Integration between Whitelabel Application and services

Integration with web page

One possible development direction for the Verestro Whitelabel Application is to expand the set of functionalities by embedding webviews. There are no restrictions on the side of Verestro technology to embed a web page in a mobile application.

This approach has many advantages over native solutions. If the same content or features are to be available in both the web and native apps, it allows us to maintain a consistent presentation layer regardless of the platform. For content that is frequently modified this is the best solution, as there is no need to go through the entire flow release of mobile applications, and all changes can be immediately implemented on the production environment, as they are transparent to mobile applications.

It is possible that a situation may happen when communication is needed between the website embedded in the application and the mobile application itself during the flow. An example is to inform the native application that a certain stage of the process has ended, so that if it is interrupted, the user who wants to continue the process does not open the first screen but the one on which it ended.

The communication method we recommend is based on a JS event, which can be easily implemented on the web application side regardless of the technology in which the website was created, so it will work for websites created in popular frameworks like Angular or React, but also for many others.

Our mobile solutions have a consistent and universal way of building communication with the web application to provide the end user with the best possible user experience, often indistinguishable from native screens.

Events supported by default by Verestro Whitelabel Application

Event	Default handling
open	Used on part-screen webviews, which are the initialization of flow. Dispatched, for example, when the "Get started" button is pressed or when the focus is detected on the webview
close	Used at the end of a flow taking place via webview. Dispatched, for example, when the "Finish" button is pressed at the end of the flow or when the last screen in the navigation is shown.

Below is a code example for use on the web application side of the React framework.

Sample for web application - React

```
import { Event } from '../types/Global';

function isIOS(): boolean {
  return /iPad|iPhone|iPod/.test(navigator.platform);
}

function postToIOSHandler(event: Event, data: string | boolean): void {
  window.webkit?.messageHandlers?.[event].postMessage(data);
}

function postToAndroidHandler(event: Event, data: string | boolean): void {
  window.Android?.[event](data);
}

export function emitMobileEvent(event: Event, data: string | boolean): void {
  if (isIOS()) {
    postToIOSHandler(event, data);
    return;
  }
  postToAndroidHandler(event, data);
}
```

Note that due to architectural differences between Apple iOS and Google Android, two events should be emitted, each dedicated to the execution platform.
Android always requires a parameter to be passed in the event, iOS allows you to handle a "pure event" without additional parameters included in the event.

The above code was built so that first the executing device is detected i.e. iOS or Android recognition and then the correct event is emitted automatically. In a simplified version, you can always emit two events, one for iOS and the other for Android, as the corresponding application should only consume its dedicated event.

The general principle is based on the broadcast of an event by the web app and consumption by the corresponding mobile apps (iOS + Android native)

image-1719310675477.png

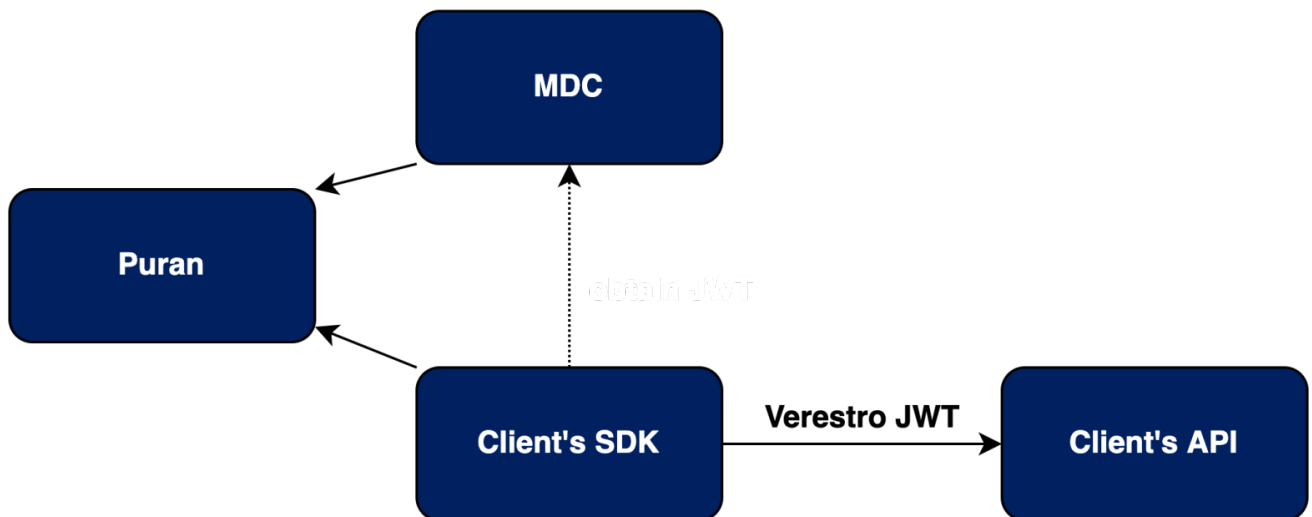An example of how to use this scheme to support webview closing

image-1719310612339.png

Creating a compatible SDKs or widgets

The mobile architecture of our systems is modular and multilayered, which enables easy and fast integration of new functionalities in the form of SDK. The creation of the appropriate SDK can be handled by Verestro's development team or it can be a customer-provided SDK. It is important that the requirements in the following subsections are met.
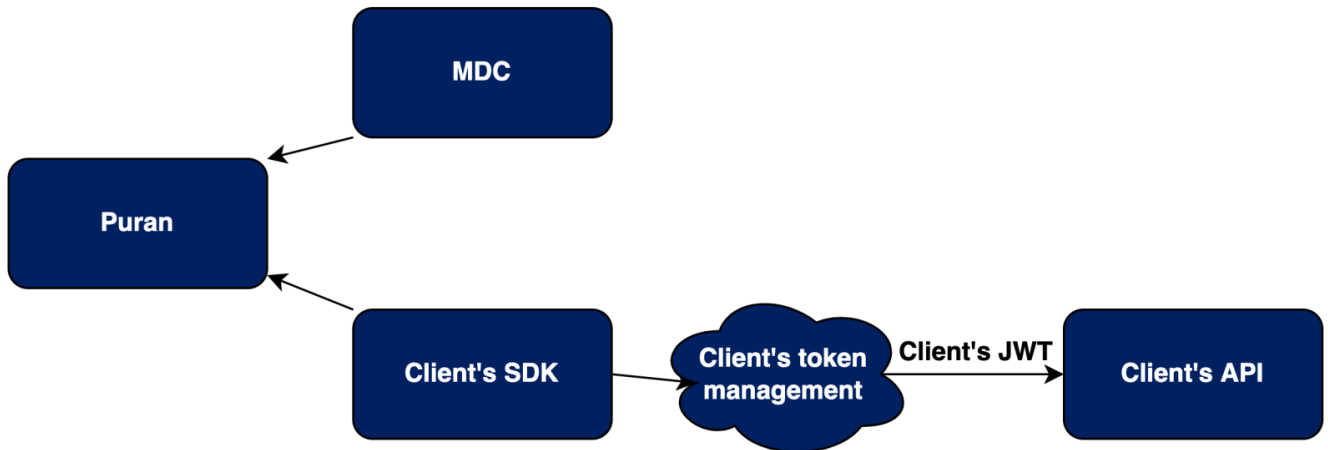
Scenario 1

Client's backend allows the
connection using JWT from Verestro

The first scenario involves using a certificate obtained from the Verestro MDC SDK to authenticate to the client server. This case requires that the servers exchange certificates with each other and the client server can verify the validity of the token. Verestro systems ensure that the token is used by a person authorized to do so and that it is that person's token (i.e., the client server does not need to verify that the user ID=2 who signed up is definitely user ID=2).

Scenario 2

Client's backend allows the
connection using own JWT

The second possible scenario makes the client server session independent of the JWT Verestro token. In this case, all communication between Client's SDK -> Client's Server relies on client security, which is transparent to the SDK operation in the application and should be implemented inside the SDK so that the session is always available and valid.

Android

- Preferred language (at least on the facade) - Kotlin,
- Exposed methods that take parameters - they should be wrapped in some model (if more parameters are added, compatibility will not be broken),
- Naming methods - any, well described in documentation,
- SDK delivery - endpoint to artifactory,
- If SDK communicates with a server - appropriate configuration and a separate endpoint to connect to the server,
- JWT - issued token parameter in SDK configuration, we pass in header.
- Minimum iOS version - 10

IOS

- Preferred language - Swift
- Exposed methods that take parameters - they should be wrapped in a model (if more parameters are added, compatibility will not be broken),
- Naming methods - any, well described in the documentation,
- Providing SDK - endpoint to be used in SPM,

- Minimum iOS version - 15

Revision #3
Created 12 July 2024 06:11:15 by Wiktor Kowalczyk
Updated 12 July 2024 06:14:35 by Wiktor Kowalczyk