ChatGPT

Revisión de URLs de la API Simulador en uso

1. URLs válidas y actualmente en uso

Estas son las rutas que el proyecto Heroku utiliza para comunicarse con el Simulador, y que efectivamente existen en el Simulador:

- `/api/login/` – *Endpoint de autenticación.* La API Heroku lo usa para obtener un token JWT válido del Simulador [1]. En el Simulador está definido el endpoint `/api/login/` que valida credenciales de un **OficialBancario** y retorna un token [2].
- `/api/transferencia/` – *Inicio de transferencia (flujo OTP).* La API Heroku invoca este endpoint (con token Bearer) para registrar una nueva transferencia y obtener un desafío OTP [3]. El Simulador expone esta ruta (`api/transferencia/`) para recibir datos de una transferencia SEPA, crearla con estado "PDNG" y generar un código OTP [4] [5].
- `/api/transferencia/verify/` – *Verificación de OTP.* Usado por Heroku para confirmar una transferencia pendiente enviando el OTP proporcionado [6]. El Simulador define `/api/transferencia/verify/` para este paso: comprueba el OTP, marca la transferencia como completada y responde con el estado final [7] [8].
- `/api/challenge` – *Solicitud de OTP (flujo OAuth).* Heroku utiliza este endpoint para obtener un OTP de desafío en flujos protegidos [9]. En el Simulador, `/api/challenge` genera un OTP asociado a un **payment_id** dado [10] [11], aunque **requiere** que ese `payment_id` haya sido autorizado previamente vía OAuth (ver sección de conflictos).
- `/api/send-transfer` – *Confirmación con OTP+TOTP (flujo alternativo).* Es empleado en el código Heroku legado para enviar la transferencia con OTP y código TOTP [12]. Corresponde al endpoint `/api/send-transfer` del Simulador, que internamente crea una transferencia y un OTP (similar a `/api/transferencia/`) [13]. *(Nota:* Este endpoint se usaba en un flujo integrado de desafío+confirmación, ver duplicidades.)*
- `/otp/single` – *Generación de OTP en flujo "single step".* Configurado en Heroku como `OTP_URL`, este endpoint se invoca para obtener un OTP único asociado a una transferencia [14]. Existe en el Simulador (`/otp/single`), mapeado al mismo handler que inicia una transferencia y devuelve un OTP [15]. *(Ver sección de conflictos sobre su uso correcto.)*
- `/oidc/authorize` – *Autorización OAuth2 simulada.* Forma parte del flujo de consentimiento de usuario (Open Banking). La aplicación redirige al usuario a este endpoint (definido en el Simulador) para que apruebe una transferencia [16]. Tras el login simulado, el Simulador marca el `payment_id` como autorizado (`OAUTH_APPROVED`) [17].
- `/oidc/token` – *Token exchange OAuth2.* Endpoint del Simulador para intercambiar el código de autorización por tokens. La API Heroku lo llama internamente después del callback OAuth2, usando la URL definida en la configuración (`TOKEN_URL`) [18]. El Simulador expone `/oidc/token` (alias de `/api/token`) que entrega un JWT de acceso si el código y PKCE son válidos [16].

Justificación: Todas las rutas anteriores aparecen configuradas en las variables de entorno o código del proyecto Heroku y tienen su contraparte implementada en el Simulador. Por ejemplo, Heroku construye

URLs base del Simulador con IP `80.78.30.242:9181` para `/api/login/`, `/api/transferencia/`, etc., según `.env` [19], y estas rutas están definidas en `banco/urls.py` del Simulador [20] [13].

2. URLs innecesarias o no utilizadas actualmente

Son endpoints configurados o presentes en el código, pero que no están siendo llamados en los flujos activos de la API, ya sea por deprecación o porque pertenecen a funcionalidades no implementadas:

- `/api/status-transfer` – Está definida en la configuración (p. ej. `STATUS_URL` en *.env*) pero **no se utiliza** en el código actual (no se encontraron referencias a solicitudes HTTP de estado). El Simulador tiene `/api/status-transfer` implementado de forma básica (responde siempre status "RJCT") solo para completar la interfaz [21], por lo que parece un stub no requerido en los flujos reales.
- `/api/transferencias/entrantes/` (y su alias `/payments`) – Endpoints expuestos por el Simulador para un flujo integrado de transferencia+OTP [22] 【8†L25-L29**, pero la API Heroku** no los llama** actualmente. En vez de usar este único endpoint "entrante" para generar y confirmar OTP, el código opta por los endpoints separados (`/api/challenge` + confirmación, o `/api/transferencia/verify/`), dejando `/payments` sin uso. Su presencia en la configuración (`API_URL = .../payments`) [18] no se ve reflejada en llamadas efectivas del código.
- `/oidc/token` (alias `/api/token`) – Aunque se configuró para OAuth, la API Heroku terminó usando en su mayoría `/api/login/` para obtener tokens (login técnico). Solo en el flujo OAuth2 se aprovecharía `/oidc/token`, pero si ese flujo no está plenamente activo en producción, este endpoint externo queda prácticamente sin invocar. Del mismo modo, `/api/token` resulta redundante (ver duplicidades) y no se usa en llamadas servidor-servidor.
- `/frontend/transfer` – Endpoint del Simulador que devuelve una vista HTML de prueba [23]. La API Heroku no interactúa con este recurso (es para uso manual desde el navegador), por lo que no influye en las llamadas API.

En resumen, estas URLs aparecen definidas en el sistema, pero actualmente no aportan al flujo de integración **real** entre Heroku y Simulador, pudiendo eliminarse o ignorarse sin impacto. Su identificación permite simplificar la configuración y concentrarse solo en las rutas realmente utilizadas.

3. URLs duplicadas o redundantes

Se hallaron varios casos donde existen endpoints distintos que realizan funciones equivalentes, ya sea por mantener compatibilidad o por transición de una convención a otra. Esto genera duplicidad innecesaria:

- **Emisión de token:** El Simulador ofrece dos URLs para autenticar y obtener JWT: `/api/login/` y `/api/token` (esta última también accesible como `/oidc/token`) [16] [20]. Ambas devuelven un token válido tras verificar las credenciales, usando la misma lógica interna (función `generar_token`) [24] [16]. En la práctica, Heroku solo necesita usar una de ellas (se optó por `/api/login/`), por lo que mantener ambas rutas es redundante.
- **Inicio de transferencia (OTP):** Existen hasta *tres* endpoints para registrar una transferencia pendiente y generar OTP: `/api/transferencia/`, `/api/send-transfer` y `/otp/single`. Todos invocan el mismo procedimiento en el Simulador (`api_send_transfer`) que crea la transferencia y produce un código OTP [13] [15]. Originalmente se usaba la ruta en inglés `/api/`

`send-transfer`, luego se añadió la versión localizada `/api/transferencia/` y un alias `/otp/single` (quizá para cierto cliente). Esto duplica funcionalidad; la API Heroku debería unificar sus llamadas en uno solo de estos endpoints.

- **Generación de desafío OTP:** De modo similar, `/api/challenge` y `/auth/challenges` son dos caminos al **mismo** servicio de generación de OTP en el Simulador (ambos mapean a `api_challenge`) [25]. La diferencia está en la semántica de uso (uno bajo prefijo `/api` y otro bajo `/auth` dentro de OAuth), pero internamente no varía: ambos crean un `OTPChallenge` y devuelven el código OTP [10]. Esto representa un solapamiento; podría estandarizarse uno solo.
- **Confirmación de transferencia:** Hay redundancia entre un **flujo en dos pasos** vs **un paso combinado**. El flujo **dos pasos** utiliza `/api/transferencia/` (crea transferencia + OTP) seguido de `/api/transferencia/verify/` (confirma con OTP). En paralelo, el Simulador expone un flujo **combinado** vía `/payments` (o `/api/transferencias/entrantes/`): en una primera llamada sin OTP devuelve un OTP, y en la segunda llamada con OTP+TOTP finaliza la transacción [26] [27]. La API Heroku implementó principalmente el flujo de dos pasos, haciendo que el endpoint combinado sea redundante. Mantener ambos enfoques puede causar confusión; se recomienda escoger un solo esquema de confirmación.
- **Definiciones duplicadas en configuración:** En la configuración de Heroku hay variables repetidas apuntando a lo mismo, ejemplo: `SIMULADOR_API_URL` y `API_URL` (ambas con la ruta base de transferencias), o `SIMULADOR_LOGIN_URL` y `TOKEN_ENDPOINT` [19] [28]. Esta duplicación refleja posiblemente etapas de migración de nombres; conviene depurar para evitar llamadas dobles o referencias inconsistentes a un mismo endpoint.

Justificación técnica: Las duplicidades anteriores no aportan nuevas funcionalidades, solo replican caminos diferentes al mismo recurso del Simulador. Esto eleva la complejidad: por ejemplo, tres URLs para crear transferencias significan más puntos a mantener y probar, aumentando el riesgo de errores. Consolidar cada caso duplicado en una única URL clara hará las llamadas más **sencillas de entender** y reducirá posibles inconsistencias.

4. URLs problemáticas que pueden generar conflictos

Por último, se identificaron algunas rutas cuyo uso (o mal uso) puede provocar errores o comportamientos inesperados, debido a inconsistencias entre el cliente Heroku y el servidor Simulador:

- **Uso de `/auth/login` en lugar de `/api/login`:** El código Heroku define una función que llama a `SIMU_BASE/auth/login` [29], pero **no existe** ningún endpoint `/auth/login` en el Simulador (las rutas bajo `/auth/` son solo `/auth/challenges`) [15]. Esta discrepancia provoca un **404** – es un bug en el código (debe usar `/api/login/`).
- **Endpoints inexistentes `/api/transfers/initiate` y `/api/transfers/confirm`:** Algunas funciones (ej. `iniciar_transferencia` y `confirmar_transferencia`) intentan usar rutas en plural (`/api/transfers/...`) [30] [31] que **no están definidas** en el Simulador. Ningún patrón en `urls.py` del Simulador coincide con "`/api/transfers`" [13], resultando en errores 404 si se invocaran. Estas llamadas parecen pertenecer a un desarrollo anterior o pruebas, y deben eliminarse o corregirse para apuntar a los endpoints reales.
- **Confirmación de OTP en endpoint incorrecto:** Se detectó que en cierto flujo Heroku envía el OTP al mismo endpoint de inicio (`/api/transferencia/`) en lugar del específico de verificación [3]. Dado que el Simulador espera el OTP en `/api/transferencia/verify/` y trata cualquier POST a `/api/transferencia/` como **nuevo** registro [4], este uso indebido puede resultar en

transferencias duplicadas o en ignorar el OTP enviado. Es fundamental que la confirmación de código se dirija al endpoint correcto para evitar inconsistencias lógicas.

- **Llamar a `/otp/single` sin datos completos:** La función `solicitar_otp` de Heroku invoca `/otp/single` pasando solo un `payment_id` [14], asumiendo que obtendrá un OTP. Sin embargo, `/otp/single` **en Simulador equivale a** `/api/transferencia/`, que intenta crear una transferencia con los datos suministrados [32]. Si faltan campos (IBAN, monto, etc.), podría lanzarse un error de validación o incluso crear un registro incompleto. En realidad, para solo generar un OTP (por ejemplo, segundo factor), debería usarse `/api/challenge` o el flujo `/payments`. Esta malinterpretación del endpoint puede generar fallos y OTPs no vinculados correctamente a una transferencia.
- **Llamada a `/api/challenge` sin consentimiento previo:** El endpoint `/api/challenge` del Simulador exige que el **payment_id** tenga autorización previa (simula que el usuario aprobó la operación vía OAuth) [17]. Si Heroku lo invoca directamente tras hacer login técnico (como ocurre en `real_transfer` sin pasar por `/oidc/authorize`) [9], el Simulador responderá con error **403 "OAuth no aprobado"**. Esto indica un posible flujo incoherente: la API está solicitando OTP sin haber obtenido consentimiento del usuario en el Simulador. La solución sería o bien usar `/auth/challenges` tras un proceso OAuth adecuado, o omitir el paso OTP si se asume un oficial autenticado. No alinear este flujo causará fallos de autorización.

Conclusión técnica: Los puntos anteriores representan desajustes entre cliente y servidor. Para optimizar la integración, el proyecto debe **corregir o eliminar** estas llamadas conflictivas. Esto implica: reemplazar URLs mal formadas (p. ej. `/auth/login` → `/api/login`), remover referencias a endpoints inexistentes, y asegurar que cada petición siga la secuencia lógica que el Simulador espera (por ejemplo, no pedir OTP sin autorización previa, y usar el endpoint correcto para confirmarla). Atender estos conflictos evitará errores 404, respuestas de error desde el Simulador y comportamientos indeterminados, logrando una comunicación **consistente y fiable** con la API del Simulador [29] [15] [30] [13].

---

[1] GitHub
https://github.com/markmur90/heroku/blob/666498531d12dc4fc572c4158dc0ef74dc50c962/api/core/auth_services.py

[2] [4] [5] [7] [8] [10] [11] [17] [21] [24] [26] [27] [32] GitHub
https://github.com/markmur90/Simulador/blob/16a98ba7c46f6172bf9a67becc3f7e7a3acda02b/simulador_banco/banco/views.py

[3] GitHub
https://github.com/markmur90/heroku/blob/666498531d12dc4fc572c4158dc0ef74dc50c962/api/gpt4/services/transfer_services.py

[6] [14] [29] [30] [31] GitHub
https://github.com/markmur90/heroku/blob/666498531d12dc4fc572c4158dc0ef74dc50c962/api/gpt4/conexion/conexion_banco.py

[9] [12] GitHub
https://github.com/markmur90/heroku/blob/666498531d12dc4fc572c4158dc0ef74dc50c962/api/core/bank_services.py

[13] [15] [16] [20] [22] [23] [25] GitHub
https://github.com/markmur90/Simulador/blob/16a98ba7c46f6172bf9a67becc3f7e7a3acda02b/simulador_banco/banco/urls.py

[18] [19] [28] GitHub
https://github.com/markmur90/heroku/blob/666498531d12dc4fc572c4158dc0ef74dc50c962/.env.local