# **📂 Final Code Bundle**

## **📄 /api/auth/callback.ts**

ts

CopyEdit

import { supabase } from '@/lib/supabaseAdmin';

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

import { google } from 'googleapis';

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

export default async function handler(req, res) {

const { code, state, error } = req.query;

if (error) {

console.error('OAuth Error:', error);

return res.redirect('/error?reason=' + encodeURIComponent(error));

}

if (!code) {

console.error('OAuth Callback Missing Code');

return res.redirect('/error?reason=missing\_code');

}

try {

const { data: session, error: supabaseError } =

await supabase.auth.exchangeCodeForSession(code);

if (supabaseError || !session) {

console.error('Supabase session exchange error', supabaseError);

return res.redirect('/error?reason=exchange\_failed');

}

const user = session.user;

const token = session.provider\_token;

const refresh = session.provider\_refresh\_token;

if (!token || !refresh) {

console.error('Tokens missing after exchange');

return res.redirect('/error?reason=tokens\_missing');

}

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

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

const channels = channelsRes.data.items;

if (!channels?.length) {

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

return res.redirect('/error?reason=no\_channels');

}

const encAccess = encryptToken(token);

const encRefresh = encryptToken(refresh);

const expiresAt = Date.now() + 3600 \* 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,

});

res.redirect('/dashboard');

} catch (err) {

console.error('OAuth Callback Error:', err);

res.redirect('/error?reason=server\_error');

}

}

## **📄 /pages/error.tsx**

tsx

CopyEdit

export default function ErrorPage({ reason }) {

return (

<div style={{ textAlign: 'center', padding: '2rem' }}>

<h1>😕 Authentication Error</h1>

<p>{reason || 'An unknown error occurred.'}</p>

<p>This can happen if your permissions expired or were revoked.</p>

<ol style={{ textAlign: 'left', display: 'inline-block' }}>

<li>

Go to{' '}

<a href="https://myaccount.google.com/permissions" target="\_blank">

Google Account Permissions

</a>

</li>

<li>Remove “TitleTesterPro”</li>

<li>Come back and log in again.</li>

</ol>

<a href="/login">

<button>Retry Login</button>

</a>

</div>

);

}

ErrorPage.getInitialProps = ({ query }) => ({ reason: query.reason });

## **📄 Incremental Consent: Frontend Logic**

### **Phase 1 Button**

ts

CopyEdit

supabase.auth.signInWithOAuth({

provider: 'google',

options: {

scopes: ['openid', 'email', 'profile'].join(' '),

},

});

### **Phase 2 Button**

Once logged in & dashboard loads, if no youtube\_channel\_id:

ts

CopyEdit

supabase.auth.signInWithOAuth({

provider: 'google',

options: {

scopes: [

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

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

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

'https://www.googleapis.com/auth/yt-analytics.readonly',

].join(' '),

queryParams: {

access\_type: 'offline',

prompt: 'consent',

},

},

});

## **📄 /pages/select-channel.tsx**

tsx

CopyEdit

export default function SelectChannel({ channels }) {

return (

<div style={{ textAlign: 'center' }}>

<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>

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

{c.snippet.title}

</button>

</form>

))}

</div>

);

}

## **📄 /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).end();

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

const temp = await storage.getTempTokens(userId);

if (!temp) return res.redirect('/error?reason=temp\_tokens\_missing');

await storage.upsertAccount(userId, {

accessToken: temp.accessToken,

refreshToken: temp.refreshToken,

youtubeChannelId: channelId,

youtubeChannelTitle: channelTitle,

expiresAt: temp.expiresAt,

});

await storage.deleteTempTokens(userId);

res.redirect('/dashboard');

}

## **📄 /lib/utils/refreshAccessToken.ts**

ts

CopyEdit

import axios from 'axios';

import { encryptToken } from './encryption';

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);

return {

accessToken: encryptToken(data.access\_token),

expiresAt: Date.now() + data.expires\_in \* 1000,

};

}

## **📝 SQL to Clear Bad Tokens**

sql

CopyEdit

DELETE FROM accounts WHERE user\_id = '<user-id>';

DELETE FROM temp\_oauth WHERE user\_id = '<user-id>';

## **📋 Testing Checklist**

✅ Login Phase 1 → returns session & user info.  
✅ Dashboard → shows prompt if no YouTube connected.  
✅ Click “Connect YouTube” → triggers Phase 2 → saves tokens & channel.  
✅ Multi-channel → select → saved properly.  
✅ Bad tokens → error page with revoke instructions.  
✅ Refresh works → YouTube API calls succeed even after access\_token expires.

# **🚀 Notes for the Agent**

* All scopes have already been approved → nothing needs to change in Google Cloud.
* Implement Phase 1 & Phase 2 as separate buttons or flows.
* Add /error page & redirect to it if anything fails.
* Log errors for debugging.
* Use .env for all keys & secrets.
* Run the SQL migration to ensure temp\_oauth table exists.