

```
WandzClient.setCustomAiFeature(String feature, Object value);

//set multiple custom AI features at once
WandzClient.setCustomAiFeatures(HashMap<String, Object> aiFeatures);
```

Parameters:

feature	A unique String name that represents the AI feature in question Type: String Required
value	Basic data typed value for the custom feature. Type: Object Required

Block AutoCapture UI Scanner

The Wandz autocapture process scans the entire UI in order to capture click events and process text for affinity purposes. There are two ways to prevent the Wandz autocapture library from scanning a view and its descendants.

1. Use the blockUIScanner API call to block/allow scanning
2. Add a tag to the view and set its value to contain wandz_block_scan: true

```
WandzAutoCapture.blockUIScannel(boolean block);
```

```
<LinearLayout
  ...
  android:tag="wandz_block_scan:true">
</LinearLayout>
```

User attributes

Get AI Feature Value

Retrieves the value of a specific AI Feature synchronously.
The method will return an Object of the current value of the feature or NULL if the SDK didn't calculate the feature value.
A second attribute can be passed to cast the Value output to a specific class.

```
WandzClient.getAiFeatureValue(WandzAiFeature aiFeature, Class<T> clazz);

//get the value of a custom AI feature
WandzClient.getAiFeatureValue(String aiFeature, Class<T> clazz);
```

Parameters:

aiFeature	A WandzAiFeature or a unique String name that represents the requested AI feature. Type: WandzAiFeature / String Required
clazz	Class for the return value of the requested feature. Type: Object

Get Audiences

Returns synchronously a list of all audiences that the end user is part of.

```
WandzClient.getAudiences();
```

Request Prediction

The SDK automatically recalculates its predictions on every screen entered. If needed a prediction request can be programmatically requested.
The receive the response, a prediction listener must be registered via the registerPredictionsListener method.

```
WandzClient.requestPrediction();
```

Report Affinity

The report affinity method is used to add an affinity to the end user. It takes two parameters: affinityType, which specifies the type of affinity, and affinityValue, which specifies the value of the affinity.

```
WandzClient.reportAffinity(AffinityType affinityType, String affinityValue)
```

Parameters:

affinityType	The type of affinity that is reported Type: AffinityType Required
affinityValue	The affinity(name) that is reported Type: String

Get Affinities

Returns a list of known affinities that are associated with the user.

```
WandzClient.getAffinities()
WandzClient.getAffinities(Integer maxAffinities, AffinityType affinityType)
```

Parameters:

maxAffinitiesToReturn	The maximum number of affinities to return. If NULL, then the number of returned affinities will be determined by platform configuration. Type: Integer
affinityType	The type of affinity that is requested. If Null, then all affinity types will be returned. Type: AffinityType

Calculate Affinities

Return a list of affinities associated with the text. The text that is passed to the method is scanned for affinities according to client-specific configurations, and the list of relevant affinities is returned, or Null is not found. Additionally, the affinities are added to the end user's affinities.

```
WandzClient.calculateAffinities(String textToScan)
WandzClient.calculateAffinities(ArrayList<String> textsToScan)
```

Parameters:

textToScan	A text string to scan for affinities. Type: String

Live QA

Certain library abilities may be marked as live QA only, thus making them invisible to the public. However, they can be turned on for certain devices for QA purposes (interactions / adaptive search /...).
To switch the installed application to Wandz live QA mode, add the following file "**wandz_<APP_PACKAGE>.qa**" to the following Android directory "**/data/local/temp**". The changes will take effect after the application restarts.

```
//switch to Wandz Live QA mode
adb push <FILE_LOCATION>/wandz_<APP_PACKAGE>.qa /data/local/tmp/

//revert to regular mode
adb shell rm /data/local/tmp/wandz_<APP_PACKAGE>.qa
```

Hybrid Apps (WebView)

The Wandz library supports data journey continuation between the native and its inner webview. To ensure that the session continues to run correctly, the webview client needs to be monitored by the Wandz library. This can be achieved by one of two methods: either wrapping the web view client object completely or calling interceptWandzCommand in ShouldInterceptRequest of the app WebViewClient object.

```
//Option 1: Wrap the WebViewClient
WebViewClient wvClient = new WebViewClient() {
 …
};
webview.setWebViewClient(WandzWebViewClient.wrap(wvClient));

//Option 2: interceptWandzCommands
WebViewClient wvClient = new WebViewClient() {
   @Override
   public WebResourceResponse shouldInterceptRequest(WebView view,
WebResourceRequest request) {
      WebResourceResponse response =
WandzWebViewClient.interceptWandzCommands(request.getUrl());
      if(response != null) {
         Return response;
      }
      return super.shouldInterceptRequest(view, request);
   }
};
```

Data Types

PredictionModel

This structure is used in callbacks to get information about the client's predictions.

```
struct PredictionModel {
     guid: String
     displayName: String
     predictionScore: Double
     prediction: Boolean
}
```

guid	A unique set of characters that identify the prediction. Type: String
displayName	A human-readable string that describes the prediction. Type: String
predictionScore	The prediction percent value is 0-1. Type: Double
prediction	The Boolean value of the prediction according to its threshold. Type: Boolean

Audience

This class is used to get information about the client's associated audiences.

```
public class WandzCartItem {
    string uuid
    string name
}
```

uuid	Unique set of characters that identify an audience Type: String
name	A human-readable string that describes the audience Type: String

ScreenType Enum

The accuracy of Wandz AI is dependent on tracking the movement of the end user through the app. The accuracy of the Wandz increases the more screen types are sent when screens are changed. The following event types can be used to programmatically report screen changes to the SDK to calculate optimal predictions. This is done from the "Track Client Journey" APIs.

```
enum ScreenType
{
ACCOUNT,
CART,
CATEGORY,
CHECKOUT,
CUSTOMER_SERVICE,
ERROR_SCREEN,
HOME,
LOGIN,
ORDER_CONFIRMATION,
ORDER_DETAILS,
PRODUCT,
REGISTRATION,
SEARCH,
SPLASH,
STORES,
TRACK_ORDER
}
```

AffinityType Enum

User affinities can be of different types, in order to assist in sorting them out, all affinities are categorized into types. When reporting on any user affinity, a type must be assigned via the "reportAffinity" API.

```
enum AffinityType
{
CATEGORY,
BRAND,
COLOR
}
```

EventType Enum

The accuracy of Wandz AI is dependent on the number and variety of event types sent. The accuracy of the Wandz increases if more event types are sent. The following event types can be used to programmatically report events to the SDK to calculate optimal predictions. This is done from the "Track Client Journey" APIs.

```
enum ClientEvent
{
CLICK_CART,
CLICK_CONTACT,
CLICK_CHECKOUT,
CLICK_GIRT_CARD,
CLICK_HELP,
CLICK_LOGIN,
CLICK_LOGOUT,
CLICK_NEW_ARRIVALS,
CLICK_PRODUCT_DETAILS,
CLICK_READ_REVIEW,
CLICK_RETURN_INFO,
CLICK_SEARCH,
CLICK_SHIPPING_DETAILS,
CLICK_SIGN_UP,
CLICK_SIMILAR_PRODUCTS,
CLICK_SOCIAL_MEDIA,
CLICK_TRACK_ORDERS,
CLICK_WHICH_LIST,
CLICK_WRITE_REVIEW,
CLICK_IMAGE,
CLICK_FILTER,
ADDED_TO_CART,
REMOVED_FROM_CART,
CLICK_COUPON,
PURCHASE
}
```

WandzAiFeature Enum

This is an enum that holds basic key names used to get / set information that is stored within the library.

```
enum WandzAiFeature
{
APP_VERSION
BATTERY_IS_CHARGING
BATTERY_PERCENT
BATTERY_PERCENT_SCORE
BATTERY_OVERHEATED
BATTERY_CHARGE_TYPE
COUNTRY
COUNTRY_SIM
CPU_STRENGTH_SCORE
CPU_STRENGTH_VALUE
DAY_OF_SESSION
DEVICE_LAYOUT
DEVICE_TYPE
DEVICE
LANGUAGE_CHINESE
LANGUAGE_DUTCH
LANGUAGE_ENGLISH
LANGUAGE_FRENCH
LANGUAGE_GERMAN
LANGUAGE_SPANISH
LANGUAGE_RUSSIAN
LANGUAGE_UKRAINIAN
LANGUAGE_POLISH
LANGUAGE_ITALIAN
LANGUAGE_HEBREW
LANGUAGE_ARABIC
LANGUAGE_OTHER
APPROVED_CAMERA
APPROVED_LOCATION
APPROVED_NOTIFICATIONS
IS_DARK_MODE
IS_DAY
NETWORK_STRENGTH_SCORE
NETWORK_STRENGTH_VALUE
NUMBER_OF_SESSIONS
NUMBER_OF_PURCHASES
OS
OS_VERSION
PAGE_COUNT
PREF_LANGUAGE
PX_HEIGHT
PX_WIDTH
REFERRER
REFERRER_URI
TIME_OFF_APP_SCORE
TIME_OFF_APP_VALUE
TIME_ON_APP_SCORE
TIME_ON_APP_VALUE
TIME_ON_SCREEN_SCORE
TIME_ON_SCREEN_VALUE
TIME_SINCE_LAST_VISIT_VALUE
TIME_SINCE_LAST_VISIT_SCORE
TIME_ZONE
SCREEN_BRIGHTNESS_VALUE
SCREEN_BRIGHTNESS
AMBIENT_BRIGHTNESS_VALUE
AMBIENT_BRIGHTNESS
USER_MOBILITY
PAGE_TYPE
HOUR_IN_DAY
SCREEN_TITLE
IN_CART
IN_CHECKOUT
IN_CONFIRMATION_PAGE
CLICKS_IN_SESSION_SCORE
CLICKS_IN_SESSION_VALUE
NUMBER_OF_PRODUCT_VIEWS
PURCHASE
}
```

Sample App

To run the sample project, you will need Android Studio installed.

1. Download the sample project from the following location:

   https://github.com/namogoo/wandz-android-sample

2. Open and run in Android Studio.