Fecha del registro	*26*	*03*	*2025*

Entidad	*Aval Valor Compartido*
Responsable	*Wilson Pinto Sanchez*
Cargo	*Gerente de proyecto*

1. NOMBRE GENERAL DE LA INICIATIVA

Integración Botón Aval

2. CONTROL DE DOCUMENTACIÓN

Historial de cambios

Versión	Fecha de registro del cambio (AAAA/MM/DD)	Descripción del cambio	Responsable de aprobación	Entidad
1.0	*2025/03/20*	*Se realiza la primera versión del Manual de implementación para Botón Aval*	*Wilson Pinto Sanchez – Gerente de proyecto*	*AVC*
2.0	*2025/03/25*	*Se adiciona información consumo servicio consulta estado transacción (sonda)*	*Wilson Pinto Sanchez – Gerente de proyecto*	*AVC*
3.0	*2025/03/26*	*Se ajusta información generación comprobante de pago comercio*	*Wilson Pinto Sanchez – Gerente de proyecto*	*AVC*

Información general

Proyecto	*Integración Botón Aval*
Prioridad	***MEDIA***
¿Entidad realiza desarrollos?	***NO***
¿Entidad y/o tercero certifica?	*NO*
¿Iniciativa normativa?	*No*
Actores relevantes	*Comercios que deseen implementar el Botón Aval*

TABLA DE CONTENIDO

Contenido

3. ALCANCE DE LA INICIATIVA

3.1 OBJETIVO GENERAL

Presentar un manual con el paso a paso para la implementación de la integración del Botón Aval

3.1.1 Alcance

Este manual cubre:

- Integración del Botón Aval dentro del comercio.
- Incorporación del iframe para procesar la transacción.
- Intercambio de datos entre el comercio y el iframe mediante postMessage.
- Implementación del consumo del servicio de consulta para la generación del comprobante de transacción.

El comercio solo debe implementar el Botón Aval, insertar el iframe y enviar la información de la transacción al mismo. Toda la lógica de procesamiento y comunicación con la API de Integración Botón Avalse gestiona dentro del iframe.

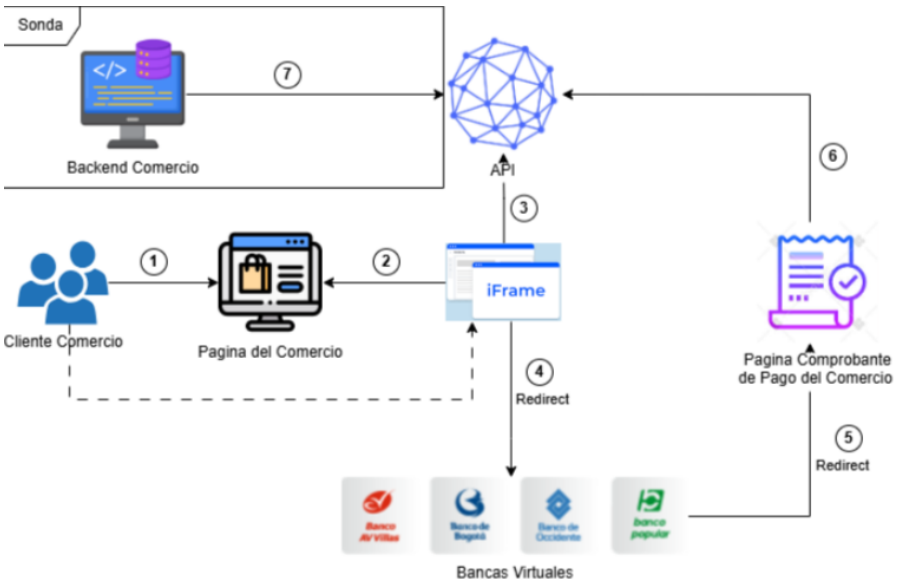
4. INTRODUCCION

Este documento proporciona las instrucciones técnicas necesarias para que un comercio implemente el Botón Aval y el iframe encargado de procesar la transacción. La implementación descrita en este manual está basada en Angular 17, pero también puede servir como referencia para su integración en otras tecnologías web.

Esta solución permite que el comercio envíe los datos de la transacción al iframe, el cual se encargará de la comunicación con la API de integración Botón aval, asegurando un procesamiento seguro y estructurado de la transacción.

5. IMPLEMENTACION EN EL COMERCIO

Los siguientes ejemplos muestran una implementación en Angular 17. Sin embargo, esta guía puede ser adaptada a otras tecnologías según las necesidades de cada comercio. Es importante ajustar la integración del iframe y el Botón Aval de acuerdo con la estructura y herramientas propias de la plataforma utilizada.



Listado de componentes a desarrollar:

- **Integración del Iframe en la página del comercio:** El iframe es el encargado de procesar la transacción. La implementación descrita en este manual está basada en Angular 17, pero también puede servir como referencia para su integración en otras tecnologías web. Esta solución permite que el comercio envíe los datos de la transacción al iframe, el cual se encargará de la

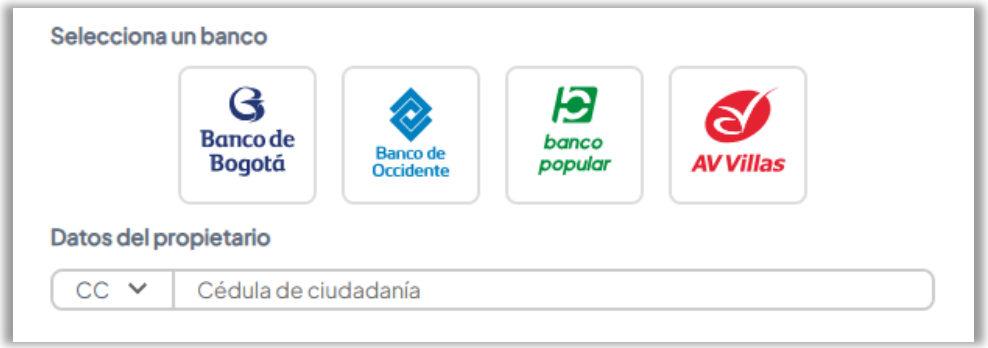
comunicación con la API de pasarela de pagos, asegurando un procesamiento seguro y estructurado de la transacción.

- **Página de resultado o comprobante de pago:** Se debe implementar una página para recibir el resultado de la transacción a partir del ID de la transacción realizar la consulta
- **Proceso de actualización de estados de transacciones (Sonda):** Es un mecanismo que se debe implementar para obtener el estado de las transacciones que quedaron pendientes en la plataforma del comercio

5.1. Incluir el componente iframe

El comercio debe agregar un elemento HTML <iframe> dentro de su portal web. Este iframe debe apuntar a la URL donde se aloja el componente encargado de gestionar el procesamiento de transacciones. Su función principal es recibir los datos de la transacción, procesarlos y comunicarse con la API de integración Botón aval.

- **Light mode**



- **Dark mode**

5.1.1 Implementación en HTML

```
<!-- Contenedor del iframe, visible solo si isVisible es true -->
<div class="iframe-container" [style.display]="isVisible ? 'block' : 'none'">
  <iframe #miIframe [src]="iframeUrl" frameborder="0"></iframe>
</div>
```

Descripción de la Implementación

- Se define un contenedor para el iframe, el cual controla su visibilidad utilizando la propiedad **display**, basada en la variable isVisible.
- Este contenedor permite aplicar estilos personalizados, como dimensiones (width y height) y márgenes, asegurando que el iframe se ajuste correctamente dentro del portal del comercio.
- El atributo **[src]** del iframe se vincula dinámicamente a la variable iframeUrl, lo que permite definir la URL del servicio de procesamiento de transacciones.
- Se establece frameborder="0" para eliminar los bordes del iframe y mejorar la integración visual dentro del comercio.

5.1.2 Lógica de Implementación en TypeScript

En la implementación del iframe, es fundamental gestionar la asignación de la URL de manera segura para prevenir vulnerabilidades relacionadas con contenido incrustado, como ataques de inyección de código malicioso.

En Angular 17, este proceso se maneja de la siguiente manera:

Definición de la Variable para la URL del Iframe

```
1 reference
iframeUrl: SafeResourceUrl; // URL segura para el iframe
```

- Se define la variable iframeUrl de tipo **SafeResourceUrl** para garantizar que la URL asignada al iframe sea segura y confiable.
- Esta variable almacenará la dirección del servicio que procesará la transacción dentro del iframe.

Asignación Segura de la URL en el Constructor

```
0 references | 1 reference
constructor(private sanitizer: DomSanitizer) {
  // Se asigna una URL segura al iframe para evitar problemas de seguridad
  this.iframeUrl = this.sanitizer.bypassSecurityTrustResourceUrl(
    "URL donde se aloja el componente"
  );
}
```

- Se inyecta **DomSanitizer** en el constructor, una herramienta proporcionada por Angular para manejar contenido potencialmente peligroso dentro de la aplicación.

- Se usa **bypassSecurityTrustResourceUrl()** para sanitizar la URL y evitar restricciones de seguridad impuestas por Angular, garantizando que el iframe pueda cargar el contenido externo sin riesgo.

5.1.3 Ejemplo de estilos CSS aplicados al contenedor del iframe

A continuación, se muestra un ejemplo de los estilos visuales que se pueden aplicar tanto al contenedor como al iframe para ajustar sus dimensiones dentro del sitio del comercio.

```
// Contenedor del iframe
.iframe-container {
  width: 90%;
  max-width: 600px;
  height: 275px;
  background-color: #ffffff;
  margin-top: -25px;

  @include for-mobile {
    height: 405px;
  }

  @media (prefers-color-scheme: dark) {
    background-color: #001119;
  }
}

// Estilos del iframe
iframe {
  width: 100%;
  height: 100%;
}
```

- Se ajustan las dimensiones y estilos del contenedor del iframe para que se adapten al estilo visual del comercio.
- Se ajusta el alto y el ancho del elemento iframe para que ocupe el 100% del espacio asignado a su contenedor.

5.1.4 Recomendaciones de diseño

Para la implementación del iframe dentro del comercio, se recomienda tener en cuenta las siguientes consideraciones:

- En dispositivos móviles, ajustar la altura del contenedor a aproximadamente 405px, o adaptarla según las necesidades específicas del comercio.

- En dispositivos de escritorio, establecer una altura de aproximadamente 275px, permitiendo modificaciones según los requisitos particulares de la integración.

Estas configuraciones garantizan una correcta visualización del iframe en diferentes dispositivos, optimizando la experiencia del usuario.

Nota Importante:

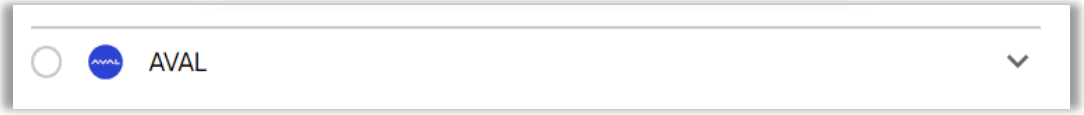
Se recomienda inyectar el iframe en el sitio del comercio desde el inicio de la carga de la página y gestionar su visibilidad a través del contenedor.

Esto ayuda a prevenir errores de funcionamiento debido a retrasos en la carga del iframe en tiempo de ejecución, mejorando la experiencia del usuario y garantizando una integración más fluida.

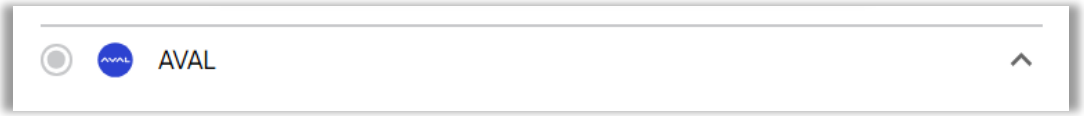
5.2 Incluir el Botón Aval

El comercio debe agregar el componente "Botón Aval", el cual se encargará de gestionar la lógica para cambiar el estado del contenedor del iframe y hacerlo visible dentro del sitio web.

- **Light mode (Close)**



- **Light mode (Open)**



- **Dark mode (Close)**

- **Dark mode (Open)**

Nota Importante:

La estructura HTML y los estilos CSS del botón serán proporcionados al comercio. No obstante, a continuación, se muestra un ejemplo de implementación en Angular 17 para ilustrar su funcionamiento y lógica de control de visibilidad.

5.2.1 Implementación en HTML

```
<div class="container">
  <label>
    <input type="radio" name="aval" (change)="showIframe()" />
    <span class="icon"></span>
    AVAL
    <span class="arrow"></span>
  </label>
</div>
```

Descripción de la Implementación

- Se define una estructura HTML con un input radio, que al cambiar su estado ejecuta una función que permite mostrar el iframe.
- La etiqueta **<label>** envuelve el contenido del botón, incluyendo elementos visuales como el icono y la flecha, para permitir el manejo de sus estilos.

5.2.2 Lógica de Implementación en TypeScript

En la implementación lógica del Botón Aval, al cambiar su estado y ejecutar la función asociada, se modifica la variable que controla la visibilidad del iframe, estableciéndola en true. Esto permite que el iframe se muestre dentro del comercio.

```
// Método para mostrar el iframe en pantalla
0 references
showIframe() {
  this.isVisible = true;
}
```

Descripción de la Lógica

- La función **showIframe()** cambia el valor de la variable isVisible a true, lo que permite que el iframe sea visible dentro del sitio web del comercio.
- Como se explicó en el punto 3.1, la visibilidad del iframe está condicionada por la variable isVisible, la cual modifica la propiedad CSS **display** de none a block, asegurando que el iframe se muestre correctamente en el sitio del comercio.

5.2.3 Estilos CSS aplicados al Botón Aval

Los siguientes estilos CSS aseguran que el Botón Aval mantenga una apariencia uniforme y accesible dentro del comercio. Se incluyen estilos responsivos, ajustes para el modo oscuro y mejoras visuales para una mejor experiencia del usuario.

5.2.3.1. Descripción de los Estilos

5.2.3.1.1. Diseño del Contenedor (.container)

```
.container {
  font-size: 14px;
  display: flex;
  padding: 10px;
  width: 100%;
  background-color: white;

  @media (prefers-color-scheme: dark) {
    background-color: #001119;
  }
}
```

- Define un fondo blanco por defecto y ajusta el **padding** para una mejor distribución del contenido.

- Aplica **background-color** dinámico según el esquema de color del sistema (**prefers-color-scheme**).

5.2.3.1.2. Estilización del label

```css
label {
  display: flex;
  align-items: center;
  gap: 10px;
  cursor: pointer;
  position: relative;
  width: 100%;
  font-family: "jakarta-regular";

  @media (prefers-color-scheme: dark) {
    color: white;
  }
}
```

- Usa **display: flex; align-items: center;** para alinear correctamente los elementos dentro del botón.

- Agrega **gap: 10px;** para espaciar los elementos internos.

- Aplica una fuente personalizada y ajusta el color en modo oscuro.

5.2.3.1.3. Línea separadora *(label::before)*

```css
label::before {
  content: "";
  position: absolute;
  top: -5px;
  left: 0;
  width: 100%;
  height: 1px;
  background-color: #c7c8ca;

  @media (prefers-color-scheme: dark) {
    background-color: white;
  }
}
```

- Se agrega una línea sutil en la parte superior del botón para una mejor separación visual.

5.2.3.1.4. Personalización del Radio Button *(input[type="radio"])*

```css
/* Estilizar el radio button para que sea visible */
input[type="radio"] {
  appearance: none;
  width: 18px;
  height: 18px;
  border: 2px solid #c7c8ca;
  border-radius: 50%;
  position: relative;
  cursor: pointer;

  @media (prefers-color-scheme: dark) {
    border: 2px solid white;
  }
}
```

- Se oculta el estilo por defecto y se personaliza con **appearance: none;**.
- Se establece un borde redondeado (**border-radius: 50%**) para que luzca como un selector visualmente atractivo.

5.2.3.1.5. Icono de selección *(input[type="radio"]:checked::before)*

```css
/* Punto dentro del radio cuando está seleccionado */
input[type="radio"]:checked::before {
  content: "";
  width: 10px;
  height: 10px;
  background-color: #c7c8ca;
  border-radius: 50%;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);

  @media (prefers-color-scheme: dark) {
    background-color: white;
  }
}
```

- Al seleccionar el botón, aparece un punto interno indicando la selección activa.

5.2.3.1.6. Iconos y Flechas (.icon y .arrow)

```css
.icon {
  background-image: url("/assets/images/favicon-aval-azul-oscuro.svg");
  background-size: 25px;
  background-repeat: no-repeat;
  width: 25px;
  height: 25px;

  @media (prefers-color-scheme: dark) {
    background-image: url("/assets/images/favicon-aval-blanco.svg");
  }
}

.arrow {
  background-image: url("/assets/icons/open.svg");
  background-size: 25px;
  background-repeat: no-repeat;
  width: 25px;
  height: 25px;
  position: absolute;
  right: 0;

  @media (prefers-color-scheme: dark) {
    background-image: url("/assets/icons/open-dark.svg");
  }
}

/* Cambiar flecha cuando el radio está seleccionado */
input[type="radio"]:checked ~ .arrow {
  background-image: url("/assets/icons/close.svg");

  @media (prefers-color-scheme: dark) {
    background-image: url("/assets/icons/close-dark.svg");
  }
}
```

- Se aplican imágenes de fondo según el estado del botón y el esquema de color del sistema.

- Se cambia dinámicamente la flecha (**open.svg → close.svg**) cuando el botón es seleccionado.

5.2.4 Recomendaciones de diseño

Para la implementación del Botón Aval, se recomienda seguir las siguientes consideraciones:

- Mantener las dimensiones del icono del botón (Logo Aval) y su contenedor en 37px con una proporción 1:1 para garantizar una apariencia uniforme.

- Utilizar la fuente **Jakarta-Regular (400)** con un tamaño de 14px para mantener coherencia visual con el diseño general.

- Ajustar el espaciado entre el botón y el iframe para asegurar una presentación visual armoniosa y una mejor experiencia de usuario.

Con esta implementación, el Botón Aval se integra de manera adecuada en el sitio del comercio, manteniendo una apariencia coherente y una funcionalidad intuitiva. La combinación de HTML, TypeScript y CSS permite garantizar una experiencia de usuario óptima, asegurando que la visibilidad del iframe se gestione de forma eficiente.

5.3 Enviar los Datos al Iframe

Para procesar la transacción, el comercio debe enviar la información necesaria al iframe. Este envío de datos se realizará cuando el usuario presione el botón Pagar, es decir, el botón que el comercio ya maneja dentro de su flujo de pago. Al hacerlo, se ejecutará una función encargada de construir y transmitir la información de la transacción al iframe.

5.3.1 Construcción del objeto de transacción:

Se debe generar un objeto **JSON** con la información relevante tanto del usuario como de la transacción. Este objeto incluirá detalles como headers, datos de la factura, impuestos y referencias necesarias para el procesamiento de la transacción.

Ejemplo de estructura del objeto de la transacción:

```
let transactionInfo =
    {
      "Credentials": {
        "ClientId": environment.credentials.clientId,
        "ClientSecret": environment.credentials.clientSecret,
      },
      "Headers": {
        "XCompanyId": "00089899",
        "XIPAddr": "10.155.111.46",
        "XAuthorization":
"dMZfG/4EX2Q=ZnJOZm1OZ1JNUUlJTCtxZGdYNmhQUzh1N3ZwRXFMQlBZZG5VWDVFVXNKakUzQkNMSmpWcV1td0RhUVowZTA
0VWZ1UWxyNGpWUTRhaWFDTTRPUEdHUkdiTXZTQWZveWkwNW1qSEJQc2tkOXo2dVNaeTVXOGxYazVxenBHd1FXK2k4ZWl1TGc
9PQ=="
,
      },
      "ProcessTransaction": {
        "Agreement": {
          "AgrmId": "00002336"
        },
        "InvoiceInfo": {
          "InvoiceNum": this.clientForm.value.paymentReference,
          "Desc": "a tu depósito electrónico",
          "PmtInfo": {
            "PmtInfoType": "3"
          }
        },
        "Fee": {
          "CurAmt": {
            "Amt": formattedAmount,
          }
        },
        "Tax": {
          "CurAmt": {
            "Amt": 10,
          }
        },
        "RefInfo": [
          {
            "RefId": "PortalURL",
            "RefType": `${environment.domain}/comprobante`
          },
          {
            "RefId": "LogoURL",
            "RefType": "https:\/\/qa.avalpaycenter.com\/imagenes\/convenios\/logo\/00405.png"
          },
        ],
      }
    }
```

Nota importante:

Los atributos como ClientId, ClientSecret, XCompanyId, XAuthorization y el objeto Agreement, se le serán suministrados al comercio para poder realizar el correcto envió del objeto de la transacción.

- **Tabla General: Estructura del Objeto de Transacción**

Esta tabla sirve como referencia rápida de la estructura general del objeto sin entrar en los detalles internos de cada propiedad.

Objeto Principal	Tipo	Descripción
Credentials	object	Credenciales del comercio en específico necesarias para el acceso al consumo de los servicios del API de integración Botón aval.

Headers	object	Headers necesarios para la autenticación y control de la transacción.
ProcessTransaction	object	Información general de la transacción a procesar.

- **Tabla 1: Credentials**

Contiene las credenciales de acceso del cliente necesarias para el consumo de los servicios de autenticación y gestión de transacciones. **(ver Anexo 1)**.

Propiedad	Tipo	Descripción
ClientId	string	Identificador único del cliente.
ClientSecret	string	Clave secreta del cliente.

- **Tabla 2: Headers**

Contiene los encabezados necesarios para la autenticación y control de la transacción.

Propiedad	Tipo	Descripción
XCompanyId	string	Empresa origen de la invocación, identificador propio del comercio.
XIPAddr	string	IP Origen
XAuthorization	string	Llave de acceso única por convenio vinculado a la Integración Boton Aval

- **Tabla 3: ProcessTransaction**

Contiene la información general de la transacción a procesar.

Propiedad	Tipo	Descripción
Agreement	object	Información del Comercio.
InvoiceInfo	object	Información de la Factura.
Fee	object	Información del costo de la factua.
Tax	object	Información de Impuestos de la factura
RefInfo	array[object]	Datos de redireccionamiento.

- **Tabla 4: Agreement**

Contiene información sobre el acuerdo comercial con el comercio.

Propiedad	Tipo	Descripción
AgrmId	string	Código de identificación del comercio, Proporcionado por ATH.

- **Tabla 5: InvoiceInfo**

Contiene información de la factura relacionada con la transacción.

Propiedad	Tipo	Descripción
InvoiceNum	String	Número de factura o referencia de pago.
Desc	String	Descripción del pago.
PmtInfo	Objeto	Información del pago.
PmtInfo.PmtInfoType	String	Tipo de pago Valores constantes (Números): • 1 = Pago CPV • 2 = Pago Normal • 3 = Pago de Obligación • 4 = Pago de Impuesto

- **Tabla 6: Fee**

Contiene información sobre las tarifas aplicadas a la transacción.

Propiedad	Tipo	Descripción
CurAmt	Objeto	Valor del Pago.
CurAmt.Amt	Número	Monto.

- **Tabla 7: Tax**

Contiene información sobre los impuestos aplicados a la transacción.

Propiedad	Tipo	Descripción
CurAmt	Objeto	Valor del Impuesto
CurAmt.Amt	Número	Monto.

- **Tabla 8: RefInfo**

Contiene información de referencias adicionales para la transacción.

Propiedad	Tipo	Descripción
RefInfo	Array de Objetos	Información adicional referencial.
RefInfo[0].RefId	String	En la primera posición del arreglo RefInfo enviar valor constante "PortalURL".
RefInfo[0].RefType	String	En la primera posición del arreglo RefInfo enviar valor Dirección web (URL) a la cual la banca virtual debe redireccionar una vez se finaliza la transacción (URL del sitio del comprobante). (Ej: "http://sitio-comercio.com/comprobante").
RefInfo[1].RefId	String	En la segunda posición del arreglo enviar enviar valor constante: "LogoURL"
RefInfo[1].RefType	String	En la segunda posición del arreglo enviar Valor que corresponde a la url del logo del convenio. (Ej: "https://qa.avalpaycenter.com/imagenes/convenios/logo/O0405.png").

5.3.2 Construcción del objeto del mensaje:

Una vez que el objeto de transacción ha sido generado con todos los datos requeridos, es necesario encapsularlo dentro de un objeto de mensaje con una estructura definida. Este objeto será el encargado de transmitir la información al iframe mediante el método de **postMessage**.

Estructura del Objeto del Mensaje

El objeto para enviar en el mensaje debe tener la siguiente estructura:

- **flag:** Una bandera que identifica el tipo de mensaje que se está enviando. El valor será **"BOTON-AVAL",** lo que indica que el mensaje corresponde a una transacción procesada a través del botón Aval.

- **message**: Un string que contiene el objeto de transacción convertido a formato string, asegurando que pueda ser interpretado correctamente por el iframe receptor.

Ejemplo del Objeto del Mensaje

```
const message = {
  flag: 'BOTON-AVAL', // Bandera que indica el tipo de mensaje enviado
  message: JSON.stringify(transactionInfo) // Datos del formulario convertidos a string
};
```

Nota Importante:

Es fundamental que la bandera (flag) tenga el valor **"BOTON-AVAL"**, ya que esto permite al iframe identificar correctamente el mensaje recibido. Asimismo, el parámetro message debe contener el objeto de transacción con la estructura requerida por el iframe, convertido a un string mediante JSON.stringify(), para garantizar su correcta interpretación.

5.3.3 Envío del mensaje al iframe:

Para que el iframe reciba y procese la información de la transacción, el comercio debe enviar un mensaje utilizando el método **postMessage** del navegador. Este mecanismo permite la comunicación segura entre el comercio y el iframe, asegurando que los datos se transmitan correctamente.

5.3.3.1. Ejemplo de implementación en Angular

La implementación en Angular sigue una serie de pasos estructurados para garantizar que el comercio pueda transmitir la información de la transacción al iframe de manera segura y eficiente.

5.3.3.1.1. Identificación del Elemento *<iframe>* en el DOM

El comercio debe identificar y referenciar en el DOM el elemento HTML **<iframe>**, previamente definido en la sección 3.1 de este documento.

```
<!-- Contenedor del iframe, visible solo si isVisible es true -->
<div class="iframe-container" [style.display]="isVisible ? 'block' : 'none'">
  <iframe #miIframe [src]="iframeUrl" frameborder="0"></iframe>
</div>
```

5.3.3.1.2. Implementación:

Para obtener una referencia al iframe, se utiliza el decorador **@ViewChild** en el componente Angular:

```
// Referencia al elemento iframe en el DOM
2 references
@ViewChild('miIframe') miIframe!: ElementRef;
```

- **@ViewChild('miIframe')** permite acceder al iframe que se ha inyectado en el DOM dentro del sitio del comercio.
- **ElementRef** proporciona acceso directo al elemento HTML, facilitando su manipulación y permitiendo la comunicación con el iframe.

5.3.3.1.3. Envío del Mensaje al iframe con postMessage

Una vez obtenida la referencia al iframe, se utiliza el método postMessage para enviar el mensaje con la información de la transacción.

5.3.3.1.4. Uso del Método postMessage:

```
// Se verifica que el iframe esté presente en el DOM y sea visible
if (this.miIframe?.nativeElement?.contentWindow && this.isVisible) {
  //Objeto de la transacción
  let transactionInfo =···
  }
  //Objeto del Mensaje
  const message = {
    flag: 'BOTON-AVAL', // Bandera que indica el tipo de mensaje enviado
    message: JSON.stringify(transactionInfo) // Datos del formulario convertidos a JSON
  };

  console.log('🔵 CLIENTE: Enviando mensaje al IFRAME', message);

  // Se envía el mensaje al iframe utilizando postMessage
  this.miIframe.nativeElement.contentWindow.postMessage(
    message, // Contiene el objeto del mensaje ({flag, message})
    "https://url-del-iframe.com" // URL donde se aloja el componente iframe
  );
}
```

- **postMessage** es un mecanismo seguro para la comunicación entre contextos de navegación distintos, como la ventana principal y el iframe.

- Este método recibe dos parámetros:

  - **Mensaje** → Contiene el objeto del mensaje construido en el punto anterior.

  - **Destino** (targetOrigin) → Debe coincidir con la URL donde está alojado el iframe, es decir la URL donde se aloja el componente que procesa la transacción.

Nota Importante:

Antes de ejecutar postMessage, se recomienda validar que el iframe esté presente en el DOM y sea visible.

¿Por qué es necesario?

- Evita errores de referencia nula: Si el iframe no ha sido renderizado en el DOM, contentWindow será null, lo que generará un error al intentar acceder a él y no será posible enviar el mensaje con la información de la transacción.

- El iframe debe estar visible ya que es necesario completar la información de la transacción por medio del formulario que el elemento contiene.

5.4 Consumo del servicio consulta transacción para generar comprobante de pago comercio

Una vez finalizado el proceso de pago en el sitio del banco, el usuario será redirigido automáticamente a la URL de retorno base (sitio del comprobante) proporcionada por el comercio en el objeto de la transacción (sección 3.3 de este documento). Como parte de esta redirección, el banco añadirá a la URL un parámetro de consulta **(pmtId)**, el cual contiene el identificador único de la transacción con el siguiente formato:

```
"http://localhost:4200/comprobante?pmtId={Id de la transacción}"
```

Ejemplo:

- **URL de retorno enviada en el objeto de la transacción:**

```
"RefInfo": [
    {
        "RefId": "PortalURL",
        "RefType": "http://localhost:4200/comprobante"
    },
```

- **URL a la que retorna luego de culminar el pago en el portal bancario.**

```
"http://localhost:4200/comprobante?pmtId={Id de la transacción}"
```

Con base en lo anterior, el comercio debe consumir el servicio de consulta de la transacción por pmtid para obtener los detalles de la transacción y generar el comprobante correspondiente. Para ello debe realizar lo siguiente:

- Extraer el **pmtId** de la URL de retorno proporcionada (es decir, la página a la que el banco redirigió al usuario tras completar el pago).
- Utilizar el identificador**(pmtId)** como parámetro en la solicitud al servicio de consulta de transacción.

Nota importante: El consumo de este servicio no solo retorna el estado de la transacción también proporciona toda la información relacionada a la misma para que el comercio pueda construir su página de comprobante y mostrársela al usuario.

A continuación, se presenta un ejemplo de implementación en Angular 17, que puede ser tomado como referencia para su integración en el comercio. Este ejemplo muestra cómo extraer el pmtId de la URL de retorno y utilizarlo para consultar la API de la integración Botón Aval, con el fin de obtener y mostrar la información del comprobante de la transacción.

5.4.1 Ejemplo de Implementación

5.4.1.1 Recuperación del parámetro de la URL

Para extraer el **pmtId** de la URL de retorno, se utiliza **ActivatedRoute** en Angular, lo que permite acceder a los parámetros de la URL de manera dinámica. Este identificador se almacena en una variable y se valida antes de realizar la consulta a la API de la integración Botón aval.

Elementos Clave de la Implementación

- **Variable de almacenamiento:** Se define una variable para almacenar el **pmtId**.

```
/** * pmtId de la transacción obtenida de la URL */
5 references
pmtId !: string;
```

- Método que rescata el parámetro **“pmtId”** de la URL del sitio.

```
/**
 * * Obtiene los parámetros de la URL y extrae el ID de la transacción.
 */
1 reference
private loadQueryParams(): void {
  this.route.queryParams.subscribe(params => {
    this.pmtId = params['pmtId'] || null;
  });
}
```

Notas Importantes:

- Validar que el pmtId se haya obtenido antes de proceder con la consulta.
- Implementar un manejo de errores en caso de que el parámetro no esté presente en la URL.

5.4.1.2 Consulta de la transacción

Una vez obtenido el **pmtId** de la URL, el comercio debe utilizarlo para realizar una solicitud a la API de la Integración Botón Aval y obtener los detalles de la transacción.

Ejemplo de Implementación en Angular 17

Se implementa un método encargado de consultar la API de la Integración Botón Avaly recuperar la información de la transacción. Este método sigue los siguientes pasos:

- Recupera el pmtId previamente obtenido de la URL de retorno.
- Realiza la solicitud HTTP al servicio de consulta de transacciones del API.
- Almacena la respuesta con los detalles de la transacción en una variable para su posterior manipulación y visualización en la UI.

Este proceso garantiza que el comercio pueda acceder a la información de la transacción de forma estructurada y segura.

Implementación del método

```
/**
 * Obtiene los detalles de la transacción desde el servicio.
 * Si la transacción es válida, almacena la respuesta y asigna la imagen correspondiente.
 * En caso de error, muestra un mensaje apropiado.
 */
1 reference
private getTransactionDetails(): void {
  if (!this.pmtId) return;

  this.isLoading = true;
  this.errorMessage = null;

  this.transactionQueryService.getTransaction(this.pmtId).subscribe({
    next: (data: TransactionResponse) => {
      if (data && data.result?.PmtId) {
        this.transactionResponse = data;
        this.setImage(); // Determina la imagen a mostrar según el estado de la transacción
      } else {
        this.errorMessage = 'No se encontraron detalles de la transacción.';
      }
      this.isLoading = false;
    },
    error: (error) => {
      console.error('❌ Error en la consulta de transacción:', error);
      this.errorMessage = 'Error al obtener los detalles de la transacción.';
      this.isLoading = false;
    }
  });
}
```

5.4.1.3 Implementación del Servicio para la Consulta de Transacciones

El servicio **TransactionQueryService** es el encargado de realizar la consulta a la API de la Integración Botón Avalpara obtener los detalles de una transacción específica. Los endpoints para el consumo de los servicios de autenticación y de consulta de transacciones están enlistados en el **Anexo 2**, donde se especifican las URL's correspondientes para los diferentes ambientes.

Descripción de la Implementación

5.4.1.4 Autenticación con la API

Antes de enviar la solicitud de consulta de transacciones, es necesario obtener un token de autenticación mediante el servicio **AuthService**. Este servicio realiza una solicitud a la API de autenticación y almacena temporalmente el token, el cual se utilizará en la consulta de transacciones.

- Se obtiene un token de autenticación utilizando el servicio AuthService.
- Este token es obligatorio para autorizar la solicitud al servicio de consulta de transacciones.

A continuación, se presenta la implementación del **servicio de autenticación**, el cual gestiona la obtención y almacenamiento del token requerido para realizar solicitudes seguras a la API de Integración Botón Aval.

Nota importante:

Los atributos clientId y clientSecret presentes en el cuerpo (body) de la petición, son los mismos atributos que se enviaron al iframe por medio de postMessage, por ende, serán los mismos a asignar en dicho body.

5.4.1.5 Código de Implementación - Servicio de Autenticación (AuthService)

```
export class AuthService {
  1 reference
  private apiUrl = `${environment.search.baseUrl}${environment.search.authenticator}`;
  5 references
  private tokenData: { response: AuthResponse, createdAt: number } | null = null;
  8 references
  private tokenRequest$: Subject<AuthResponse> | null = null;

  0 references | 1 reference
  constructor(private readonly http: HttpClient) { }

  1 reference
  private postAuth(): Observable<AuthResponse> {
    if (!this.tokenRequest$) {
      this.tokenRequest$ = new Subject<AuthResponse>();

      const body = {
        "grantType": "client_credentials",
        "clientId": environment.credentials.clientId,
        "clientSecret": environment.credentials.clientSecret,
        "scope": "scope1"
      }

      this.http.post<AuthResponse>(this.apiUrl, body, {
        headers: new HttpHeaders({
          "Content-Type": "application/x-www-form-urlencoded",
          "Accept": "application/json"
        })
      }).pipe(
        tap(resp => {
          this.tokenData = { response: resp, createdAt: Date.now() };
        })
      ).subscribe({
        next: resp => {
          this.tokenRequest$?.next(resp);
          this.tokenRequest$?.complete();
          this.tokenRequest$ = null;
        },
        error: () => {
          this.tokenRequest$?.error("Error obteniendo el token");
          this.tokenRequest$ = null;
        }
      });
    }

    return this.tokenRequest$.asObservable();
  }

  1 reference
  getToken(): Observable<AuthResponse> {
    const now = Date.now();

    if (this.tokenData) {
      const tiempoTranscurrido = now - this.tokenData.createdAt;
      const tiempoExpiracion = this.tokenData.response.result.expires_in * 1000;

      if (tiempoTranscurrido < tiempoExpiracion) {
        return of(this.tokenData.response);
      }
    }

    return this.postAuth();
  }
}
```

5.4.1.6 Parámetros requeridos para el consumo del servicio

Tabla 1: Parámetros del Body de la Solicitud

Parámetro	Tipo	Descripción
grantType	string	Tipo de autenticación solicitada. En este caso, debe ser "client_credentials".
clientId	string	Identificador único del cliente.
clientSecret	string	Clave secreta del cliente.
scope	string	Define los permisos o alcance de la autenticación. En este caso, "scope1".

Ejemplo del cuerpo de la solicitud (body) en formato JSON:

```
const body = {
  "grantType": "client_credentials",
  "clientId": environment.credentials.clientId,
  "clientSecret": environment.credentials.clientSecret,
  "scope": "scope1"
}
```

Tabla 2: Headers de la Solicitud

Header	Tipo	Descripción
Content-Type	string	Especifica el tipo de contenido enviado en la solicitud. En este caso, "application/x-www-form-urlencoded".
Accept	string	Indica el tipo de respuesta esperada de la API, "application/json".

Ejemplo de los headers de la solicitud en formato JSON:

```
{
  "Content-Type": "application/x-www-form-urlencoded",
  "Accept": "application/json"
}
```

5.4.1.7 Datos Claves sobre la Implementación

Flujo de Autenticación

- ✓ Se realiza una solicitud **POST** a la API de autenticación con el **body** definido.
- ✓ Si la solicitud es exitosa, la API responde con un token de acceso.
- ✓ El servicio **almacena temporalmente** el token y su tiempo de creación para reutilizarlo.
- ✓ Antes de realizar una nueva solicitud de autenticación, se **verifica si el token almacenado sigue siendo válido**.

Manejo de Errores

- ✓ Si la API responde con un error, el servicio captura y maneja la excepción, evitando fallos en la aplicación.

Optimización

- ✓ Se evita hacer solicitudes repetitivas, reutilizando el token hasta su expiración.

5.4.1.8 Solicitud de Información de la Transacción

Para consultar los detalles de la transacción, el comercio debe realizar una solicitud a la API de la Integración Botón Aval. Esta consulta requiere autenticación previa, por lo que primero se debe obtener un token de acceso mediante el servicio AuthService, como se explicó en la sección anterior. Una vez autenticado, se envía la petición con los encabezados requeridos y el pmtId como parámetro para obtener la información del comprobante de pago.

Descripción de la Implementación

5.4.1.8.1 Autenticación y generación del token

- ✓ Antes de enviar la solicitud de consulta de transacciones, es necesario obtener un token de autenticación mediante el servicio AuthService, como se mostró en la sección anterior.
- ✓ El token se obtiene mediante el método **getAuthToken()** y se incluye en los encabezados de la solicitud.

5.4.1.8.2 Envío de la solicitud a la API

- ✓ Se envía una solicitud **POST** a la API de Integración Botón Avalcon el **pmtId** en el cuerpo de la petición.

- ✓ Se incluyen los headers específicos.

5.4.1.8.3 Recepción y manejo de la respuesta

- ✓ Si la solicitud es exitosa, se almacena la información de la transacción.
- ✓ En caso de error, se maneja la excepción y se proporciona un mensaje adecuado.

Nota importante:

El header X-IBM-Client-Id es el mismo atributo (clientId) que se envió al iframe por medio de postMessage, por ende, será el mismos a asignar para al momento de realizar la petición.

5.4.1.8.4 Código de Implementación - Servicio de Consulta de Transacción (TransactionQueryService)

```
export class TransactionQueryService {
  1 reference
  private apiUrl = `${environment.search.baseUrl}${environment.search.query}`;

  0 references | 1 reference | 1 reference
  constructor(private http: HttpClient, private authService: AuthService) { }

  // • Obtener el token de autenticación
  1 reference
  private getAuthToken(): Observable<string> {
    return this.authService.getToken().pipe(
      map((resp: AuthResponse) => resp.result.access_token),
      catchError(error => {
        return throwError(() => new Error('Falló la autenticación'));
      })
    );
  }

  // • Método para obtener la transacción
  1 reference
  getTransaction(pmtId: string): Observable<TransactionResponse> {
    return this.getAuthToken().pipe(
      switchMap(token => {

        let body = {
          pmtId
        }

        const headers = new HttpHeaders({
          'Authorization': `Bearer ${token}`,
          'X-RqUID': '10305290',
          'X-Channel': '3',
          'X-GovIssueIdentType': 'CC',
          'X-IdentSerialNum': '234322432243',
          'X-IBM-Client-Id': environment.credentials.clientId,
          'X-IPAddr': '10.155.111.46',
        });

        return this.http.post<TransactionResponse>(this.apiUrl, body, { headers });
      }),
      catchError(error => {
        return throwError(() => new Error('Falló la transacción'));
      })
    );
  }
}
```

Parámetros requeridos para la consulta

Tabla 1: Parámetros del Body de la Solicitud

Parámetro	Tipo	Descripción
pmtId	string	Identificador único de la transacción.

Ejemplo del cuerpo de la solicitud (body) en formato JSON:

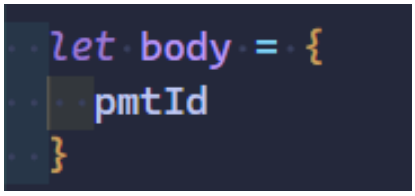


Tabla 2: Headers de la Solicitud

Header	Tipo	Descripción
Authorization	string	Token de autenticación obtenido previamente.
X-RqUID	string	Identificador único de la solicitud.
X-Channel	string	Canal desde donde se realiza la transacción.
X-IPAddr	string	Dirección IP del usuario que realiza la transacción.
X-IBM-Client-Id	string	Identificador del cliente en la API de integración Botón aval.
X-GovIssueIdentType	string	Tipo de documento de identificación del usuario.
X-IdentSerialNum	string	Número de documento de identificación del usuario.

Ejemplo de los headers de la solicitud en formato JSON:

```
const headers = new HttpHeaders({
  'Authorization': `Bearer ${token}`,
  'X-RqUID': '10305290',
  'X-Channel': '3',
  'X-GovIssueIdentType': 'CC',
  'X-IdentSerialNum': '23432243243',
  'X-IBM-Client-Id': environment.credentials.clientId,
  'X-IPAddr': '10.155.111.46',
});
```

Flujo de la Implementación

- ✓ Se obtiene el **pmtId** de la URL de retorno.
- ✓ Se solicita un token de autenticación con AuthService.
- ✓ Se construye la solicitud con el pmtId y los headers requeridos.

- ✓ Se envía la petición a la API de consulta de transacciones.
- ✓ Se maneja la respuesta de la API y se almacena la información de la transacción.
- ✓ En caso de error, se captura y se muestra un mensaje adecuado.

Notas Importantes

- ✓ Validar que el **pmtId** esté presente antes de hacer la solicitud.
- ✓ Manejar errores adecuadamente en caso de fallos en la API.
- ✓ Se debe asegurar que el token de autenticación es válido antes de enviar la solicitud.

5.4.2 Generación del comprobante en la UI

Una vez obtenidos los detalles de la transacción mediante la consulta a la API de la integración Botón aval, el comercio debe procesar y presentar esta información en la interfaz de usuario (UI).

Recepción de los datos de la transacción:

- La respuesta de la API se captura y almacena en una variable para su procesamiento.
- Se estructura la información para que pueda ser mostrada de manera clara y comprensible en la UI.

Renderización de los datos en la UI:

Se crea un componente encargado de recibir y presentar la información de la transacción en la interfaz del usuario.

Se presentan los datos clave como:

- Estado de la transacción.
- Identificador de pago **(pmtId).**
- Información del cliente.
- Fecha y hora de la transacción.
- Monto del pago.
- Método de pago.
- Entidad financiera, entre otros.

5.4.2.1 Manejo de errores y validaciones:

- ✓ Se debe validar que los datos se hayan recibido correctamente antes de mostrarlos en la UI.
- ✓ En caso de que la consulta a la API falle, se debe manejar el error y mostrar un mensaje informativo al usuario.

A continuación, se presenta un ejemplo de implementación en Angular 17, que puede servir como referencia para integrar la generación del comprobante en el comercio.

5.4.2.2 Captura de la respuesta del servicio

```
/**
 * 🔹 Obtiene los detalles de la transacción desde el servicio.
 * Si la transacción es válida, almacena la respuesta y asigna la imagen correspondiente.
 * En caso de error, muestra un mensaje apropiado.
 */
1 reference
private getTransactionDetails(): void {
  if (!this.pmtId) return;

  this.isLoading = true;
  this.errorMessage = null;

  this.transactionQueryService.getTransaction(this.pmtId).subscribe({
    next: (data: TransactionResponse) => {
      if (data && data.result?.PmtId) {
        this.transactionResponse = data; // Almacena la respuesta del servicio en una variable
        this.setImage(); // Determina la imagen a mostrar según el estado de la transacción
      } else {
        this.errorMessage = 'No se encontraron detalles de la transacción.';
      }
      this.isLoading = false;
      this.loader = false;
    },
    error: (error) => {
      console.error('❌ Error en la consulta de transacción:', error);
      this.errorMessage = 'Error al obtener los detalles de la transacción.';
      this.isLoading = false;
    }
  });
}
```

5.4.2.3 Estructura de la interfaz del comprobante (HTML)

```
<apc-loading *ngIf="loader"></apc-loading>
<div class="container-main">
  <!-- Título principal del comprobante de pago -->
  <h2 class="title">Comprobante de Pago</h2>
  <div class="container">
    <section class="section-one">
      <div class="voucher-container">
        <!-- Mensaje de carga -->
        <p class="loading-message" *ngIf="isLoading">Cargando datos de la transacción ... </p>

        <!-- Mensaje de error si no se encuentra la transacción -->
        <p *ngIf="!isLoading && errorMessage" class="error-message">{{ errorMessage }}</p>

        <!-- Datos de la transacción (solo si transactionResponse tiene datos) -->
        <div *ngIf="!isLoading && transactionResponse">
          <p><strong>Estado: </strong>{{ transactionResponse.result.TrnState.Desc }}</p>
          <p>
            <strong>Nombre del Cliente: </strong>
            {{ transactionResponse.result.PersonInfo.PersonName.FirstName +
            (transactionResponse.result.PersonInfo.PersonName.MiddleName ? ' ' +
            transactionResponse.result.PersonInfo.PersonName.MiddleName : '') +
            ' ' + transactionResponse.result.PersonInfo.PersonName.LastName }}
          </p>
          <p><strong>Email del Cliente: </strong>
            {{transactionResponse.result.PersonInfo.ContactInfo.EmailAddr?.toLocaleLowerCase()}}</p>
          <p><strong>Fecha de Transacción: </strong> {{ transactionResponse.result.TrnState.EffDt }}</p>
          <p><strong>Referencia Interna: </strong> {{ transactionResponse.result.PmtId }}</p>
          <p><strong>Monto Total: </strong> $ {{ transactionResponse.result.Fee.CurAmt.Amt }} {{
            transactionResponse.result.Fee.CurAmt.CurCode }}</p>
          <p><strong>Método de Pago: </strong> {{ transactionResponse.result.PmtType === '2' ? "Botón Aval"
            : ""}}</p>
          <p><strong>Entidad Financiera: </strong> {{transactionResponse.result.RefInfo[0].RefType}}</p>
        </div>
        <!-- Botón para regresar a la página principal del comercio -->
        <button class="button" (click)="back()">Regresar</button>
      </div>
    </section>
    <!-- Sección de la imagen con loader -->
    <section class="section-two">
      <!-- Loader mientras carga -->
      <div class="skeleton-loader" *ngIf="isLoading"></div>
      <!-- Mostrar la imagen cuando termine de cargar -->
      <img *ngIf="!isLoading" class="image" [src]="imageSrc" alt="Estado de la transacción">
    </section>
  </div>
</div>
```

5.4.2.4 Ejemplo de UI del Comprobante de Pago

Nota Importante:

Los estilos del comprobante no están predefinidos y pueden ser implementados por el comercio de acuerdo con su política de diseño y UI. Se recomienda mantener una estructura clara y accesible para mejorar la experiencia del usuario.

5.5 Consumo del servicio de consulta transacción para actualización de estado transacción (sonda)

Una vez generado el comprobante de pago el comercio debe realizar el consumo del servicio de consulta de la transacción por pmtid durante un tiempo de 50 minutos con frecuencia de 2 minutos con el objetivo de obtener el estado final de la transacción culminando así el flujo.

El consumo del servicio debe realizarse como se mostro en la sección del apartado 5.4. del presente documento.

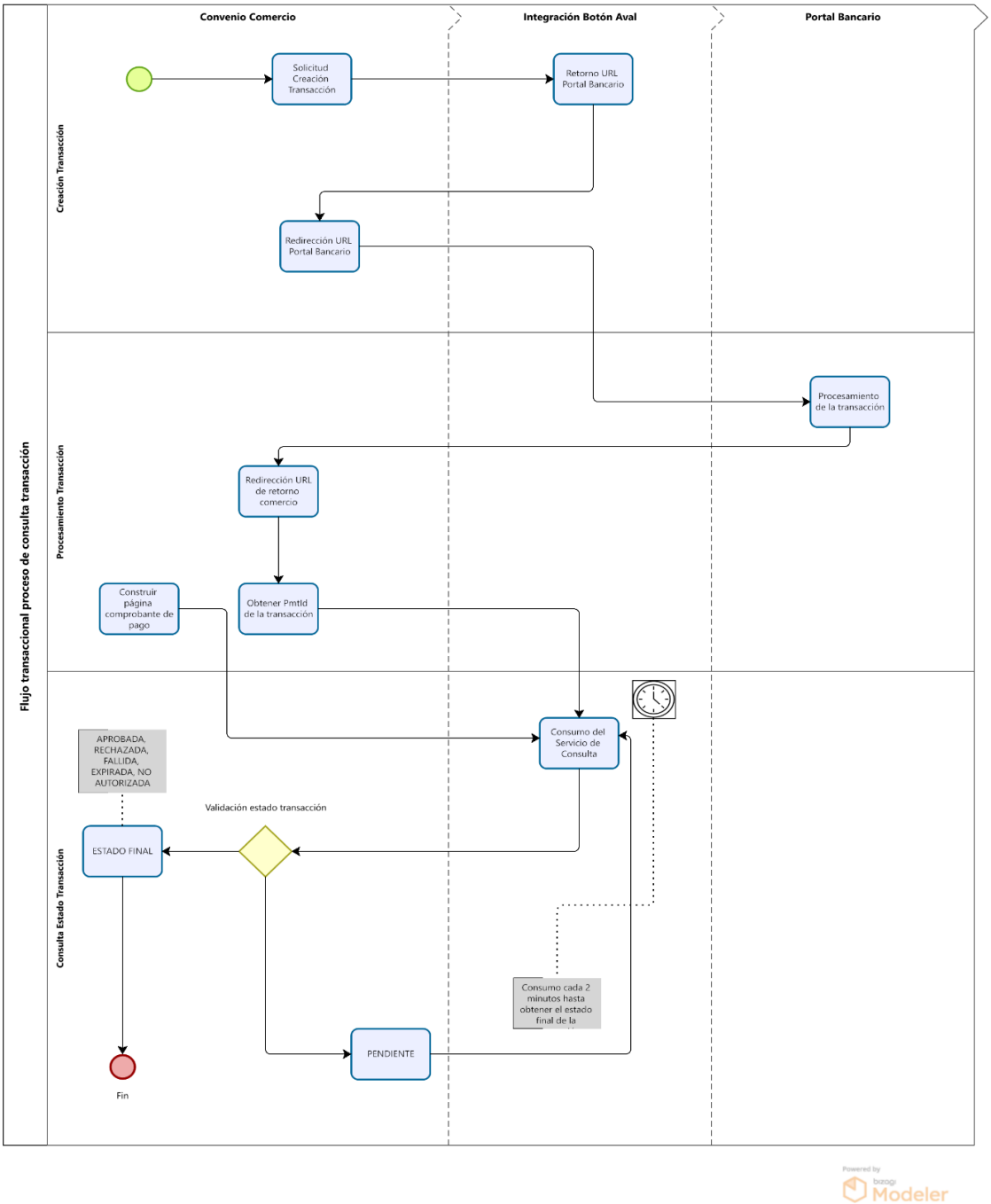
5.5.1 Precondiciones para el consumo del servicio de consulta de la transacción

- ✓ Obtener el identificador de la transacción (pmtid).
- ✓ Realizar el consumo de la operación oauth2-token para obtener el Access Token Oauth2.0.
- ✓ Cuando se retorna a la URL del comercio/convenio se debe consumir el servicio. Si el estado entregado es **PENDIENTE**, debe consultar cada **2 minutos durante 50 minutos** hasta que se obtenga un estado final (APROBADA, RECHAZADA, FALLIDA, EXPIRADA, NO AUTORIZADA).

5.5.2 Información técnica servicio consulta transacción

Información Técnica del Servicio	
Tipo de Servicio	REST / JSON
Método	GET
Frecuencia de Ejecución	A solicitud
Frecuencia de consumo	Cada 2 minutos durante 50 minutos.
Observaciones	Solo permite obtener el resultado de una transacción. La Integración con el botón Aval tiene un tiempo estimado de 50 minutos para obtener el estado de una transacción pendiente una vez se encuentre realizado el pago, por lo tanto, el comercio/convenio debe tener en cuenta este tiempo para obtener el estado final de la transacción. Las transacciones que ya fueron pagadas pueden quedar pendientes por los siguientes motivos: ✓ El cliente – usuario final no finaliza el pago. ✓ La respectiva entidad es decir el banco, aún se encuentra con la transacción pendiente. En caso de que el cliente – usuario final inicie el flujo de la transacción, pero no realice ninguna acción en el portal bancario, la transacción se encontrará pendiente hasta que lo cierre el respectivo proceso de Integración en un tiempo estimado de 12 minutos después de creada la transacción.

5.5.3 Flujo transaccional proceso de consulta transacción



6. Conclusión

La implementación del **Botón Aval** y el **iframe** en el comercio permite una integración eficiente y segura con la API de integración Botón aval. Este manual proporciona una guía detallada para su correcta implementación en Angular 17, pero puede adaptarse a otras tecnologías según las necesidades del comercio.

A lo largo del documento, se ha explicado cómo incluir estos elementos dentro del comercio, gestionar la visibilidad del iframe y garantizar la comunicación segura mediante postMessage. Con esta integración, se optimiza la experiencia del usuario, asegurando un flujo transaccional claro y estructurado.

7. Recomendaciones y Buenas Prácticas

- ✓ **Validaciones de datos:** Antes de enviar la transacción, asegurarse de que los datos del usuario y la transacción estén correctamente estructurados.
- ✓ **Manejo de errores:** Implementar mecanismos de control para detectar y manejar posibles errores en la comunicación entre el comercio y el iframe.
- ✓ **Pruebas previas a producción:** Validar la integración en un ambiente de pruebas antes de su despliegue en producción para garantizar su correcto funcionamiento.

8. Anexos

Anexo 1: Tabla de credenciales para el consumo de los servicios

Ambiente	Credencial	Valor
Desarrollo (PT)	clientId	
	clientSecret	
Calidad (QA)	clientId	f7bb7cb8bfa9bc05cb92bad1e5140fde
	clientSecret	591c25448fbd5e165353ced1d1254975
Producción (PR)	clientId	156f38ffcff633a5ca0221091501f0b2
	clientSecret	5bfa9a24171ad003fa04b2d57f0a5bd7

Anexo 2: Tabla de EndPoints para el consumo de los servicios

Ambiente	EndPoint Autentificación	EndPoint Transacción

Desarrollo (PT)	https://nq2anjoqs9.execute-api.us-east-1.amazonaws.com/pt/boton-aval/auth	https://nq2anjoqs9.execute-api.us-east-1.amazonaws.com/pt/boton-aval/getTransaction
Calidad (QA)	https://1sk0k84ucb.execute-api.us-east-1.amazonaws.com/qa/boton-aval/auth	https://1sk0k84ucb.execute-api.us-east-1.amazonaws.com/qa/boton-aval/getTransaction
Producción (PR)	https://mvfwlyrwt7.execute-api.us-east-1.amazonaws.com/prd/boton-aval/auth	https://mvfwlyrwt7.execute-api.us-east-1.amazonaws.com/prd/boton-aval/getTransaction

Anexo 3: Archivo .zip con los archivos guía para la implementación de componente "Botón Aval"