Pasos para implementar la integración con WebPay

1. Instalar la librería de transbank en el backend

`pip install transbank-sdk`

En este caso se añadió al archivo de requirements que se instalará al utilizar *docker-compose*

2. Backend (FastAPI) – Implementación de endpoints

2.1. Crear el endpoint de creación de transacción

Ruta: `POST /webpay/create`

- Recibe: amount
- Genera: un transaction_id corto y único (máx 26 caracteres)
- Guarda la orden en la base de datos con estado "PENDING"
- Llama a Transbank para crear la transacción
- Retorna al frontend el url de redirección y el token_ws

```python
@app.post("/webpay/create")
def iniciar_webpay(data: dict, user=Depends(verify_token)):
    user_id = user["sub"]
    user = users_collection.find_one({"correo": user_id})
    if not user:
        return {"error": "Usuario no encontrado."}
    stock = collection.find_one({"symbol": data["symbol"]})
    if not stock:
        return {"error": f"Stock con símbolo {data['symbol']} no encontrado."}
    if stock["quantity"] < data["quantity"]:
        return {"error": "No hay suficientes acciones disponibles."}

    id = uuid.uuid4()

    b64 = base64.urlsafe_b64encode(id.bytes).decode('utf-8').rstrip('=')
    transaction_id = b64[:26]

    request_id = str(id)
    amount = round(float(data["amount"]))

    # Build the frontend payment URL from the environment variable
    frontend_payment_url = f"{URL_FRONTEND}/payment" if URL_FRONTEND else "https://www.arquitecturadesoftware.me/payment"
    trx_resp = tx.get_tx().create(transaction_id, "stocks_buy", amount, frontend_payment_url)

    mqtt_manager.publish_buy_request(request_id, data["symbol"], data["quantity"], trx_resp["token"])
    mqtt_manager.publish_validation(request_id, "ACCEPTED", trx_resp["token"])
    transaction = {
        "request_id": request_id,
        "transaction_id": transaction_id,
        "token_ws": trx_resp["token"],
        "symbol": data["symbol"],
        "quantity": data["quantity"],
        "user_email": user_id,
        "timestamp": datetime.utcnow(),
        "status": "PENDING",
    }

    transactions_collection.insert_one(transaction)
    return {"url": trx_resp["url"], "token_ws": trx_resp["token"], "request_id": request_id}
```

Figura 1. Endpoint de creación de transacción de la API

2.2. Crear el endpoint de confirmación de transacción

Ruta: POST /webpay/commit

- El frontend llama este endpoint tras recibir el token de retorno de WebPay tras el intento de pago.
- Si token_ws está presente:
    - Se llama a commit(token_ws) para validar el pago.
    - Se actualiza el estado de la orden (OK, REJECTED).
    - Retorna al frontend el estado de la transacción (status).
- Si no hay token_ws:
    - Se trata como cancelación del usuario.
    - Retorna al frontend el status=CANCEL.

```python
@app.post("/webpay/commit")
async def commit_transaction(request: Request, user=Depends(verify_token)):
    body = await request.json()
    token_ws = body.get("token_ws")
    transaction = transactions_collection.find_one({"request_id": body.get("request_id")})

    # TRANSACCIÓN ANULADA POR EL USUARIO
    if not token_ws or token_ws == "":
        mqtt_manager.publish_validation(body.get("request_id"), "REJECTED", transaction["token_ws"]) #Token enviado cuando
        return {"status": "ANULLED",
                "message": "Transacción anulada por el usuario."}

    try:
        response = tx.get_tx().commit(token_ws)

        response_status = "OK" if response["response_code"] == 0 else "REJECTED"
        mqtt_manager.publish_validation(body.get("request_id"), response_status, transaction["token_ws"])

        # TRANSACCIÓN RECHAZADA
        if response["response_code"] != 0:
            return {"status":"REJECTED",
                    "message": "Transacción ha sido rechazada.",
                    }

        # TRANSACCIÓN ACEPTADA
        #Generación de boleta
        receipt_url = purchase_receip.generate_receipt(···
        # modificar transaccion con receipt_url
        transactions_collection.update_one(···

        return {"status": "OK",
                "message": "Transacción ha sido autorizada.",
                "transaction_id": response["buy_order"],
                "receipt_url": receipt_url}

    except Exception as e:
        print("Error en la transacción:", e)
        return {"status": "ERROR",
                "message": "Error en la transacción."}
```

Figura 2. Endpoint de confirmación de transacción de la API

3. Frontend (React en S3/CloudFront)

3.1. Al presionar el botón “Comprar con Web Pay”

Se añadió un botón "Comprar con WebPay" que, al ser presionado, ejecuta la función `handleWebpayBuy`. Esta función invoca a `webpayStock` del servicio `stocksService`, el cual realiza un `POST` a la ruta `/webpay/create` de la API de stocks.

Dicha ruta retorna un `token_ws` y una URL de redirección hacia la plataforma de pago de WebPay. El usuario es redirigido automáticamente a esta URL para completar el proceso de pago.

```
<button
  onClick={handleWebpayBuy}
  disabled={loading}
  className="w-full bg-blue-600 text-white font-medium py-3 px-4 rounded-md hover:bg-blue-700 disabled:opacity-50 text-lg"
>
  {loading ? 'Procesando...' : 'Comprar con WebPay'}
</button>
```

Figura 3. Botón del frontend para comprar con WebPay

```
const handleWebpayBuy = async () => {
  if (!isAuthenticated) {
    loginWithRedirect();
    return;
  }

  const total = price * quantity;

  setLoading(true);
  try {
    const token = await getAccessTokenSilently();
    const result = await webpayStock(token, symbol, total, quantity);

    if (result.success) {
      setMessage(`Procesando compra: ${result.data.message || 'Procesando compra'}`);

      // Redireccionar al usuario a WebPay
      const form = document.createElement('form');
      form.method = 'POST';
      form.action = result.data.url;

      const input = document.createElement('input');
      input.type = 'hidden';
      input.name = 'token_ws';
      input.value = result.data.token_ws;

      form.appendChild(input);
      document.body.appendChild(form);
      form.submit();
    } else {
      setMessage(`Error: ${result.error}`);
    }
  } catch (error) {
    setMessage('Error en la conexión con el servidor');
    console.error(error);
  }

  setLoading(false);
};
```

Figura 4. Función para invocar la petición de creación de transacción a la API y redireccionamiento a página de WebPay

```
export const webpayStock = async (token, symbol, amount, quantity) => {
  try {
    // Set auth token in the header
    api.defaults.headers.common['Authorization'] = `Bearer ${token}`;

    // Create request body with the same format que usábamos antes
    const requestBody = {
      request_id: uuidv4(),
      timestamp: new Date().toISOString(),
      amount,
      quantity,
      symbol,
    };

    // Enviar la solicitud como se hacía en PurchaseMenu
    const response = await api.post(`/webpay/create`, requestBody, {
      headers: {
        'Content-Type': 'application/json',
      }
    });

    console.log("Webpay stock response:", response.data);
    return {
      success: true,
      data: response.data
    };
  } catch (error) {
    console.error('Error buying stock:', error);
    return {
      success: false,
      error: error.response?.data?.error || 'Error en la conexión con el servidor'
    };
  }
}
```

Figura 5. Función de `stocksService` que realiza petición `POST` a la API para la creación de la transacción

3.2. Página `/payment` del frontend para mostrar resultado:

Una vez finalizado el pago (ya sea exitoso o fallido), WebPay redirige al usuario a la URL previamente configurada al crear la transacción: `{URL_FRONTEND}/payment`, incluyendo el parámetro `token_ws`.

Este token es enviado al backend a través de la ruta `/webpay/commit`, donde se valida la transacción con Transbank. Finalmente, la respuesta de dicha validación se reenvía al frontend, donde se muestra un mensaje claro al usuario con el resultado final de la operación: pago aprobado, rechazado o cancelado.

Nota: Para esto se actualizaron las rutas en `App.jsx`

```jsx
import { useEffect, useState } from "react";
import { useSearchParams, Link } from "react-router-dom";
import { commitTransaction } from "../api/stocksService";
import { useAuth0 } from '@auth0/auth0-react';

const TransactionStatus = () => {
  const [searchParams] = useSearchParams();
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const { getAccessTokenSilently, isAuthenticated, loginWithRedirect } = useAuth0();

  const token_ws = searchParams.get("token_ws");

  useEffect(() => {
    const fetchTransactionStatus = async () => {
      try {
        const token = await getAccessTokenSilently();
        const result = await commitTransaction(token_ws, token);

        if (result.success) {
          setData({ message: result.data.message });
        } else {
          setData({ message: result.error });
        }
      } catch {
        setData({ message: "Error al confirmar transacción" });
      } finally {
        setLoading(false);
      }
    };

    fetchTransactionStatus();
  }, [token_ws]);

  if (loading) {
    return (
      <div className="p-20">
        <h1>Cargando...</h1>
      </div>
    );
  }

  return (
    <div className="p-8 mt-20 flex flex-col gap-3 w-1/3 mx-auto rounded-xl shadow-[0_0px_8px_#b4b4b4]">
      <h1 className="text-center font-bold text-lg">Resultado de la Compra</h1>
      <p className="text-center">{data?.message}</p>
      <Link to="/" className="bg-black text-white px-3 py-2 rounded text-center">Volver a inicio</Link>
    </div>
  );
};

export default TransactionStatus;
```

Figura 6. Página payment del frontend que recibe como `token_ws` como parámetro en la URL

```
export const commitTransaction = async (token_ws, token) => {
  const requestBody = {
    request_id: localStorage.getItem('request_id'),
    token_ws,
  };
  try {
    // Set auth token in the header
    api.defaults.headers.common['Authorization'] = `Bearer ${token}`;

    // Enviar la solicitud de confirmación de transacción
    const response = await api.post(`/webpay/commit`, requestBody, {
      headers: {
        'Content-Type': 'application/json',
      }
    });

    console.log("Commit transaction response:", response.data);
    return {
      success: true,
      data: response.data
    };
  } catch (error) {
    console.error('Error committing transaction:', error);
    return {
      success: false,
      error: error.response?.data?.error || 'Error en la conexión con el servidor'
    };
  }
};
```

Figura 7. Función de `stocksService` que realiza petición `POST` a la API para la confirmación de la transacción

```jsx
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Home from './views/home';
import Layout from './components/navbar/layout';
import Profile from './views/profile';
import Stocks from './views/stocks';
import UserStocks from './views/userStocks';
import EventHistory from './views/historial';
import TransactionStatus from './views/payment_status';

function App() {
  return (
    <Routes>
      <Route path="/" element={<Layout><Home /></Layout>} />
      <Route path="/profile" element={<Layout><Profile /></Layout>} />
      <Route path="/stocks" element={<Layout><Stocks /></Layout>} />
      <Route path="/userStocks" element={<Layout><UserStocks /></Layout>} />
      <Route path="/historial" element={<Layout><EventHistory /></Layout>} />
      <Route path="/payment" element={<Layout><TransactionStatus /></Layout>} />
    </Routes>
  );
}

export default App
```

Figura 8. Actualización de rutas del frontend