## **1. FRONTEND: Extract and Persist Google Tokens After Login**

In your AuthCallback.tsx (or the component/page handling the OAuth redirect):

jsx

CopyEdit

* import { useEffect } from "react";
* import { supabase } from "../lib/supabaseClient"; // Adjust path to your setup
* export default function AuthCallback() {
* useEffect(() => {
* async function persistYouTubeTokens() {
* const { data: { session } } = await supabase.auth.getSession();
* if (session && session.provider\_token && session.provider\_refresh\_token) {
* await fetch("/api/accounts/save-tokens", {
* method: "POST",
* headers: { "Content-Type": "application/json" },
* credentials: "include", // critical for cookies/session!
* body: JSON.stringify({
* accessToken: session.provider\_token,
* refreshToken: session.provider\_refresh\_token,
* }),
* });
* }
* // Optional: Redirect to dashboard or desired page after tokens are saved
* window.location.href = "/dashboard";
* }
* persistYouTubeTokens();
* }, []);
* return <div>Finishing login…</div>;
* }

**Place this in your OAuth callback effect, after a successful login/session set.**

## **2. BACKEND: Save Tokens Endpoint**

In your Express/Node backend (e.g., add to server/routes.ts):

js

CopyEdit

* // Add this endpoint to save tokens after login
* app.post("/api/accounts/save-tokens", requireAuth, async (req, res) => {
* const { accessToken, refreshToken } = req.body;
* const userId = req.user.id; // populated by requireAuth middleware
* if (!accessToken || !refreshToken) {
* return res.status(400).json({ error: "Missing tokens" });
* }
* // Save or update tokens in your accounts table
* await db.accounts.upsert({
* where: { userId, provider: "google" },
* update: { accessToken, refreshToken },
* create: { userId, provider: "google", accessToken, refreshToken },
* });
* return res.json({ success: true });
* });

**Replace the db.accounts.upsert logic with your ORM/db layer as appropriate (e.g., Prisma, Drizzle, Knex, raw SQL).**

## **3. BACKEND: Use Tokens for YouTube API Calls**

Anywhere you need a user’s YouTube token (e.g., in your video selection tool or youtubeService):

js

CopyEdit

* // Example: Fetch user's Google account tokens for YouTube API calls
* const account = await db.accounts.findFirst({ where: { userId, provider: "google" } });
* if (!account || !account.accessToken) {
* return res.status(401).json({ error: "YouTube account not connected. Please reconnect." });
* }
* // Use account.accessToken (and refreshToken for refresh)

## **4. BACKEND: Token Refresh Logic (Recommended)**

Add this helper so you can refresh an expired access token using Google’s OAuth endpoint:

js

CopyEdit

* async function refreshGoogleAccessToken(refreshToken) {
* const response = await fetch("https://oauth2.googleapis.com/token", {
* method: "POST",
* headers: { "Content-Type": "application/x-www-form-urlencoded" },
* body: 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 response.json();
* if (data.access\_token) {
* return data.access\_token;
* }
* throw new Error("Failed to refresh Google access token");
* }
* **On access token expiry, use this to get a new token, then update your database.**

## **5. FRONTEND & API: Ensure Credentials and Cookies**

* All fetch/Axios/XHR to your API **must** use credentials: "include" so browser sends session cookies!
* **Backend cookies (sb-access-token, etc.)** must be SameSite: 'none' and secure: true in prod for cross-domain use.
* **Double-check in browser DevTools > Network tab** that API requests send these cookies.

## **6. REMOVE/IGNORE Old YouTube Token Logic**

* Delete any code that tries to use Supabase’s provider\_token outside the initial login callback.
* Stop trying to fetch YouTube tokens from Supabase session after login—they are only present at login.

## **7. OPTIONAL: “Reconnect Google” Button**

Let users reconnect/reauth if their tokens become invalid:

jsx

CopyEdit

* <Button onClick={() => supabase.auth.signInWithOAuth({
* provider: 'google',
* options: { prompt: 'consent', access\_type: 'offline' }
* })}>
* Reconnect Google Account
* </Button>

## **8. VIDEO SELECTION TOOL FIX**

* Make sure /api/videos/channel and /api/videos/recent endpoints now use the tokens from your accounts DB, **not the Supabase session**.
* This ensures thumbnails, titles, and YouTube analytics work for all authenticated users.

# **Final AI Agent Prompt (Copy-Paste for Replit AI)**

**You are a senior full-stack engineer. Fix the YouTube OAuth token crisis for TitleTesterPro, enabling persistent, secure Google tokens so users can fetch videos, thumbnails, and run tests.**

**Do the following:**

1. In the frontend OAuth callback handler (AuthCallback.tsx), immediately after Supabase login/session is set, extract provider\_token and provider\_refresh\_token and POST them to a new /api/accounts/save-tokens endpoint with credentials included.
2. In the backend, implement /api/accounts/save-tokens so that for the logged-in user, access and refresh tokens are upserted into the accounts table (by userId, provider='google').
3. Update all YouTube API calls to always use these saved tokens from the accounts table, not the ephemeral Supabase session.
4. Add a refreshGoogleAccessToken helper to refresh expired tokens as needed, and update your DB when a new access token is received.
5. Ensure all fetch/XHR calls use credentials: "include"; make cookies SameSite: 'none' and secure: true in production.
6. Remove any old code that tries to fetch YouTube tokens from Supabase session after login.
7. Instruct users to reconnect Google account with full consent if their token is missing or expired.
8. The video selection tool must now fetch data from the saved tokens and work for all users.

**All code must match TitleTesterPro’s file structure and naming conventions. Do not change other business logic or database tables. This must solve “No active session” and enable the video selection tool and dashboard as shown in user screenshots.**