YouTube Channel Display Fix - Accounts-Based Architecture

🏗️ Understanding Your Architecture

If you've migrated from "users" to "accounts", this means:

- **Main table is now** `accounts` - Contains user profiles AND YouTube channel info
- **No separate** `users` **table** - Everything is consolidated
- **All methods use account terminology** - `getAccountById`, `updateAccount`, etc.

🔧 Updated Implementation

1. OAuth Callback Handler (Updated for Accounts)

typescript

```
// server/routes/auth-supabase.ts
router.get('/api/auth/callback/google', async (req: Request, res: Response) => {
  const { code, error } = req.query;

  if (error) {
    console.error('❌ [CALLBACK] OAuth callback error:', error);
    return res.redirect('/login?error=oauth_error');
  }

  if (!code) {
    return res.redirect('/login?error=no_code');
  }

  try {
    // Exchange code for session
    const { data, error: sessionError } = await supabase.auth.exchangeCodeForSession(code as string);

    if (sessionError || !data.session) {
      console.error('❌ [CALLBACK] Session exchange error:', sessionError);
      return res.redirect('/login?error=session_error');
    }

    console.log('✅ [CALLBACK] Session exchanged successfully for user:', data.session.user.email);

    // Set secure cookies
    res.cookie('sb-access-token', data.session.access_token, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'lax',
      maxAge: 60 * 60 * 24 * 7, // 7 days
      path: '/'
    });

    res.cookie('sb-refresh-token', data.session.refresh_token, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'lax',
      maxAge: 60 * 60 * 24 * 30, // 30 days
      path: '/'
    });

    // IMPORTANT: Using accounts table, not users
    const { user } = data.session;
```

```
let dbAccount = await storage.getAccountByEmail(user.email!);

if (!dbAccount) {
  console.log('👤 [CALLBACK] Creating new account:', user.email);
  // Create new account with Google profile info
  dbAccount = await storage.createAccount({
    id: user.id,
    email: user.email!,
    name: user.user_metadata.full_name || user.email!.split('@')[0],
    image: user.user_metadata.avatar_url,
    // OAuth fields
    provider: 'google',
    providerAccountId: user.id,
    type: 'oauth',
    // Default subscription fields
    subscriptionTier: 'free',
    subscriptionStatus: 'inactive'
  });
} else {
  console.log('👤 [CALLBACK] Account already exists:', user.email);
}

// CRITICAL: Fetch YouTube channel information
console.log('📺 [CALLBACK] Fetching YouTube channel info...');
try {
  const providerToken = data.session.provider_token;
  const providerRefreshToken = data.session.provider_refresh_token;

  if (providerToken) {
    // Use Google API to fetch YouTube channel
    const channelInfo = await googleAuthService.getYouTubeChannel(providerToken);

    if (channelInfo && channelInfo.id) {
      console.log('✅ [CALLBACK] YouTube channel found:', channelInfo.title);

      // Update account (not user!) with YouTube channel info
      await storage.updateAccount(dbAccount.id, {
        youtube_channel_id: channelInfo.id,
        youtube_channel_title: channelInfo.title || 'YouTube Channel',
        // Update OAuth tokens
        accessToken: providerToken,
        refreshToken: providerRefreshToken,
        expiresAt: data.session.expires_at ? new Date(data.session.expires_at * 1000).getTime() : null,
        updatedAt: new Date()
```

```
        });

        console.log('💾 [CALLBACK] YouTube channel info saved to accounts table');
      }
    }
  } catch (channelError: any) {
    console.error('❌ [CALLBACK] YouTube channel fetch failed:', channelError.message);
  }

  console.log('✅ [CALLBACK] OAuth complete, redirecting to dashboard');
  res.redirect('/dashboard');

 } catch (error) {
  console.error('💥 [CALLBACK] Fatal OAuth callback error:', error);
  res.redirect('/login?error=processing_error');
 }
});
```

2. Updated Storage Interface

typescript

```
// server/storage.ts - Account-based methods
export interface IStorage {
 // Account methods (replacing user methods)
 getAccount(id: string): Promise<Account | undefined>;
 getAccountByEmail(email: string): Promise<Account | undefined>;
 createAccount(account: InsertAccount): Promise<Account>;
 updateAccount(id: string, account: Partial<Account>): Promise<Account>;

 // OAuth specific
 getAccountByProvider(provider: string, providerAccountId: string): Promise<Account | undefined>;
 updateAccountTokens(accountId: string, tokens: {
  accessToken: string;
  refreshToken: string;
  expiresAt: number | null;
 }): Promise<Account>;

 // Other methods remain the same...
}
```

3. YouTube Service Updates

typescript

```
// server/youtubeService.ts
async withTokenRefresh<T>(
  accountId: string, // Changed from userId to accountId
  operation: (tokens: { accessToken: string; refreshToken: string }) => Promise<T>
): Promise<T> {
  console.log(`🔑 [YOUTUBE] Starting operation for account ${accountId}`);

  try {
    // Get account directly (not through a separate OAuth table)
    const account = await storage.getAccount(accountId);

    if (!account || !account.accessToken) {
      throw new Error('No Google authentication found. Please reconnect your YouTube account.');
    }

    try {
      return await operation({
        accessToken: account.accessToken,
        refreshToken: account.refreshToken || ''
      });
    } catch (error: any) {
      if (error.code === 401) {
        // Token expired, try refresh
        if (account.refreshToken) {
          const refreshedTokens = await googleAuthService.refreshAccessToken(account.refreshToken);

          // Update account with new tokens
          await storage.updateAccount(accountId, {
            accessToken: refreshedTokens.access_token!,
            expiresAt: refreshedTokens.expiry_date || null
          });

          // Retry operation
          return await operation({
            accessToken: refreshedTokens.access_token!,
            refreshToken: account.refreshToken
          });
        }
      }
      throw error;
    }
  } catch (error: any) {
    console.error('[YOUTUBE] Operation failed:', error);
```

```
    throw error;
  }
}
```

4. API Endpoint Updates

typescript

```typescript
// server/routes.ts
router.get('/api/auth/account', async (req: Request, res: Response) => {
  const token = req.cookies['sb-access-token'];

  if (!token) {
    return res.status(401).json({ error: 'Not authenticated' });
  }

  try {
    const { data: { user }, error } = await supabase.auth.getUser(token);

    if (error || !user) {
      return res.status(401).json({ error: 'Invalid session' });
    }

    // Get account from database
    const dbAccount = await storage.getAccountByEmail(user.email!);

    if (dbAccount) {
      console.log('💻 [AUTH-ACCOUNT] Account info:');
      console.log('   Email:', dbAccount.email);
      console.log('   Name:', dbAccount.name);
      console.log('   YouTube Channel:', dbAccount.youtube_channel_title || 'NOT SET');
      console.log('   Channel ID:', dbAccount.youtube_channel_id || 'NOT SET');
    }

    res.json({
      account: dbAccount, // Changed from 'user' to 'account'
      session: {
        access_token: token,
        expires_at: user.exp
      }
    });

  } catch (error) {
    console.error('Get account error:', error);
    res.status(401).json({ error: 'Session invalid' });
  }
});
```

5. Frontend Updates

typescript

```typescript
// Update authService to use account terminology
// client/src/lib/auth.ts
export const authService = {
  async getCurrentAccount() { // Changed from getCurrentUser
    const response = await fetch('/api/auth/account', { // Changed endpoint
      credentials: 'include'
    });

    if (!response.ok) {
      throw new Error('Not authenticated');
    }

    const data = await response.json();
    return data.account; // Changed from data.user
  },

  // ... other methods
};

// Update React Query keys
const { data: account } = useQuery({
  queryKey: ['/api/auth/account'], // Changed from /api/auth/user
  queryFn: () => authService.getCurrentAccount()
});
```

6. Display Component Updates

typescript

```tsx
// client/src/components/AccountDisplay.tsx (renamed from UserDisplay)
export function getDisplayName(account: Account | null | undefined): string {
  if (!account) return 'Account';

  // Priority order:
  // 1. YouTube channel name
  // 2. Account name
  // 3. Email username
  // 4. Default

  if (account.youtube_channel_title) {
    return account.youtube_channel_title;
  }

  if (account.name) {
    return account.name;
  }

  if (account.email) {
    return account.email.split('@')[0];
  }

  return 'Account';
}

export function AccountDisplay({ account, showChannelBadge = true }: AccountDisplayProps) {
  const displayName = getDisplayName(account);
  const isYouTubeCreator = !!account.youtube_channel_id;

  return (
    <div className="flex items-center space-x-3">
      <img
        src={account.image || '/default-avatar.png'}
        alt={displayName}
        className="w-10 h-10 rounded-full"
      />
      <div>
        <p className="font-medium">{displayName}</p>
        {showChannelBadge && (
          <p className="text-sm text-gray-500">
            {isYouTubeCreator ? 'YouTube Creator' : 'Google Account'}
          </p>
        )}
```

```
      </div>
    </div>
  );
}
```

🔍 Key Changes Summary

1. **All `user` references → `account`**
   - `getUserByEmail` → `getAccountByEmail`
   - `createUser` → `createAccount`
   - `updateUser` → `updateAccount`
   - `/api/auth/user` → `/api/auth/account`
2. **Database queries use accounts table**
   - `storage.getUserByEmail()` → `storage.getAccountByEmail()`
   - `users.youtube_channel_title` → `accounts.youtube_channel_title`
3. **Frontend updates**
   - `useQuery` keys updated
   - Component props expect `account` not `user`
   - Display functions use account terminology
4. **OAuth tokens stored directly in accounts**
   - No separate OAuth table needed
   - Tokens are fields on the account record

✅ Testing the Implementation

bash

```
# Check your database structure
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'accounts';

# Verify YouTube channel fields exist
SELECT id, email, name, youtube_channel_title, youtube_channel_id
FROM accounts
LIMIT 5;
```

This implementation assumes your `accounts` table has all necessary fields including YouTube channel information. If you need any adjustments based on your specific schema, let me know!