# API Architecture

## API Strategy: Database-First (PostgREST)

**No custom REST or GraphQL API**. Supabase auto-generates a RESTful API via PostgREST based on database schema. All API calls are database queries through the Supabase client.

### Supabase Client API Patterns

**Query Builder Syntax**:

// SELECT with joins  
const { data, error } = await supabase  
 .from('initiatives')  
 .select(`  
 \*,  
 strategic\_goal:strategic\_goals(\*),  
 department:departments(\*),  
 budgets:initiative\_budgets(\*),  
 kpis:initiative\_kpis(\*)  
 `)  
 .eq('status', 'in\_progress')  
 .order('rank\_within\_priority')

**Insert/Update/Delete**:

// INSERT  
const { data, error } = await supabase  
 .from('initiatives')  
 .insert({ name: 'New Initiative', ... })  
 .select()  
 .single()  
  
// UPDATE  
const { error } = await supabase  
 .from('initiatives')  
 .update({ status: 'completed' })  
 .eq('id', initiativeId)  
  
// DELETE (soft delete preferred)  
const { error } = await supabase  
 .from('strategic\_plans')  
 .update({ status: 'archived' })  
 .eq('id', planId)

**Real-time Subscriptions** (Phase 2):

const channel = supabase  
 .channel('plan-changes')  
 .on(  
 'postgres\_changes',  
 {  
 event: 'UPDATE',  
 schema: 'public',  
 table: 'strategic\_plans',  
 filter: `id=eq.${planId}`,  
 },  
 (payload) => {  
 console.log('Plan updated:', payload)  
 }  
 )  
 .subscribe()

## Server Actions as API Endpoints

For complex operations, use Next.js Server Actions instead of REST endpoints:

// app/actions/budgets.ts  
'use server'  
  
export async function validateBudget(initiativeId: string) {  
 const supabase = createServerSupabaseClient()  
  
 // Fetch initiative with budgets  
 const { data: initiative } = await supabase  
 .from('initiatives')  
 .select('\*, budgets:initiative\_budgets(\*)')  
 .eq('id', initiativeId)  
 .single()  
  
 // Business logic: validate funding sources sum to total  
 const totalBudget =  
 initiative.total\_year\_1\_cost +  
 initiative.total\_year\_2\_cost +  
 initiative.total\_year\_3\_cost  
  
 const totalFunding = initiative.financial\_analysis.funding\_sources  
 .reduce((sum, source) => sum + source.amount, 0)  
  
 if (Math.abs(totalBudget - totalFunding) > 0.01) {  
 return {  
 valid: false,  
 error: `Budget mismatch: Total=$${totalBudget}, Funding=$${totalFunding}`,  
 }  
 }  
  
 return { valid: true }  
}

## API Security

All API access is secured via Supabase RLS policies (see [Security Architecture](#security-architecture)).