**✅ Paso 1: Cambiar el main y el start**

Como querés que Render ejecute el webhook, cambiamos esto:

json

"main": "index.js",

"scripts": {

"start": "node index.js"

}

Por esto:

json

"main": "webhookListener.js",

"scripts": {

"start": "node webhookListener.js"

}

**✅ Paso 2: Confirmar estructura de archivos**

Tu carpeta debe tener:

* webhookListener.js → el servidor Express que escucha el code
* index240925.js → el script que intercambia el token
* .env.example → con las claves necesarias (Render te pedirá que las cargues manualmente)
* package.json → actualizado como arriba

**✅ Paso 3: Subir a GitHub**

1. Creá un repositorio en GitHub (ej: mercadolibre-oauth-hook)
2. Subí tu carpeta con los archivos mencionados
3. Confirmá que el start funciona localmente:

bash

npm install

npm start

**✅ Paso 4: Desplegar en Render**

1. Ingresá a https://render.com
2. Hacé clic en **"New Web Service"**
3. Elegí **"Deploy from GitHub"**
4. Configurá:
   * **Name**: mercadolibre-hook
   * **Environment**: Node
   * **Build Command**: npm install
   * **Start Command**: npm start
   * **Environment Variables**:
     + CLIENT\_ID
     + CLIENT\_SECRET
     + REDIRECT\_URI → https://mercadolibre-hook.onrender.com/hook
5. Render te dará una URL como:

Código

https://mercadolibre-hook.onrender.com/hook

**✅ Paso 5: Actualizar en Mercado Libre**

En tu app de desarrollador:

* Cambiá el redirect\_uri a la URL de Render
* Guardá los cambios

¿Querés que te prepare el contenido completo de webhookListener.js para que lo copies y subas sin tocar nada más? También puedo ayudarte a crear el .env.example para que Render te lo pida automáticamente.

webhookListener.js no lo tengo

Perfecto, Justo. Te lo dejo listo para que lo copies directamente en un archivo llamado webhookListener.js. Este script escucha el code que Mercado Libre envía al redirect\_uri y ejecuta automáticamente tu script index240925.js para intercambiar el token.

**🧩 Código completo: webhookListener.js**

js

// webhookListener.js

import express from 'express';

import { exec } from 'child\_process';

const app = express();

const PORT = process.env.PORT || 3000;

app.get('/hook', (req, res) => {

const code = req.query.code;

if (!code) {

console.error('❌ No se recibió el code en el webhook');

return res.status(400).send('Falta el parámetro code');

}

console.log('📥 Code recibido:', code);

// Ejecuta el script principal con el code recibido

exec(`node index240925.js ${code}`, (err, stdout, stderr) => {

if (err) {

console.error('❌ Error al ejecutar index240925.js:', err.message);

return res.status(500).send('Error interno al procesar el token');

}

console.log('📦 Resultado del intercambio:\n', stdout);

res.send('✅ Code procesado y token solicitado');

});

});

app.listen(PORT, () => {

console.log(`🚀 Webhook escuchando en http://localhost:${PORT}/hook`);

});

**✅ ¿Qué hacés ahora?**

1. Guardás este código en un archivo llamado webhookListener.js.
2. Actualizás tu package.json para que Render sepa qué ejecutar:

json

"main": "webhookListener.js",

"scripts": {

"start": "node webhookListener.js"

}

1. Subís tu proyecto a GitHub.
2. Lo desplegás en Render como te expliqué antes.
3. En tu app de Mercado Libre, ponés como redirect\_uri:

Código

https://TU\_DOMINIO\_RENDER.onrender.com/hook

**🧩 Paso a paso para subir solo lo necesario a GitHub**

**✅ 1. Crear una carpeta limpia**

En tu máquina local, creá una carpeta nueva (por ejemplo mercadolibre-hook) y copiá dentro solo estos archivos:

* webhookListener.js
* index240925.js
* package.json (ajustado con "start": "node webhookListener.js")
* .env.example (sin claves reales, solo los nombres)
* verifier.json (opcional, si querés mostrar estructura)
* token.json (opcional, si querés mostrar cómo se guarda)

**✅ 2. Inicializar el repositorio**

Abrí una terminal en esa carpeta y ejecutá:

bash

git init

git add .

git commit -m "Webhook listo para Render"

**✅ 3. Crear el repositorio en GitHub**

1. Ingresá a https://github.com
2. Hacé clic en **New repository**
3. Poné un nombre como mercadolibre-oauth-hook
4. Elegí **"Private"** si no querés que sea público
5. No marques ninguna opción extra (ni README ni .gitignore)

**✅ 4. Conectar tu carpeta local con GitHub**

GitHub te va a mostrar un bloque como este:

bash

git remote add origin https://github.com/tuusuario/mercadolibre-oauth-hook.git

git branch -M develop

git push -u origin develop

Copiás y pegás eso en tu terminal. Así subís tu carpeta local a la rama develop.

**✅ 5. Verificás en GitHub**

Entrás al repositorio y confirmás que solo están los archivos que querías subir.

**🧠 Tip extra: Ignorar archivos sensibles**

Si querés evitar subir archivos como .env o token.json, creá un archivo .gitignore con esto:

Código

.env

token.json

verifier.json

node\_modules/

Y luego:

bash

git add .gitignore

git commit -m "Ignorar archivos sensibles"

git push

¡Hola Justo! Qué gusto saludarte 😊  
¿En qué estás trabajando hoy? ¿Seguimos con el despliegue en Render, el dashboard local o te gustaría revisar algo nuevo? También podemos hacer una pausa técnica y charlar de curiosidades si querés.

¡Perfecto que estés en esa etapa, Justo! Vamos a aclararlo bien:

🔐 **CLIENT\_ID y CLIENT\_SECRET**   
Cuando estás integrando OAuth2 con Mercado Libre, **los valores que tenés que usar son los que te da Mercado Libre** al registrar tu aplicación en su [Developer Portal](https://developers.mercadolibre.com.ar/es_ar/autenticacion-y-autorizacion).  
Estos identificadores son únicos para tu app y son los que Mercado Libre usa para validar tu identidad como desarrollador.

🌐 **¿Y Render?**   
Render no genera client\_id ni client\_secret para OAuth2. Lo que sí hace es alojar tu aplicación (por ejemplo, tu webhookListener.js) y permitirte configurar variables de entorno.  
Ahí es donde **vas a guardar de forma segura** los valores que te dio Mercado Libre, como:

CLIENT\_ID=tu\_client\_id\_de\_mercado\_libre

CLIENT\_SECRET=tu\_client\_secret\_de\_mercado\_libre

📦 **¿Dónde los usás?**   
En tu script de intercambio de token (tokenExchange.js, por ejemplo), vas a leer esos valores desde process.env.CLIENT\_ID y process.env.CLIENT\_SECRET, para que no estén hardcodeados.

Si querés, te puedo ayudar a validar que tu redirect\_uri esté bien configurado o revisar cómo estás leyendo las variables en tu código. ¿Querés que lo veamos paso a paso?

¡Excelente, Justo! La captura que compartiste confirma que estás configurando correctamente las variables de entorno en Render 🎯

🔑 **CLIENT\_ID y CLIENT\_SECRET** → **Usás los que te dio Mercado Libre**, no Render.  
Render simplemente te da el espacio para alojar tu app y guardar esos valores de forma segura. Lo que hiciste está perfecto: los definiste como variables de entorno para que tu código pueda acceder a ellos sin exponerlos.

📁 En tu código, deberías leerlos así:

const clientId = process.env.CLIENT\_ID;

const clientSecret = process.env.CLIENT\_SECRET;

🧩 Esto te permite mantener tu app modular y segura. Si querés, podemos revisar juntos cómo estás armando el redirect\_uri o cómo estás leyendo los tokens en el script de intercambio. También puedo ayudarte a validar que el flujo esté completo y autónomo.

¿Querés que armemos un checklist rápido para confirmar que todo esté listo para el despliegue?

¡Vamos a destrabar eso, Justo! Si Render te está pidiendo algo más además del CLIENT\_ID y CLIENT\_SECRET, lo más probable es que se refiera a alguno de estos puntos clave del flujo OAuth2:

🔁 **REDIRECT\_URI**   
Este es fundamental. Tiene que coincidir exactamente con el que registraste en Mercado Libre.  
Ejemplo típico si estás usando Render:

https://tu-app-en-render.onrender.com/callback

📦 **Otros posibles campos que podrías necesitar configurar:**

| **Variable** | **¿Para qué sirve?** |
| --- | --- |
| REDIRECT\_URI | Donde Mercado Libre redirige después del login. Debe estar registrado en su portal. |
| PORT | Puerto en el que corre tu servidor (usualmente 3000 o el que Render asigna). |
| BASE\_URL | URL pública de tu app en Render, útil para construir rutas dinámicas. |
| ACCESS\_TOKEN\_PATH | Ruta donde guardás o usás el token, si lo estás modularizando. |

📌 **Validación rápida:**

* ¿Tu redirect\_uri está registrado en Mercado Libre y coincide con el que usás en tu código?
* ¿Tu script de intercambio (tokenExchange.js) está leyendo correctamente las variables?
* ¿Tenés configurado el webhookListener.js para recibir el código y disparar el intercambio automáticamente?

Si querés, te ayudo a revisar el flujo completo paso a paso o te armo un esquema modular para validar que todo esté conectado. ¿Querés que lo hagamos?

¡Perfecto, Justo! Ya estás en la recta final para desplegar tu servicio en Render 🚀  
La sección que mostraste con el botón **“Deploy Web Service”** indica que ya configuraste las variables necesarias (CLIENT\_ID, CLIENT\_SECRET) y estás listo para lanzar tu app.

Antes de hacer clic en ese botón, te recomiendo revisar estos puntos clave:

✅ **Checklist previa al despliegue:**

1. **Variables de entorno**
   * CLIENT\_ID y CLIENT\_SECRET → ✅ Ya los tenés.
   * REDIRECT\_URI → ¿Está registrado en Mercado Libre y coincide con el que usás en tu código?
2. **Script de escucha (webhookListener.js)**
   * ¿Está preparado para recibir el code de Mercado Libre y disparar el script de intercambio automáticamente?
3. **Script de intercambio (tokenExchange.js)**
   * ¿Lee correctamente las variables de entorno?
   * ¿Guarda el access\_token y refresh\_token de forma segura?
4. **Logs y monitoreo**
   * ¿Tenés algún log visual o consola para validar que el flujo se ejecuta correctamente?
5. **Dominio en Render**
   * ¿Tu URL pública está configurada como redirect\_uri en Mercado Libre?

Si querés, puedo ayudarte a validar el redirect\_uri, revisar cómo disparar el script de intercambio desde el webhook, o incluso armar un esquema modular para que el flujo quede 100% autónomo.

¿Querés que revisemos el webhookListener.js antes de hacer el deploy?

¡Vamos a revisar el webhookListener.js, Justo! Te propongo una estructura modular y clara para que el webhook reciba el code de Mercado Libre y dispare automáticamente el script de intercambio de token. Así eliminamos pasos manuales y dejamos el flujo autónomo 💡

**🧩 Estructura sugerida para webhookListener.js**

import express from 'express';

import { exec } from 'child\_process';

import path from 'path';

const app = express();

const PORT = process.env.PORT || 3000;

// Ruta que Mercado Libre usa como redirect\_uri

app.get('/callback', (req, res) => {

const authCode = req.query.code;

if (!authCode) {

return res.status(400).send('No se recibió el código de autorización.');

}

console.log('✅ Código recibido:', authCode);

// Ejecutar el script de intercambio de token

const scriptPath = path.resolve('./tokenExchange.js');

const command = `node ${scriptPath} ${authCode}`;

exec(command, (error, stdout, stderr) => {

if (error) {

console.error('❌ Error al ejecutar tokenExchange:', error);

return res.status(500).send('Error al intercambiar el token.');

}

console.log('📦 STDOUT:', stdout);

console.error('⚠️ STDERR:', stderr);

res.send('Token intercambiado correctamente. Podés cerrar esta ventana.');

});

});

app.listen(PORT, () => {

console.log(`🚀 Webhook escuchando en http://localhost:${PORT}/callback`);

});

**🧠 ¿Qué hace este webhook?**

* Escucha en /callback, que debe coincidir con tu redirect\_uri registrado en Mercado Libre.
* Recibe el code y lo pasa como argumento al script tokenExchange.js.
* Ejecuta el script automáticamente, sin intervención manual.
* Devuelve una respuesta clara al navegador.

**🛠️ ¿Y el tokenExchange.js?**

Ese script debe estar preparado para recibir el code como argumento (process.argv[2]) y hacer el POST a Mercado Libre para obtener el access\_token.

Si querés, te ayudo a revisar ese script también y validar que esté leyendo bien las variables de entorno (CLIENT\_ID, CLIENT\_SECRET, REDIRECT\_URI) y guardando los tokens.

¿Querés que lo revisemos juntos ahora?

Sí, Justo — cada vez que hacés una modificación local en tu código (por ejemplo, ajustás el webhookListener.js para que dispare automáticamente el intercambio de token), tenés que volver a subir esos cambios a GitHub **y asegurarte de que Render los reciba**.

🔁 **Pasos para actualizar tu app en Render:**

1. **Modificás tu código localmente.**
2. **Hacés commit y push al repositorio remoto en GitHub:**

git add .

git commit -m "Actualizo webhook para intercambio automático de token"

git push origin main

1. **Render detecta el cambio automáticamente** (si lo vinculaste a GitHub) y hace el redeploy.
   * Si no lo hace solo, podés ir a Render y tocar el botón **“Manual Deploy”** desde la sección de tu servicio.

📦 **Tip extra:**   
Si querés validar que el nuevo código se está ejecutando, podés agregar un console.log('Versión actual del webhookListener.js') y ver los logs en Render.

¿Querés que revisemos cómo está vinculado tu repo con Render o te muestro cómo hacer un redeploy manual por si no se actualiza solo?

¡Buena pregunta, Justo! Para verificar si Render aplicó los cambios que subiste a GitHub, podés seguir estos pasos dentro del panel de tu servicio en Render:

**🔍 ¿Cómo validar que se hizo el redeploy?**

1. **Entrá a tu servicio en Render**   
   Desde tu dashboard, hacé clic en el nombre del servicio que estás desplegando.
2. **Buscá la sección “Deploys” o “Events”**   
   Ahí vas a ver una lista de los últimos despliegues. Cada uno tiene:
   * Fecha y hora
   * Mensaje de commit (si lo hiciste desde GitHub)
   * Estado: ✅ “Live” o ❌ “Failed”
3. **Verificá los logs**   
   En la pestaña **“Logs”**, podés ver si el nuevo código se está ejecutando.  
   Buscá algún console.log que hayas agregado como marca, por ejemplo:
4. console.log('Versión actual del webhookListener.js');
5. **Forzar redeploy manual (si no se actualizó solo)**   
   Si Render no detectó el cambio automáticamente:
   * Tocá el botón **“Manual Deploy”**
   * Elegí el branch (main, por ejemplo)
   * Confirmá el despliegue

**🧠 Tip extra**

Si estás usando variables de entorno nuevas o modificaste el package.json, también podés revisar la sección **“Environment”** para asegurarte de que todo esté cargado correctamente.

¿Querés que te ayude a interpretar los logs o validar que el webhook esté funcionando como esperás?

Perfecto, Justo. Si ya configuraste las variables de entorno y subiste el código a GitHub, estás a un clic del 🚀 **deploy final**. Te explico cómo hacerlo paso a paso para que el webhook quede escuchando y el flujo OAuth2 esté listo para funcionar:

**✅ Últimos pasos para hacer el deploy en Render**

1. **Entrá a tu servicio en Render**
   * Desde tu dashboard, hacé clic en el nombre del servicio que creaste.
2. **Verificá que esté vinculado a tu repo de GitHub**
   * Si lo creaste desde GitHub, Render ya tiene acceso a tu código.
   * Si lo hiciste manualmente, asegurate de haber subido el código correcto.
3. **Tocá el botón “Deploy Web Service”**
   * Este es el paso que lanza tu app al entorno cloud.
   * Render va a instalar dependencias, levantar el servidor y mostrarte los logs.
4. **Validá los logs**
   * Buscá mensajes como 🚀 Webhook escuchando en http://localhost:3000/callback o el que hayas definido.
   * Si agregaste console.log('Versión actual del webhookListener.js'), debería aparecer.
5. **Probá el flujo OAuth2**
   * Iniciá el login desde Mercado Libre con tu client\_id y redirect\_uri.
   * Verificá que el navegador redirija a tu app en Render y que el code se reciba correctamente.
   * El webhook debería disparar automáticamente el script de intercambio y mostrarte el mensaje de éxito.

Si querés, puedo ayudarte a testear el flujo con una URL de prueba o revisar cómo guardar el token para el scraping. ¿Querés que armemos una prueba controlada para validar que todo esté funcionando?

¡Bien ahí, Justo! Si ya cargaste todo en Render pero aún no hiciste el deploy, entonces tu código está listo pero **todavía no está corriendo en la nube**. Eso significa que:

* Si salís a probar el flujo OAuth2 (por ejemplo, iniciar sesión con Mercado Libre), **no va a funcionar todavía**, porque el webhook (/callback) no está activo.
* El redirect\_uri que apunta a tu dominio en Render va a fallar o devolver error 404, porque el servidor aún no está desplegado.

**🔥 ¿Qué te conviene hacer ahora?**

👉 **Hacé el deploy primero.**   
Así Render levanta tu servidor, activa el webhook y podés probar el flujo completo con el client\_id, redirect\_uri y el intercambio automático de token.

Una vez desplegado:

* Probás el login con Mercado Libre.
* Validás que el navegador redirija a tu webhook.
* Confirmás que el script de intercambio se ejecuta y te devuelve el token.

¿Querés que te acompañe paso a paso en la prueba del flujo una vez que hagas el deploy? Podemos validar logs, tokens y hasta simular el scraping si querés.

¡Excelente, Justo! El log que compartiste muestra que el despliegue fue exitoso 🎉  
Tu app ya está corriendo en la nube, y Render instaló las dependencias, ejecutó el build y levantó el servicio sin errores.

**🔍 ¿Qué podés hacer ahora?**

1. **Probar el flujo OAuth2 en vivo**
   * Iniciá el login desde Mercado Libre con tu client\_id.
   * Asegurate de que el redirect\_uri apunte a tu dominio en Render, por ejemplo:  
     https://recordedbit-auth-hook.onrender.com/callback
   * El navegador debería redirigir ahí y tu webhook debería recibir el code.
2. **Verificar que se dispara el script de intercambio**
   * En los logs de Render, buscá algo como:  
     ✅ Código recibido: ...   
     📦 STDOUT: ...
   * Eso confirma que el tokenExchange.js se ejecutó correctamente.
3. **Guardar y visualizar el token**
   * Si estás guardando el token en archivo o base de datos, validá que se haya generado.
   * Si tenés un dashboard local en mente, podemos empezar a armarlo para visualizar logs y métricas.

¿Querés que te ayude a testear el flujo en vivo o revisar cómo guardar el token para el scraping? También podemos modularizar el dashboard si querés avanzar con eso.

¡Genial, Justo! Ya que el deploy fue exitoso, vamos a probar el flujo OAuth2 completo. Te dejo los pasos para validar que todo esté funcionando como esperás:

**🧪 Paso a paso para testear el flujo OAuth2 con Mercado Libre**

1. **Iniciá el login desde Mercado Libre**
   * Usá tu client\_id y el redirect\_uri que apunta a tu app en Render.
   * Ejemplo de URL para iniciar el login:
2. https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=TU\_CLIENT\_ID&redirect\_uri=https://TU\_APP\_RENDER.onrender.com/callback
3. **Autenticación del usuario**
   * Ingresás con una cuenta de Mercado Libre.
   * Al aceptar los permisos, Mercado Libre redirige al redirect\_uri con el code.
4. **Webhook recibe el code**
   * Tu webhookListener.js debería capturar ese code y mostrar en los logs algo como:
   * ✅ Código recibido: abc123
5. **Se dispara el script de intercambio**
   * El webhook ejecuta tokenExchange.js automáticamente.
   * En los logs deberías ver:
   * 📦 STDOUT: Token recibido correctamente
6. **Validás el token**
   * Confirmás que se guardó el access\_token y refresh\_token.
   * Si tenés un archivo, base de datos o consola, revisás que estén ahí.

Si querés, puedo ayudarte a construir un dashboard local para visualizar el estado del token, los logs y las métricas de scraping en tiempo real. También podemos modularizar el flujo para que sea más escalable.

¿Querés que armemos juntos el dashboard o preferís validar primero que el token se esté guardando bien?

¡Exacto, Justo! Ese redirect\_uri es el que tenés que registrar en Mercado Libre y usar en tu flujo OAuth2. Pero hay que reemplazar TU\_APP\_RENDER por el nombre real de tu servicio en Render.

**🔧 Ejemplo real del redirect\_uri**

Si tu servicio en Render se llama recordedbit-auth-hook, entonces el redirect\_uri sería:

https://recordedbit-auth-hook.onrender.com/callback

Este valor debe cumplir dos condiciones:

1. **Estar registrado en el Developer Portal de Mercado Libre**
   * En la configuración de tu aplicación, agregalo como redirect\_uri.
2. **Coincidir exactamente con el que usás en tu código**
   * En el tokenExchange.js, cuando hacés el POST para intercambiar el code, usás ese mismo redirect\_uri.

**🧠 Tip para evitar errores**

* No pongas / al final si no lo registraste así.
* Asegurate de que el dominio esté activo (ya hiciste el deploy).
* Podés probarlo directamente en el navegador para ver si el webhook responde.

¿Querés que revisemos juntos cómo estás armando la URL de autorización o cómo estás usando el redirect\_uri en el script de intercambio? También puedo ayudarte a validar que Mercado Libre lo acepte correctamente.

¡Obvio, Justo! Vamos a hacerlo bien simple y directo 👇

**✅ ¿Qué tenés que poner como redirect\_uri?**

Tenés que usar esta URL exacta:

https://recordedbit-auth-hook.onrender.com/callback

**📌 ¿Dónde la usás?**

1. **En Mercado Libre**
   * Vas al panel de desarrollador donde registraste tu app.
   * Pegás esa URL en el campo redirect\_uri.
2. **En tu código (tokenExchange.js)**
   * Usás esa misma URL cuando hacés el POST para pedir el token.

**🧪 ¿Cómo probás el login?**

Abrí esta URL en el navegador (reemplazá TU\_CLIENT\_ID por el tuyo):

https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=TU\_CLIENT\_ID&redirect\_uri=https://recordedbit-auth-hook.onrender.com/callback

Eso te lleva al login de Mercado Libre. Cuando aceptás, te redirige a tu webhook en Render y ahí se dispara el script que pide el token automáticamente.

¿Querés que te arme esa URL con tu client\_id real para que la copies y pegues directo?

¡Perfecto, Justo! Si ese es tu client\_id, entonces ya puedo armarte la URL completa para iniciar el login con Mercado Libre y probar el flujo OAuth2.

**🔗 URL lista para probar el login**

https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=TU\_CLIENT\_ID\_REAL&redirect\_uri=https://recordedbit-auth-hook.onrender.com/callback

Solo reemplazá TU\_CLIENT\_ID\_REAL por tu valor real (el que me acabás de confirmar). Por ejemplo, si tu client\_id es 123456789, la URL quedaría así:

https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=123456789&redirect\_uri=https://recordedbit-auth-hook.onrender.com/callback

**🧪 ¿Qué pasa cuando la abrís?**

1. Te lleva al login de Mercado Libre.
2. Iniciás sesión y aceptás los permisos.
3. Mercado Libre te redirige a tu webhook en Render (/callback).
4. Tu servidor recibe el code y dispara el script tokenExchange.js.
5. Si todo está bien

¡Listo Justo! Ya vi que tu servicio se llama mercadolibre-oauth-hook, así que el redirect\_uri correcto es:

https://mercadolibre-oauth-hook.onrender.com/callback

**🔗 URL completa para iniciar el login con Mercado Libre**

Solo reemplazá TU\_CLIENT\_ID por el tuyo real:

https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=TU\_CLIENT\_ID&redirect\_uri=https://mercadolibre-oauth-hook.onrender.com/callback

Por ejemplo, si tu client\_id es 123456789, la URL queda así:

https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=123456789&redirect\_uri=https://mercadolibre-oauth-hook.onrender.com/callback

**✅ ¿Qué pasa cuando la abrís?**

1. Te lleva al login de Mercado Libre.
2. Iniciás sesión y aceptás los permisos.
3. Te redirige a tu webhook en Render (/callback).
4. Tu servidor recibe el code y dispara automáticamente el script tokenExchange.js.
5. En los logs de Render deberías ver que se recibió el código y se ejecutó el intercambio.

Si querés, podés probarlo ahora y me contás qué ves en los logs. También puedo ayudarte a guardar el token o validar que esté listo para scraping. ¿Lo probamos juntos?

Gracias por compartir la captura, Justo. El error que te muestra Mercado Libre —“La aplicación no está preparada para conectarse a Mercado Libre”— indica que el redirect\_uri que estás usando **no coincide exactamente** con el que registraste en el panel de desarrollador de Mercado Libre.

**🔍 ¿Qué está mal en la URL?**

La URL que aparece en la barra del navegador es:

https://mercadolibre-oauth-hook-render.com/callback

Pero tu servicio en Render está desplegado en:

https://mercadolibre-oauth-hook.onrender.com/callback

💥 **La diferencia está en el dominio**: pusiste -render.com en lugar de .onrender.com.

**✅ Solución rápida**

1. **Entrá al panel de desarrollador de Mercado Libre**   
   [https://developers.mercadolibre.com.ar](https://developers.mercadolibre.com.ar/)
2. **Buscá tu aplicación registrada**
   * Editá la configuración.
   * En el campo redirect\_uri, pon exactamente:
3. https://mercadolibre-oauth-hook.onrender.com/callback
4. **Guardá los cambios y volvé a probar el login**   
   Usá esta URL para iniciar el flujo:
5. https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=TU\_CLIENT\_ID&redirect\_uri=https://mercadolibre-oauth-hook.onrender.com/callback

Una vez corregido, el flujo debería funcionar sin errores y tu webhook recibir el code. ¿Querés que te ayude a validar que el tokenExchange.js esté listo para recibirlo? También podemos revisar cómo guardar el token para el scraping.

¡Bien ahí, Justo! Detectarlo ahora te ahorra dolores de cabeza después. Vamos a dejarlo listo en minutos 💪

**🛠️ ¿Cómo registrar el redirect\_uri en Mercado Libre?**

1. **Entrá al portal de desarrolladores**   
   👉 [https://developers.mercadolibre.com.ar](https://developers.mercadolibre.com.ar/)
2. **Iniciá sesión y buscá tu aplicación**
   * Vas a ver una lista de tus apps registradas.
   * Hacé clic en la que estás usando para el flujo OAuth2.
3. **Editá la configuración**
   * Buscá el campo llamado redirect\_uri o “URL de redirección”.
   * Pegá esta URL exacta:
4. https://mercadolibre-oauth-hook.onrender.com/callback
5. **Guardá los cambios**
   * Mercado Libre ahora va a aceptar esa URL como válida para redirigir después del login.

**🧪 Después de registrar**

Podés volver a probar el login con esta URL:

https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=TU\_CLIENT\_ID&redirect\_uri=https://mercadolibre-oauth-hook.onrender.com/callback

Y si todo está bien, tu webhook va a recibir el code y disparar el intercambio automáticamente 🎯

¿Querés que te ayude a validar que el token se guarda bien o que el script responde como esperás? También podemos armar el dashboard local si querés avanzar con eso.

¡Bien ahí, Justo! Detectarlo ahora te ahorra dolores de cabeza después. Vamos a dejarlo listo en minutos 💪

**🛠️ ¿Cómo registrar el redirect\_uri en Mercado Libre?**

1. **Entrá al portal de desarrolladores**   
   👉 [https://developers.mercadolibre.com.ar](https://developers.mercadolibre.com.ar/)
2. **Iniciá sesión y buscá tu aplicación**
   * Vas a ver una lista de tus apps registradas.
   * Hacé clic en la que estás usando para el flujo OAuth2.
3. **Editá la configuración**
   * Buscá el campo llamado redirect\_uri o “URL de redirección”.
   * Pegá esta URL exacta:
4. https://mercadolibre-oauth-hook.onrender.com/callback
5. **Guardá los cambios**
   * Mercado Libre ahora va a aceptar esa URL como válida para redirigir después del login.

**🧪 Después de registrar**

Podés volver a probar el login con esta URL:

https://auth.mercadolibre.com.ar/authorization?response\_type=code&client\_id=TU\_CLIENT\_ID&redirect\_uri=https://mercadolibre-oauth-hook.onrender.com/callback

Y si todo está bien, tu webhook va a recibir el code y disparar el intercambio automáticamente 🎯

¿Querés que te ayude a validar que el token se guarda bien o que el script responde como esperás? También podemos armar el dashboard local si querés avanzar con eso.

26/9/25

¡Vamos a modularizarlo, Justo! Tu script index240925.js tiene varias responsabilidades: intercambio de token, refresco, scraping con API, fallback con Puppeteer… y eso es perfecto para dividirlo en módulos reutilizables. Así ganás claridad, escalabilidad y control visual desde el dashboard.

## 🧩 Estructura modular recomendada

/scraper

├── index.js ← script principal que ejecuta el scraping

├── tokenManager.js ← manejo de token: carga, refresco, guardado

├── apiClient.js ← consulta a la API de Mercado Libre

├── puppeteerFallback.js ← scraping con Puppeteer si la API falla

├── utils.js ← funciones auxiliares (extraer ID, logs, etc.)

## 🔧 Paso 1: tokenManager.js

import fs from 'fs';

import fetch from 'node-fetch';

import dotenv from 'dotenv';

dotenv.config();

const { CLIENT\_ID, CLIENT\_SECRET, REDIRECT\_URI } = process.env;

export function loadToken() {

if (fs.existsSync('token.json')) {

return JSON.parse(fs.readFileSync('token.json'));

}

return null;

}

export async function refreshToken(refresh\_token) {

const res = await fetch('https://api.mercadolibre.com/oauth/token', {

method: 'POST',

headers: { 'Content-Type': 'application/x-www-form-urlencoded' },

body: new URLSearchParams({

grant\_type: 'refresh\_token',

client\_id: CLIENT\_ID,

client\_secret: CLIENT\_SECRET,

redirect\_uri: REDIRECT\_URI,

refresh\_token

})

});

const data = await res.json();

if (data.access\_token) {

fs.writeFileSync('token.json', JSON.stringify(data, null, 2));

console.log('🔄 Token actualizado');

return data.access\_token;

} else {

console.error('❌ Error al refrescar token:', data);

return null;

}

}

## 🔧 Paso 2: apiClient.js

import fetch from 'node-fetch';

import { loadToken, refreshToken } from './tokenManager.js';

export async function getItemData(itemId) {

let tokenData = loadToken();

if (!tokenData) {

console.error("❌ No hay token guardado.");

return null;

}

let access\_token = tokenData.access\_token;

const url = `https://api.mercadolibre.com/items/${itemId}`;

const descriptionUrl = `${url}/description`;

const [itemRes, descRes] = await Promise.all([

fetch(url, { headers: { Authorization: `Bearer ${access\_token}` } }),

fetch(descriptionUrl, { headers: { Authorization: `Bearer ${access\_token}` } })

]);

if (itemRes.status === 401 || descRes.status === 401) {

console.warn("⚠️ Token expirado. Intentando refrescar...");

access\_token = await refreshToken(tokenData.refresh\_token);

if (!access\_token) return null;

return await getItemData(itemId); // Reintenta

}

if (!itemRes.ok || !descRes.ok) {

console.error(`❌ Error en API: ${itemRes.status} / ${descRes.status}`);

return null;

}

const itemData = await itemRes.json();

const descData = await descRes.json();

return {

titulo: itemData.title,

precio: itemData.price,

moneda: itemData.currency\_id,

ubicacion: itemData.seller\_address?.city?.name,

descripcion: descData.plain\_text,

imagenes: itemData.pictures?.map(pic => pic.url) ?? []

};

}

## 🔧 Paso 3: puppeteerFallback.js

import puppeteer from 'puppeteer';

export async function scrapeWithPuppeteer(url) {

const browser = await puppeteer.launch({ headless: true });

const page = await browser.newPage();

try {

await page.goto(url, { waitUntil: 'networkidle2' });

await page.waitForFunction(() => window.\_\_PRELOADED\_STATE\_\_);

const data = await page.evaluate(() => window.\_\_PRELOADED\_STATE\_\_.item);

return {

titulo: data?.title,

precio: data?.price,

moneda: data?.currency\_id,

ubicacion: data?.seller\_address?.city?.name,

descripcion: data?.description,

imagenes: data?.pictures?.map(pic => pic.url) ?? []

};

} catch (error) {

console.error("❌ Error en Puppeteer:", error.message);

return null;

} finally {

await browser.close();

}

}

## 🔧 Paso 4: utils.js

export function extractItemId(url) {

const match = url.match(/MLA-\d+/);

return match ? match[0].replace("-", "") : null;

}

## 🔧 Paso 5: index.js (script principal)

import { extractItemId } from './utils.js';

import { getItemData } from './apiClient.js';

import { scrapeWithPuppeteer } from './puppeteerFallback.js';

const url = "https://departamento.mercadolibre.com.ar/MLA-1413050342-departamentos-semipisos-en-venta-a-estrenar-de-3-ambientes-en-recoleta-gran-categoria-\_JM";

const itemId = extractItemId(url);

(async () => {

const apiData = await getItemData(itemId);

if (apiData) {

console.log("📡 Datos desde la API:", apiData);

return;

}

console.log("🔄 API falló, usando fallback...");

const fallbackData = await scrapeWithPuppeteer(url);

console.log("📦 Datos desde fallback:", fallbackData);

})();

## ✅ Beneficios de esta modularización

* Podés testear cada módulo por separado
* Podés reutilizar tokenManager en el dashboard
* Podés escalar el scraping con múltiples URLs
* Podés integrar métricas fácilmente

¿Querés que preparemos una versión que devuelva los datos como JSON para integrarlo al dashboard? También podemos guardar los resultados en un archivo o mostrarlos en una tabla visual. Vos decime cómo seguimos.

¡Perfecto, Justo! Si querés que el archivo index.js simplemente ejecute tu scraper modularizado (index240925.js), te armo una versión mínima y funcional que podés usar como entrypoint en Render o desde tu dashboard.

## 🧩 index.js básico para ejecutar el scraper

import { exec } from 'child\_process';

import express from 'express';

const app = express();

const PORT = process.env.PORT || 3000;

// ✅ Ruta de prueba

app.get('/', (req, res) => {

res.send('Servidor activo ✅');

});

// 🚀 Ruta para ejecutar el scraper

app.get('/scrape', (req, res) => {

exec('node index240925.js', (error, stdout, stderr) => {

if (error) {

console.error('❌ Error al ejecutar el scraper:', error);

return res.status(500).send('Error al ejecutar el scraper.');

}

console.log('📦 STDOUT:', stdout);

console.error('⚠️ STDERR:', stderr);

res.send('Scraper ejecutado correctamente.');

});

});

// 🔊 Levantar el servidor

app.listen(PORT, () => {

console.log(`🟢 Servidor corriendo en http://localhost:${PORT}`);

});

### ✅ ¿Qué hace esto?

* Levanta un servidor Express en el puerto 3000 (o el que Render defina).
* Te permite probar que el servidor está vivo con /.
* Ejecuta tu script index240925.js desde la ruta /scrape.

### 🧪 ¿Cómo lo probás?

Una vez subido a GitHub y redeployado en Render:

* Entrás a https://mercadolibre-oauth-hook.onrender.com/ → ves “Servidor activo ✅”
* Entrás a https://mercadolibre-oauth-hook.onrender.com/scrape → ejecuta el scraper

¿Querés que el scraper devuelva los datos como JSON en lugar de solo logs? También podemos guardar los resultados en un archivo o mostrarlos en el dashboard. Vos decime cómo seguimos.