Hitachi Portal Integration (Android)

Confidentiality

Version:	0.3
File Name:	Hitachi Portal Integration (Android)_v0.3
Confidentiality Category:	For Internal Use Only

Document Change Control

Version	Date	Author(s)	Description
0.1	15 July 2024	Chris Lee	Initial Version
0.2	30 July 2024	Chris Lee	Update addEventListener
0.3	6 Aug 2024	Chris Lee	Update Notify Listener

Table of Contents

1 Introduction

1.1 Overview

This document provides a comprehensive overview of the HASPortal SDK, highlighting its primary features and benefits for Android development. The SDK includes functionalities such as Wrapping, Token Interceptor, and Publish-Subscribe to enhance the overall performance, security, and scalability of the application.

Objectives:

- **Encapsulation and Security:** The SDK ensures that the portal functionality is encapsulated within a secure environment, providing better control over lifecycle events and consistent integration across different parts of the application.
- **Enhanced Security and Streamlined Authentication:** The Token Interceptor automates the management of authentication tokens, simplifying user session handling and ensuring secure communication between the app and backend services.
- **Asynchronous Communication and Decoupling:** The Pub-Sub mechanism facilitates non-blocking, event-driven communication, promoting a modular and scalable architecture by decoupling event producers and consumers.

Key Benefits:

- **Encapsulation:** Keeps portal interactions secure and consistent, with enhanced control over initialization, usage, and termination.
- **Security and Efficiency:** Token management, reducing the risk of errors and ensuring secure network communications.
- **Modularity and Flexibility:** Supports dynamic addition and removal of event listeners, improving application responsiveness and maintainability through asynchronous event handling.

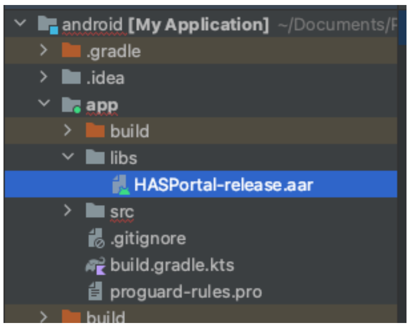
2 Specification

- Android minimum sdk: 24
- SDK Size: 27kb

3 Installation

3.1 Add the AAR Module to Your Project

- Open your project in Android Studio.
- Place the AAR file in the app/libs directory of your project.
    - Create the libs directory inside app if it does not exist.



3.2 Update Build Settings

- Open the **build.gradle / build.gradle.kts** file of your app.
- Add the following lines to the dependencies section to include the AAR module:

For build.gradle

```
dependencies {
    implementation("androidx.appcompat:appcompat:1.4.1")
    implementation("androidx.browser:browser:1.4.0")
    implementation("com.google.android.material:material:1.5.0")

    implementation files("libs/HASPortal-release.aar")
}
```

For build.gradle.kts

```
dependencies {
    implementation("androidx.appcompat:appcompat:1.4.1")
    implementation("androidx.browser:browser:1.4.0")
    implementation("com.google.android.material:material:1.5.0")

    implementation(files("libs/HASPortal-release.aar"))
}
```

4 Example Usage

Below is an example of how to use the integrated xcframework in your iOS project:

```
package com.has.myapplication

import android.content.Context
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Button
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.platform.LocalContext
import com.has.portal.Portal

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Surface(color = MaterialTheme.colorScheme.background) {
                PortalButton()
            }
        }
    }
}

@Composable
fun PortalButton() {
    val context = LocalContext.current
    val activity = context as? ComponentActivity

    Button(
        onClick = { activity?.let { openPortal(context, it, "https://example.com")
} },
        modifier = Modifier.padding(8.dp)
    ) {
        Text("Loyalty Program")
    }
}

fun openPortal(context: Context, activity: ComponentActivity, url: String) {
    val configuration: MutableMap<String, Any> = mutableMapOf(
        "accessToken" to "access token",
    )

    val portal = Portal(context, activity, configuration)
```

```kotlin
    portal.addEventListener("eventsCallbacks") { data ->
        println("Listener event data: $data")

        val name = data["name"] as? String
        val type = data["type"] as? String

        when {
            name == "redeem" && type == "navigation" -> {
                println("Handling redeem navigation event")
                portal.close()
                // Handle the redeem navigation event here
            }
            name == "close" -> {
                println("Handling close event")
                // Handle the close event here
            }
            name == "token-expired" -> {
                println("Handling Invalid Token event")
                // Handle the access token expired event here
                val newConfiguration: MutableMap<String, Any> = mutableMapOf(
                    "accessToken" to "new token",
                )
                portal.notifyListener("token-updated", newConfiguration)
            }
        }
        null
    }
    portal.open(url)
}
```

5 API

5.1 Portal

```
public constructor(context: Context, activity: Activity, configuration:
MutableMap<String, Any>)
```

Parameters

Param	Type	Additional Information
context	[String: Any]	

Param	Type	Additional Information
context	Context	The context of the application.
activity	ComponentActivity	The activity from which the portal is launched.
configuration	MutableMap<String, Any>	A mutable map containing configuration options such as `accessToken`.

Sample

```
val configuration: MutableMap<String, Any> = mutableMapOf(
    "accessToken" to "access token",
)

val portal = Portal(context, activity, configuration)
```

5.2 open

```
public void open(String urlString)
```

Parameters

Param	Type	Additional Information
urlString	String	

Sample

```
var urlString = "https://example.com"
portal.open(url)
```

5.3 close

```
public void close()
```

Sample

```
portal.close()
```

5.4 addEventListener

```
public void addEventListener(String eventName, Function<Map<String, Object>,
Map<String, Object>> listener)
```

Parameters

Param	Type	Additional Information
eventName	String	
listener	Consumer<Map<String, Object>>	

Sample

```
portal.addEventListener("eventsCallbacks") { data ->
    println("Listener event data: $data")

    val name = data["name"] as? String
    val type = data["type"] as? String

    when {
        name == "redeem" && type == "navigation" -> {
            println("Handling redeem navigation event")
            portal.close()
            // Handle the redeem navigation event here
        }
        name == "close" -> {
            println("Handling close event")
            // Handle the close event here
        }
        name == "token-expired" -> {
            println("Handling Invalid Token event")
            // Handle the access token expired event here
            val newConfiguration: MutableMap<String, Any> = mutableMapOf(
                "accessToken" to "new token",
            )
            portal.notifyListener("token-updated", newConfiguration)
        }
    }
    null
}
```

5.5 notifyListener

```
private void notifyListener(String eventName, Map<String, Object> data)
```

Parameters

Param	Type	Additional Information
eventName	String	
data	Map<String, Object>	

Sample

```
val newConfiguration: MutableMap<String, Any> = mutableMapOf(
    "accessToken" to "new token",
)
portal.notifyListener("token-updated", newConfiguration)
```