Subscription Flow Diagnostic Report

Executive Summary

The session appears to **reset or become invalid** during subscription checkout because `supabase.auth.refreshSession()` on **line 183** of `BillingCheckout.tsx` can emit a `TOKEN_REFRESHED` auth event that races against the `onAuthStateChange` listener in `AuthProvider.tsx`. While `TOKEN_REFRESHED` is filtered out at line 258, the **real problem** is that `refreshSession()` itself can **fail silently or return a new session** that triggers Supabase's internal auth state change, potentially causing `loadUserData` to re-fire and briefly set `isLoading = true`, which makes `App.tsx` **render the loading screen** instead of the checkout page.

---

1. Frontend Flow

Step-by-Step Execution

```
sequenceDiagram
    participant U as User
    participant BC as BillingCheckout
    participant SA as supabase.auth
    participant EF as Edge Function
    participant S as Stripe

    U->>BC: Click "Suscribirse"
    BC->>BC: handleSubscribe(plan)
    BC->>BC: setLoadingPlanKey(plan.key)
    BC->>SA: refreshSession()
    SA-->>BC: { session } or { error }
    BC->>EF: functions.invoke('create-checkout-session')
    EF-->>BC: { url }
    BC->>S: window.location.href = url
    S-->>BC: Redirect to /platform/billing?success=true
```

Files Involved

File	Purpose
BillingCheckout.tsx	Subscription UI + click handler
AuthProvider.tsx	Auth context + `onAuthStateChange` listener
useAuth.ts	Thin wrapper exposing `AuthContext`
App.tsx	Routing + loading/auth guards
supabase.ts	Supabase client + impersonation header injection
subscription.config.ts	Plan ↔ Stripe Price ID mapping
create-checkout-session/index.ts	Edge function creating Stripe session
stripe-webhook/index.ts	Webhook processing subscription events

---

2. Authentication State

How Session is Obtained

1. **On mount**: `AuthProvider.tsx` calls `supabase.auth.getSession()` → `loadUserData(userId, true)` (AuthProvider.tsx:240)
2. **Ongoing**: `onAuthStateChange` listener handles `SIGNED_IN`, `INITIAL_SESSION`, `SIGNED_OUT` events (AuthProvider.tsx:250-281)
3. **Token storage**: Supabase JS SDK manages tokens in `localStorage` by default (no custom `persistSession` config in `supabase.ts`)
4. **Checkout flow**: `BillingCheckout.tsx` calls `supabase.auth.refreshSession()` **before** the edge function call to ensure a fresh token (BillingCheckout.tsx:183)

Auth Event Filtering in AuthProvider

```
TOKEN_REFRESHED → ignored (line 258) ✅
USER_UPDATED    → ignored (line 258) ✅
SIGNED_OUT      → resetState()
SIGNED_IN       → loadUserData() (with debounce guard)
INITIAL_SESSION → loadUserData() (with debounce guard)
```

---

3. Backend / Edge Function Flow

create-checkout-session/index.ts

1. **Auth validation** (line 38): Uses `supabase.auth.getUser(token)` with the `Authorization: Bearer` header sent from the frontend
2. **User lookup** (line 64): Fetches `is_super_admin` and `email` from `users` table
3. **Membership check** (line 81): Verifies `company_members` role ∈ `[admin, owner, manager]` (skipped for SuperAdmin)
4. **Stripe customer** (line 135-163): Get-or-create `stripe_customer_id` on the company
5. **Checkout session** (line 171-198): Creates a Stripe Checkout Session with `mode: 'subscription'`
6. **Returns** `{ url }` to the frontend

> *[!NOTE] The edge function uses `SUPABASE_SERVICE_ROLE_KEY` — so JWTs are validated server-side via `getUser()` against the raw access token. This is secure and correct.*

---

4. Payment Provider Integration

Stripe Configuration

Aspect	Value
Secret Key	`STRIPE_SECRET_KEY` env var (edge function)
Public Key	Not used — Stripe Checkout is redirect-based
API Version	`2024-06-20` (checkout) / `2022-11-15` (webhook)
Checkout Mode	`subscription`

Success URL	`${APP_URL}/platform/billing?success=true`
Cancel URL	`${APP_URL}/platform/billing?canceled=true`

Redirect Path

```
BillingCheckout → Stripe Checkout (external domain) → /platform/billing?success=true
```

> *[!WARNING] The **success redirect** goes to `/platform/billing` (the Billing page), but the user was on `/platform/billing/checkout` (the BillingCheckout page). The `useEffect` on line 133-150 of `BillingCheckout.tsx` **will never fire** on return from Stripe because the redirect targets a different route. The success/cancel banner only works if the URL matches the `/platform/billing/checkout` route.*

Webhook Handling

The stripe-webhook/index.ts handles:

- `customer.subscription.created/updated` → updates `subscription_status`, `stripe_subscription_id`, etc.
- `customer.subscription.deleted` → sets status to `canceled`
- `invoice.payment_succeeded` → sets status to `active`
- `invoice.payment_failed` → sets `past_due` + 7-day grace period

---

5. Session Break Points — Root Cause Analysis

🔴 PRIMARY CAUSE: `refreshSession()` Race Condition

Location: BillingCheckout.tsx:183

```
const { data: refreshData, error: refreshError } = await supabase.auth.refreshSession();
```

What happens internally:

1. `refreshSession()` calls the Supabase Auth API to get a new access + refresh token pair
2. The Supabase JS SDK **overwrites** the stored session in `localStorage`
3. This triggers an **internal `TOKEN_REFRESHED` event** on the `onAuthStateChange` listener

Why it's problematic:

- While `TOKEN_REFRESHED` is correctly filtered out at line 258 of AuthProvider, the `refreshSession()` call can **fail** if the refresh token has already been consumed (single-use refresh tokens in Supabase). In that case:
  - `refreshError` is set → the `throw` on line 187 fires
  - The **old session in `localStorage` has already been invalidated** by the refresh attempt
  - The user is now effectively logged out from Supabase's perspective
  - The `onAuthStateChange` listener may fire a `SIGNED_OUT` event → `resetState()` → **session lost**

🟠 SECONDARY CAUSE: `isLoading` Guard in `onAuthStateChange`

Location: AuthProvider.tsx:273

```
if (isLoading || (userIdRef.current && userIdRef.current === incomingUserId)) {
    log.info('SIGNED_IN/INITIAL_SESSION but user already loaded → SKIP reload');
    return;
}
```

This uses the **state variable** `isLoading` (not a ref), which means it captures the value at the time the listener was created. This is a **stale closure**. If `isLoading` changes after the listener was registered, the listener won't see the new value. This guard may not reliably prevent duplicate loads.

🟡 TERTIARY CAUSE: Hard Redirect Destroys React State

Location: BillingCheckout.tsx:208

```
window.location.href = data.url;
```

This is a **full page navigation** to Stripe's domain. When the user returns to `${APP_URL}/platform/billing?success=true`, the entire React app re-mounts:

1. `AuthProvider` runs `init()` → `getSession()` → `loadUserData()`
2. If the refresh token was consumed by the pre-checkout `refreshSession()` and Stripe's redirect took long enough for the new token to expire, `getSession()` could fail
3. Result: user lands on `/login`

🟢 MINOR: Impersonation Header Injection

Location: supabase.ts:20-38

The monkeypatched `supabase.functions.invoke()` calls `supabase.auth.getUser()` during impersonation. This is an extra network call that could fail or delay execution, but it's only active during impersonation mode and doesn't directly cause session loss for regular users.

Execution Flow Diagram

```
flowchart TD
    A["User clicks Suscribirse"] --> B["handleSubscribe(plan)"]
    B --> C{"plan.key === 'demo'?"}
    C -->|Yes| D["navigate('/dashboard')"]
    C -->|No| E["Validate plan key + priceId + companyId"]
    E --> F["setLoadingPlanKey(plan.key)"]
    F --> G["🔴 supabase.auth.refreshSession()"]

    G --> H{"refreshError?"}
    H -->|Yes| I["❌ throw 'Sesión expirada'<br>Old session INVALIDATED"]
    I --> J["Catch block → setError()"]
    H -->|No| K["✅ Get fresh access_token"]

    K --> L["supabase.functions.invoke<br>('create-checkout-session')"]
    L --> M{"Edge function error?"}
    M -->|Yes| N["setError(message)"]
    M -->|No| O["Receive { url }"]
    O --> P["🟡 window.location.href = url<br>FULL PAGE NAVIGATION"]

    P --> Q["Stripe Checkout (external)"]
    Q --> R["User completes payment"]
    R --> S["Redirect to /platform/billing?success=true"]
    S --> T["React App RE-MOUNTS"]
    T --> U["AuthProvider.init() → getSession()"]
    U --> V{"Session valid?"}
    V -->|Yes| W["loadUserData → render Billing page"]
    V -->|No| X["❌ resetState → redirect /login"]

    style G fill:#ff6b6b,color:#fff
    style I fill:#ff6b6b,color:#fff
    style P fill:#ffd93d,color:#000
    style X fill:#ff6b6b,color:#fff
```

Exact Probable Root Cause

> *[!CAUTION] **`supabase.auth.refreshSession()` at BillingCheckout.tsx:183** is the primary suspect.*
>
> *When the refresh token has already been rotated (e.g., by a concurrent tab, by the Supabase SDK's own auto-refresh, or by a race with the `onAuthStateChange` listener), calling `refreshSession()` manually:*
>
> 1. ***Consumes and invalidates the current refresh token** on the server*
> 2. ***Fails** to return a new session*
> 3. *Leaves the client with **no valid tokens in localStorage***
> 4. *The `onAuthStateChange` listener fires `SIGNED_OUT`, calling `resetState()` → user is logged out*

Recommended Minimal Fix

Replace the explicit `refreshSession()` with `getSession()`, which reads the current valid session **without forcing a token rotation**:

```diff
// BillingCheckout.tsx, lines 182-191

- // Refresh token forzado antes de llamar a la Edge Function
- const { data: refreshData, error: refreshError } = await supabase.auth.refreshSession();
-
- if (refreshError || !refreshData.session) {
-     console.error('[BillingCheckout] Failed to refresh session:', refreshError);
-     throw new Error('Sesión expirada. Por favor, inicia sesión de nuevo.');
- }
-
- const session = refreshData.session;
+ // Obtener sesión actual (sin forzar rotación de refresh token)
+ const { data: { session }, error: sessionError } = await supabase.auth.getSession();
+
+ if (sessionError || !session) {
+     console.error('[BillingCheckout] No active session:', sessionError);
+     throw new Error('Sesión expirada. Por favor, inicia sesión de nuevo.');
+ }
```

Why this works:

- `getSession()` returns the cached session from memory/localStorage **without** calling the auth server
- If the session is still within its JWT expiry window (~1 hour), the access token is perfectly valid for the edge function call
- The Supabase SDK's own `autoRefreshToken` mechanism (enabled by default) already handles background token refresh
- This avoids the race condition entirely because no new tokens are issued

> *[!TIP] If you want extra safety, you can add a fallback that only calls `refreshSession()` when `getSession()` returns no session:*
>
> ```
> let { data: { session } } = await supabase.auth.getSession();
> if (!session) {
>     const { data: refreshData } = await supabase.auth.refreshSession();
>     session = refreshData.session;
> }
> if (!session) throw new Error('Sesión expirada...');
> ```

Secondary Fix: `isLoading` Stale Closure

In AuthProvider.tsx:273, use `isFetchingRef.current` instead of the state variable `isLoading`:

```diff
- if (isLoading || (userIdRef.current && userIdRef.current === incomingUserId)) {
+ if (isFetchingRef.current || (userIdRef.current && userIdRef.current === incomingUserId))
{
```

This ensures the guard always reads the latest value, not a stale closure capture.

Verification Results

✅ Confirmed Fixes

#	Fix	File	Status
1	`refreshSession()` → `getSession()`	BillingCheckout.tsx:186	✅ Correctly applied
2	`isLoading` → `isFetchingRef.current`	AuthProvider.tsx:273	✅ Corrected (was misapplied)

⚠️ Issue Found & Fixed During Verification

The stale-closure fix was initially applied to the **wrong location** — line 291 (login redirect `useEffect`) instead of line 273 (inside `onAuthStateChange` listener). This introduced an **undefined variable** `incomingUserId` outside its scope. Corrected by:

- Moving `isFetchingRef.current` to line 273 (the actual bug location)
- Reverting line 291 to its original `!isLoading && user` logic

✅ End-to-End Flow Trace

```
Login → AuthProvider.init() → getSession() → loadUserData() → user in context
  ↓
BillingCheckout renders → useAuth() reads { user, currentCompany }
  ↓
Click "Suscribirse" → handleSubscribe(plan)
  ↓
getSession() → reads cached token (NO rotation) ✅
  ↓
functions.invoke('create-checkout-session', { Authorization: Bearer token })
  ↓
Edge Function validates token via getUser() → creates Stripe session
  ↓
window.location.href = stripe_url → full page navigation
  ↓
Stripe → redirect to /platform/billing?success=true → React re-mounts
  ↓
AuthProvider.init() → getSession() → session still valid ✅
```

🚀 Ready to Test: Yes

Both fixes are correctly in place. The subscription flow should no longer lose the session during checkout. Test by clicking "Suscribirse" on any paid plan and verifying you reach Stripe Checkout without being redirected to `/login`.