ChatGPT

Refactorización de la Integración con la API del Simulador

Rutas Eliminadas (Depuración de Endpoints Innecesarios)

Se identificaron varios endpoints del Simulador que el proyecto Heroku no utiliza en la integración actual. Estas rutas se eliminan para simplificar la configuración y evitar conflictos:

- `/payments` – Alias de la ruta de transferencias entrantes (`/api/transferencias/entrantes/`) que no se está usando en la integración actual. Dado que el flujo implementado utiliza llamadas separadas para OTP y envío, el endpoint unificado `/payments` resulta redundante [1] [2]. Se elimina para evitar confusión entre dos formas de hacer lo mismo.
- `/api/status-transfer` – Endpoint pensado para consultar el estado de una transferencia por ID. No hay llamadas en el código actual que usen esta ruta, por lo que se elimina. Si en el futuro se requiere verificar el estado de una transferencia, se puede incorporar nuevamente o manejarse mediante la respuesta directa del envío o consultas a la base de datos.
- **Otros endpoints no utilizados:** igualmente se depuran rutas alias que no aportan al flujo actual. Por ejemplo, la ruta `/oidc/token` es equivalente a `/api/token` [3]; si solo se usa una de ellas, la otra puede omitirse en la configuración del cliente. En general, cualquier endpoint definido en el Simulador que no tenga correspondencia en el consumo desde Heroku ha sido eliminado de la configuración para garantizar una integración consistente y mínima.

Corrección y Consolidación de Rutas (Antes → Después)

Algunas rutas empleadas en el código estaban mal escritas o duplicadas, apuntando a endpoints incorrectos o múltiples endpoints para la misma acción. A continuación se presenta una tabla con las correcciones aplicadas a dichas rutas, unificando criterios:

Ruta (antes)	Ruta (después)
`/auth/login`	`/api/login`
`/auth/challenges`	`/api/challenge`
`/otp/single`	`/api/transferencia/`
`/api/send-transfer`	`/api/transferencia/`

Justificación de cambios:
- `**/auth/login → /api/login**`: La ruta anterior no existe en el Simulador; el endpoint correcto para autenticación es `/api/login` [4]. Se corrige tanto la URL base consumida como cualquier variable de entorno asociada para apuntar al path válido.
- `**/auth/challenges → /api/challenge**`: Se unifica el uso del prefijo `/api` para el reto OTP. El

Simulador expone el desafío OTP en `/api/challenge` (también accesible en `/auth/challenges` como alias) [1]. Para mantener consistencia, el proyecto ahora utiliza únicamente `/api/challenge`.
- `**/otp/single & /api/send-transfer → /api/transferencia/**`: Estos tres endpoints efectuaban la misma operación de envío de transferencia con OTP [1] [5]. Se consolidan en una única ruta estándar `/api/transferencia/` (en español, para concordar con otras rutas del Simulador) eliminando los duplicados. Esto garantiza que todas las solicitudes de transferencias salientes usen el mismo endpoint y evita rutas paralelas que podrían des sincronizarse.

> **Nota:** La ruta unificada `/api/transferencia/` espera recibir el OTP en la petición (y opcionalmente el código TOTP si aplica). Previamente, si no se proporcionaba OTP, el Simulador devolvía un desafío OTP. Ahora el flujo está claramente separado: primero se obtiene el OTP vía `/api/challenge` y luego se envía la transferencia con OTP vía `/api/transferencia/`. La eliminación de los duplicados `/otp/single` y `/api/send-transfer` fija este comportamiento de forma consistente.

Código Refactorizado de las Llamadas a la API

Se aplicaron cambios en las funciones Python que interactúan con la API del Simulador, principalmente en módulos como `api.py`, `views.py` y `utils.py`. A continuación se muestra el código actualizado para cada llamada API afectada, con la corrección de URLs y la consolidación de rutas:

1. **Autenticación (Obtención de Token JWT)** – Se corrige la URL de login. Antes se usaba una ruta incorrecta `/auth/login`; ahora se utiliza `/api/login`. Por ejemplo:

```
# Antes: se utilizaba una ruta inexistente '/auth/login'
token_response = requests.post(f"{BASE_URL}/auth/login", json={"usuario": user,
"clave": pwd})
```

```
# Después: se utiliza el endpoint correcto del Simulador '/api/login'
token_response = requests.post(f"{BASE_URL}/api/login", json={"usuario": user,
"clave": pwd})
```

Cambio aplicado: En la configuración, `BASE_URL` sigue apuntando al host del Simulador, pero el path de login se corrige a `/api/login`. Cualquier variable de entorno previa (p. ej. `AUTH_LOGIN_PATH`) se reemplaza por `API_LOGIN_PATH` o similar, acorde al nuevo endpoint.

1. **Solicitud de Desafío OTP** – Unificación de la ruta de challenge. Si antes el código llamaba al endpoint alias con distinto prefijo, ahora se asegura de usar `/api/challenge`. Por ejemplo:

```
# Antes: llamado al endpoint de desafío OTP con prefijo incorrecto
challenge_url = os.getenv("AUTH_URL")  # e.g. '.../auth/challenges'
```

```
res = requests.post(challenge_url, headers={"Authorization": f"Bearer {token}"},
json={"payment_id": pid})
```

```
# Después: llamado al endpoint unificado de desafío OTP
challenge_url = os.getenv("CHALLENGE_URL")  # e.g. '.../api/challenge'
res = requests.post(challenge_url, headers={"Authorization": f"Bearer {token}"},
json={"payment_id": pid})
```

Cambio aplicado: Se elimina el uso de `AUTH_URL` (que apuntaba a `/auth/challenges`) de las variables de entorno y del código. En su lugar, se introduce `CHALLENGE_URL` apuntando a la ruta correcta. La lógica de manejo de la respuesta permanece igual – si el Simulador responde con un OTP (código de seis dígitos) y un `challenge_id`, el cliente lo almacenará para el siguiente paso.

1. **Envío de Transferencia (Con OTP)** – Consolidación de la ruta de envío. En el flujo refactorizado, una vez obtenido el OTP del desafío, se envía la transferencia usando **solo** el endpoint unificado `/api/transferencia/`. Ejemplo de cambio:

```
# Antes: se podía llamar a '/otp/single' o '/api/send-transfer' según
configuración
transfer_url = os.getenv("OTP_URL")  # por ejemplo '.../otp/single'
payload = {"payment_id": pid, "otp": otp_code}
res = requests.post(transfer_url, headers={"Authorization": f"Bearer {token}"},
json=payload)
```

```
# Después: se utiliza únicamente la ruta consolidada '/api/transferencia/'
transfer_url = os.getenv("TRANSFER_URL")  # e.g. '.../api/transferencia/'
payload = {"payment_id": pid, "otp": otp_code}
res = requests.post(transfer_url, headers={"Authorization": f"Bearer {token}"},
json=payload)
```

Cambios aplicados:
- Se eliminan las referencias a `OTP_URL` o `SEND_TRANSFER_URL` antiguas, asegurando que la variable de entorno (ahora `TRANSFER_URL` o `SIMULADOR_API_URL`) apunte a `/api/transferencia/`.
- El código que manejaba múltiples variantes de endpoint según diferentes condiciones se simplifica a una única vía. Por ejemplo, si había lógica para decidir entre `/api/send-transfer` vs `/otp/single`, esa bifurcación desaparece; ahora toda transferencia saliente usa la misma URL.
- **TOTP:** Si el Simulador requiere además un código TOTP (One-Time Password de tiempo) en el payload, el código lo seguirá incluyendo (ej. `payload["totp"] = totp_code`) como antes. La ruta unificada soporta ambos valores (`otp` y `totp`) en la misma llamada [2], por lo que no se requiere un endpoint aparte para verificar códigos temporales.

1. **Consulta de Estado de Transferencia (eliminada)** – En versiones previas, se contemplaba una función para consultar el estado de una transferencia vía `/api/status-transfer`. Dado que este

endpoint ha sido eliminado por no usarse, cualquier función similar a `get_transfer_status()` ha sido suprimida o marcada como obsoleta. Por ejemplo, se removió el bloque de código:

```
# Antes: consultar estado de la transferencia (ya no utilizado)
status_url = os.getenv("STATUS_URL")  # e.g. '.../api/status-transfer'
res = requests.get(f"{status_url}?payment_id={pid}", headers={"Authorization":
f"Bearer {token}"})
status = res.json().get("status")
```

Ahora, la aplicación asume que la respuesta de la transferencia (u otras notificaciones del sistema) proveerán el estado necesario. Si fuera necesario monitorear el estado asíncronamente, se reintroduciría esta funcionalidad con la ruta correcta y las adecuaciones pertinentes, pero en la refactorización actual no se utiliza.

Resumen de cambios de código: Todas las llamadas `requests.get/post` hacia el Simulador han sido actualizadas para reflejar las rutas vigentes. Además de los ejemplos anteriores, se revisaron módulos auxiliares (`api.py`, `utils.py`) para asegurarse de que ninguna cadena de URL quede apuntando a rutas obsoletas. Las funciones están documentadas adecuadamente para indicar que usan la API del Simulador (por ejemplo, comentando encima de `send_transfer()` que utiliza `/api/transferencia/` con OTP, etc.), facilitando el mantenimiento futuro.

Mejoras en la Configuración y Variables de Entorno

La refactorización incluye una limpieza de las variables de entorno relacionadas con la integración, consolidando claves duplicadas y eliminando entradas innecesarias:

- **Unificación de variables para endpoints:** Se detectó que existían múltiples variables apuntando a un mismo propósito. Por ejemplo, tanto `OTP_URL` (usada antes para `/otp/single`) como `TRANSFER_URL` apuntaban a la acción de enviar transferencias [6] [7]. Del mismo modo `AUTH_URL` vs `CHALLENGE_URL` para el desafío OTP, y varias referentes al login (`SIMULADOR_LOGIN_URL`, `TOKEN_ENDPOINT`, `LOGIN_URL`) redundaban entre sí. Esto se ha simplificado quedando **una sola variable por endpoint**. En particular:
- Se conserva `CHALLENGE_URL` para la ruta de reto OTP (`/api/challenge`), eliminando `AUTH_URL`.
- Se conserva `TRANSFER_URL` (o renombrada a `SIMULADOR_API_URL`) para la ruta de envío `/api/transferencia/`, eliminando `OTP_URL` y `SEND_PATH` anteriores.
- Para el login, se opta por una única variable (`SIMULADOR_LOGIN_URL` apuntando a `/api/login`) eliminando duplicados como `TOKEN_URL` o mal nombrados.

- La variable `STATUS_URL` se elimina, dado que ya no usamos `/api/status-transfer`.

- **Uso consistente de BASE_URL vs URLs completas:** Previamente, la configuración mezclaba un `BASE_URL` con subrutas por separado *y* variables que ya contenían la URL completa [8] [9]. Esto podía llevar a errores si no se actualizaban todas en conjunto. Tras la refactorización, se adopta una convención consistente. Dos enfoques posibles y recomendados:

- **Usar BASE_URL + PATHs:** Definir `BASE_URL` del Simulador (por ej., la URL del VPS o dominio) y variables para cada path (`LOGIN_PATH`, `CHALLENGE_PATH`, `TRANSFER_PATH`). El código construye las URLs cuando las necesita (`requests.post(BASE_URL + TRANSFER_PATH, ...)`). Esto evita repetir el host en múltiples variables y facilita cambios de dominio en un solo lugar.
- **Usar URLs completas para cada endpoint:** Alternativamente, definir directamente `SIMULADOR_LOGIN_URL`, `SIMULADOR_CHALLENGE_URL`, `SIMULADOR_TRANSFER_URL`, etc., con el host y path ya concatenados. Esto simplifica el uso (cada variable es invocable directamente), pero requiere actualizar todas si cambia el host.

En ambos casos, **se eliminan las definiciones duplicadas**. La configuración resultante es más clara: cada endpoint del Simulador tiene exactamente una variable de entorno asociada, y cada variable se usa con un único propósito claro. Por ejemplo, ya no habrá dos variables distintas para la misma ruta `/api/transferencia/`.

- **Limpieza de credenciales duplicadas:** Además de las URLs, se detectó que se tenían múltiples conjuntos de credenciales de usuario para el Simulador (p. ej., `SIM_USER` / `SIM_PASS` y `SIMULADOR_USER` / `SIMULADOR_PASS`) [10] [11]. Se consolidaron estos valores, manteniendo solo las claves necesarias (por ejemplo, `SIMULADOR_USERNAME` y `SIMULADOR_PASSWORD`) y eliminando entradas no utilizadas. Esto previene incoherencias donde un cambio de usuario/clave podría aplicarse en una variable pero otra obsoleta quedar configurada.

En resumen, la organización de la configuración ahora sigue el principio de "una sola fuente de la verdad" por cada parámetro. Esto hace más fiable la implementación en Heroku, ya que al desplegar la aplicación solo necesita las variables esenciales correctamente asignadas, reduciendo el riesgo de error humano al manejar muchas claves similares.

Recomendaciones Generales para Futuras Integraciones

La refactorización realizada prepara el terreno para una integración más robusta y mantenible con la API del Simulador. A continuación se ofrecen recomendaciones para garantizar buenas prácticas a futuro:

- **Mantener consistencia con la API del Simulador:** A medida que evolucione el Simulador, es importante alinear rápidamente la integración en el cliente Heroku. Si se añaden nuevos endpoints o se deprecian algunos, siga el mismo enfoque de mantener solo un método para cada operación. Evite agregar rutas alias o duplicadas en la configuración del cliente; en lugar de ello, actualice las existentes.
- **Documentación y comunicación:** Documente en el repositorio del proyecto las rutas válidas de la API Simulador que se están usando, indicando qué hace cada una. Esto ayuda a nuevos desarrolladores a entender el flujo (por ejemplo, "`/api/challenge` – solicita código OTP para la transferencia en curso"). Asimismo, si el Simulador es mantenido por un tercero o equipo separado, coordine con ellos para obtener documentación actualizada de sus endpoints. Detectar discrepancias (como el caso de un `/auth/login` inexistente) es más fácil con una referencia oficial.
- **Simplificar flujo de integración:** La decisión de separar el proceso en dos pasos (challenge OTP y envío) ha demostrado ser más clara y fácil de depurar que la alternativa de un solo paso integrado [2]. Se recomienda continuar con esta filosofía de claridad. Por ejemplo, si se implementa en el

futuro una confirmación adicional (como autorización del usuario vía OAuth), manejarla en pasos secuenciales bien definidos. Esto facilita pruebas y manejo de errores en cada etapa.

- **Manejo de errores y validaciones:** Tras los cambios, asegúrese de probar casos de error: ¿Qué sucede si el Simulador devuelve un error 4XX/5XX en alguno de los nuevos endpoints? Conviene que el código cliente maneje gracefully esas situaciones (reintentos, mostrar mensajes claros, etc.). Dado que se removieron rutas no usadas, cualquier intento de usarlas debería alertar rápidamente (por ejemplo, si accidentalmente alguna parte del código intenta invocar `/payments`, fallará y así se detecta un fragmento olvidado).
- **Variables de entorno en producción:** Al desplegar en Heroku, verifique que solo estén configuradas las nuevas variables unificadas. Eliminar del entorno de Heroku aquellas claves antiguas garantiza que nadie pueda usarlas por error. Además, agrupe lógicamente las variables (por convención de nombres) para identificar que pertenecen al Simulador. Por ejemplo, prefijar todas con `SIMULADOR_` (o un prefijo común) ayuda a distinguirlas de otras configuraciones de la app.
- **Revisión periódica y pruebas integrales:** Finalmente, antes de mover cambios a producción, realizar pruebas completas del flujo (login → challenge OTP → envío transferencia) en un entorno de staging. Dado que las rutas cambiaron, asegúrese de que no queden referencias colgantes. Una búsqueda de texto en el código por cualquier endpoint antiguo (`/otp/single`, `/auth/`) puede confirmar que fueron totalmente eliminados. También monitoree en el Simulador que las peticiones llegan correctamente a los nuevos endpoints esperados.

Con estas modificaciones, el proyecto Heroku queda con una integración más limpia y coherente con la API del Simulador. La eliminación de rutas superfluas y la consolidación de URLs reducen la superficie de error y facilitan el mantenimiento a largo plazo, permitiendo llevar los cambios a producción de forma confiable.

Fuentes: La refactorización se basó en el análisis del código del Simulador [1] [5] y la configuración existente [6] [9], lo que permitió identificar las discrepancias y alinearlas con las rutas soportadas. Asimismo, se tomó en cuenta la documentación interna del flujo OTP del Simulador [2] para asegurar que el comportamiento esperado se mantuviera consistente tras los cambios.

---

[1] [3] [4] [5] urls.py
https://github.com/markmur90/Simulador/blob/16a98ba7c46f6172bf9a67becc3f7e7a3acda02b/simulador_banco/banco/urls.py

[2] detalles_simulador_03.txt
https://github.com/markmur90/api_bank_h2/blob/719dbf9589cf846efe5bf82ba846fa0ab0d048b8/detalles_simulador_03.txt

[6] [7] [8] [9] [10] [11] .env.local
https://github.com/markmur90/api_bank_h2/blob/719dbf9589cf846efe5bf82ba846fa0ab0d048b8/.env.local