# **PATCH**

## **📋 Objective:**

Fix TitleTesterPro’s OAuth YouTube integration so:  
 ✅ Users log in with Google → tokens & YouTube channel reliably captured in DB.  
 ✅ Supports refresh tokens & proactively refreshes access tokens before expiry.  
 ✅ Handles users with multiple (Brand) channels.  
 ✅ Retries transient YouTube API errors.  
 ✅ Encrypts tokens before storage & logs securely.  
 ✅ Follows Google & Supabase best practices (July 2025).

# **🗂️ File & Feature Overview**

### **Key Files To Create/Replace:**

pgsql

CopyEdit

/pages/api/auth/callback.ts <-- OAuth callback handler

/pages/select-channel.tsx <-- UI page if user has multiple channels

/pages/api/save-channel.ts <-- API to save channel selection

/lib/utils/refreshAccessToken.ts <-- helper to refresh token

/lib/db.ts <-- storage helpers

/migrations/20250711\_add\_temp\_oauth.sql <-- DB migration for temp storage

# **🔧 Step-by-Step Implementation**

## **1️⃣ Google Cloud Console & Supabase**

### **OAuth Consent:**

✅ Approve these scopes in your OAuth app:

nginx

CopyEdit

openid email profile

https://www.googleapis.com/auth/youtube

https://www.googleapis.com/auth/youtube.readonly

https://www.googleapis.com/auth/youtube.force-ssl

✅ On Supabase → Authentication → Providers → Google:

* Add **Client ID** & **Client Secret**.
* Set access\_type=offline and prompt=consent in the signInWithOAuth call (so you get a refresh\_token).
* Add your app’s domain and /api/auth/callback to authorized redirect URIs.

## **2️⃣ .env Variables**

Add to .env:

bash

CopyEdit

NEXT\_PUBLIC\_SUPABASE\_URL=<your\_supabase\_url>

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=<your\_anon\_key>

SUPABASE\_SERVICE\_ROLE\_KEY=<your\_service\_role\_key>

GOOGLE\_CLIENT\_ID=<your\_google\_client\_id>

GOOGLE\_CLIENT\_SECRET=<your\_google\_client\_secret>

GOOGLE\_REDIRECT\_URI=https://yourapp.com/api/auth/callback

ENCRYPTION\_KEY=<some\_strong\_secret> # for encryptToken()

## **3️⃣ Database Migration**

Create table to temporarily store tokens & multiple channel choices:

sql

CopyEdit

-- migrations/20250711\_add\_temp\_oauth.sql

CREATE TABLE temp\_oauth (

id SERIAL PRIMARY KEY,

user\_id UUID NOT NULL UNIQUE REFERENCES users(id) ON DELETE CASCADE,

access\_token TEXT NOT NULL,

refresh\_token TEXT NOT NULL,

expires\_at BIGINT NOT NULL,

channels JSONB NOT NULL,

created\_at TIMESTAMP DEFAULT NOW()

);

Ensure accounts table includes:

sql

CopyEdit

ALTER TABLE accounts

ADD COLUMN IF NOT EXISTS access\_token TEXT,

ADD COLUMN IF NOT EXISTS refresh\_token TEXT,

ADD COLUMN IF NOT EXISTS youtube\_channel\_id TEXT,

ADD COLUMN IF NOT EXISTS youtube\_channel\_title TEXT,

ADD COLUMN IF NOT EXISTS expires\_at BIGINT;

## **4️⃣ Frontend — Login Button**

ts

CopyEdit

supabase.auth.signInWithOAuth({

provider: 'google',

options: {

scopes: [

'openid',

'email',

'profile',

'https://www.googleapis.com/auth/youtube.readonly',

'https://www.googleapis.com/auth/youtube.force-ssl'

].join(' '),

queryParams: { access\_type: 'offline', prompt: 'consent' }

}

});

## **5️⃣ OAuth Callback**

📄 /pages/api/auth/callback.ts

ts

CopyEdit

import { createClient } from '@supabase/supabase-js';

import axios from 'axios';

import { google } from 'googleapis';

import { encryptToken } from '@/lib/utils/encryption';

import { storage } from '@/lib/db';

const supabase = createClient(process.env.NEXT\_PUBLIC\_SUPABASE\_URL!, process.env.SUPABASE\_SERVICE\_ROLE\_KEY!);

export default async function handler(req, res) {

try {

const { code } = req.query;

if (!code) return res.status(400).json({ error: 'Missing OAuth code' });

const { data: session, error } = await supabase.auth.exchangeCodeForSession(code);

if (error || !session) {

console.error('Supabase exchange failed', error);

return res.status(500).json({ error: 'OAuth session exchange failed' });

}

const user = session.user;

let accessToken = session.provider\_token;

let refreshToken = session.provider\_refresh\_token;

let expiresIn = 3600;

if (!accessToken || !refreshToken) {

console.warn('Falling back to direct exchange');

const params = new URLSearchParams({

code: code.toString(),

client\_id: process.env.GOOGLE\_CLIENT\_ID!,

client\_secret: process.env.GOOGLE\_CLIENT\_SECRET!,

redirect\_uri: process.env.GOOGLE\_REDIRECT\_URI!,

grant\_type: 'authorization\_code',

});

const { data } = await axios.post('https://oauth2.googleapis.com/token', params, {

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

});

accessToken = data.access\_token;

refreshToken = data.refresh\_token;

expiresIn = data.expires\_in;

if (!accessToken || !refreshToken) {

console.error('Failed to obtain tokens');

return res.status(500).json({ error: 'OAuth token exchange failed' });

}

}

const youtube = google.youtube({ version: 'v3', auth: accessToken });

let retries = 3, channelsResponse;

while (retries--) {

try {

channelsResponse = await youtube.channels.list({ part: ['snippet'], mine: true });

break;

} catch (err) {

console.warn(`YouTube API error (retries left: ${retries}):`, err.message);

if (!retries) throw err;

}

}

const channels = channelsResponse.data.items;

if (!channels || channels.length === 0) {

console.error('No YouTube channels found');

return res.status(500).json({ error: 'No YouTube channels found' });

}

const encAccess = encryptToken(accessToken);

const encRefresh = encryptToken(refreshToken);

const expiresAt = Date.now() + expiresIn \* 1000;

if (channels.length > 1) {

await storage.saveTempTokens(user.id, {

accessToken: encAccess,

refreshToken: encRefresh,

expiresAt,

channels,

});

return res.redirect('/select-channel');

}

const channel = channels[0];

await storage.upsertAccount(user.id, {

accessToken: encAccess,

refreshToken: encRefresh,

youtubeChannelId: channel.id,

youtubeChannelTitle: channel.snippet.title,

expiresAt,

});

console.log(`✅ Saved tokens & channel (${channel.snippet.title})`);

return res.redirect('/dashboard');

} catch (err) {

console.error('OAuth callback error:', err);

return res.status(500).json({ error: 'Internal server error during OAuth callback' });

}

}

## **6️⃣ Channel Selector**

📄 /pages/select-channel.tsx

tsx

CopyEdit

export default function SelectChannel({ channels }) {

return (

<div>

<h1>Select Your YouTube Channel</h1>

{channels.map(c => (

<form method="POST" action="/api/save-channel" key={c.id}>

<input type="hidden" name="channelId" value={c.id} />

<input type="hidden" name="channelTitle" value={c.snippet.title} />

<button type="submit">

<img src={c.snippet.thumbnails.default.url} alt={c.snippet.title} />

{c.snippet.title}

</button>

</form>

))}

</div>

);

}

📄 /pages/api/save-channel.ts

ts

CopyEdit

import { storage } from '@/lib/db';

export default async function handler(req, res) {

if (req.method !== 'POST') return res.status(405).json({ error: 'Method not allowed' });

const { userId, channelId, channelTitle } = req.body;

const tempTokens = await storage.getTempTokens(userId);

if (!tempTokens) return res.status(400).json({ error: 'Missing temporary tokens' });

await storage.upsertAccount(userId, {

accessToken: tempTokens.accessToken,

refreshToken: tempTokens.refreshToken,

youtubeChannelId: channelId,

youtubeChannelTitle: channelTitle,

expiresAt: tempTokens.expiresAt,

});

await storage.deleteTempTokens(userId);

return res.redirect('/dashboard');

}

## **7️⃣ Token Refresh Logic**

📄 /lib/utils/refreshAccessToken.ts

ts

CopyEdit

import axios from 'axios';

export async function refreshAccessToken(refreshToken: string) {

const params = new URLSearchParams({

client\_id: process.env.GOOGLE\_CLIENT\_ID!,

client\_secret: process.env.GOOGLE\_CLIENT\_SECRET!,

refresh\_token: refreshToken,

grant\_type: 'refresh\_token',

});

const { data } = await axios.post(

'https://oauth2.googleapis.com/token',

params,

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

);

return { accessToken: data.access\_token, expiresIn: data.expires\_in };

}

## **8️⃣ Storage Helpers**

📄 /lib/db.ts

ts

CopyEdit

export const storage = {

async upsertAccount(userId, data) {

const existing = await db.account.findFirst({ where: { userId } });

if (existing) {

await db.account.update({ where: { id: existing.id }, data });

} else {

await db.account.create({ data: { userId, ...data } });

}

},

async saveTempTokens(userId, data) {

await db.temp\_oauth.upsert({

where: { user\_id: userId },

update: data,

create: { user\_id: userId, ...data },

});

},

async getTempTokens(userId) {

return db.temp\_oauth.findFirst({ where: { user\_id: userId } });

},

async deleteTempTokens(userId) {

await db.temp\_oauth.delete({ where: { user\_id: userId } });

},

};

# **🧪 Testing Scenarios**

✅ **Login → Dashboard loads videos.** ✅ Login → Multiple channels → User selects → Dashboard loads.  
 ✅ Token expired → API automatically refreshes.  
 ✅ Broken Supabase session → fallback exchange works.  
 ✅ YouTube API transient failure → retries succeed.  
 ✅ Check accounts table: accessToken, refreshToken, youtubeChannelId, youtubeChannelTitle, expires\_at all populated.

# 