# Strategic Planning System - Full-Stack Architecture

**Project Name:** Stratic Plan **Version:** 1.0 **Last Updated:** January 9, 2025 **Status:** Architecture Approved

## Table of Contents

1. [High-Level Architecture](#high-level-architecture)
2. [Technology Stack](#technology-stack)
3. [Data Architecture](#data-architecture)
4. [API Architecture](#api-architecture)
5. [Frontend Architecture](#frontend-architecture)
6. [Backend Architecture](#backend-architecture)
7. [Security Architecture](#security-architecture)
8. [Project Structure](#project-structure)
9. [Development Workflow](#development-workflow)
10. [Deployment Architecture](#deployment-architecture)
11. [Performance Strategy](#performance-strategy)
12. [Testing Strategy](#testing-strategy)
13. [Monitoring & Observability](#monitoring--observability)
14. [Coding Standards](#coding-standards)
15. [Error Handling](#error-handling)

## High-Level Architecture

### System Overview

The Strategic Planning System is a full-stack web application built with Next.js 14+ and Supabase, designed to digitize municipal government strategic planning processes. The system reduces plan creation time from 40-50 hours to <25 hours while enabling real-time collaboration, automated budget validation, and comprehensive reporting.

### Architecture Diagram

┌─────────────────────────────────────────────────────────────────┐  
│ CLIENT LAYER │  
│ Next.js 14+ App Router (React Server Components + Client) │  
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │  
│ │ Department │ │ City Manager │ │ Finance │ │  
│ │ Dashboard │ │ Dashboard │ │ Dashboard │ │  
│ └──────────────┘ └──────────────┘ └──────────────┘ │  
│ ┌─────────────────────────────────────────────────┐ │  
│ │ Shared Component Library (shadcn/ui) │ │  
│ └─────────────────────────────────────────────────┘ │  
└─────────────────────────────────────────────────────────────────┘  
 ▼  
┌─────────────────────────────────────────────────────────────────┐  
│ API/MIDDLEWARE LAYER │  
│ Next.js Server Actions + Route Handlers │  
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │  
│ │ Supabase │ │ Business │ │ PDF/Export │ │  
│ │ Client (SSR) │ │ Logic │ │ Services │ │  
│ └──────────────┘ └──────────────┘ └──────────────┘ │  
└─────────────────────────────────────────────────────────────────┘  
 ▼  
┌─────────────────────────────────────────────────────────────────┐  
│ BACKEND SERVICES │  
│ Supabase (BaaS Platform) │  
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │  
│ │ PostgreSQL │ │ Auth & RLS │ │ Storage │ │  
│ │ + pgvector │ │ Policies │ │ (Files) │ │  
│ └──────────────┘ └──────────────┘ └──────────────┘ │  
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │  
│ │ PostgREST │ │ Edge Funcs │ │ Realtime │ │  
│ │ (Auto API) │ │ (Optional) │ │ (Optional) │ │  
│ └──────────────┘ └──────────────┘ └──────────────┘ │  
└─────────────────────────────────────────────────────────────────┘  
 ▼  
┌─────────────────────────────────────────────────────────────────┐  
│ INFRASTRUCTURE LAYER │  
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │  
│ │ Vercel CDN │ │ Supabase │ │ GitHub │ │  
│ │ (Hosting) │ │ Cloud │ │ (VCS/CI) │ │  
│ └──────────────┘ └──────────────┘ └──────────────┘ │  
└─────────────────────────────────────────────────────────────────┘

### Architecture Principles

1. **Server-First Rendering**: Use React Server Components (RSC) for data fetching, Client Components only for interactivity
2. **Type Safety**: End-to-end TypeScript with Zod validation and generated Supabase types
3. **Security by Default**: Row-Level Security (RLS) enforced at database level, not application level
4. **Progressive Enhancement**: Core functionality works without JavaScript, enhanced with client-side features
5. **Hybrid Data Model**: Normalized relational tables + JSONB flexibility
6. **API-less Architecture**: Direct database access via Supabase client (RLS-protected), no custom REST/GraphQL layer

## Technology Stack

### Complete Tech Stack Table

| Layer | Technology | Version | Purpose | Justification |
| --- | --- | --- | --- | --- |
| **Frontend Framework** | Next.js | 14.x | Server/client rendering, routing | Industry standard, RSC support, Vercel integration |
| **UI Library** | React | 18.x | Component-based UI | Foundation for Next.js, hooks ecosystem |
| **Language** | TypeScript | 5.x | Type safety | Catch errors at compile time, better DX |
| **Styling** | TailwindCSS | 3.x | Utility-first CSS | Rapid development, consistent design system |
| **Component Library** | shadcn/ui | latest | Pre-built accessible components | Built on Radix UI, customizable, copy-paste approach |
| **Form Management** | React Hook Form | 7.x | Complex form handling | Performance, validation integration |
| **Validation** | Zod | 3.x | Schema validation | Type inference, runtime safety |
| **Data Visualization** | Recharts | 2.x | Charts and graphs | React-native, declarative API |
| **Database** | PostgreSQL | 15.x | Primary data store | JSONB support, vector search (pgvector) |
| **BaaS Platform** | Supabase | latest | Backend infrastructure | Auth, RLS, auto-generated API, migrations |
| **Auth** | Supabase Auth | built-in | User authentication | Email/password, OAuth, RLS integration |
| **ORM** | Supabase Client | latest | Database access | Type-safe queries, RLS enforcement |
| **PDF Generation** | react-pdf | 3.x | Export to PDF | City Council reports, plan exports |
| **State Management** | React Context | built-in | Global state (minimal) | User session, theme, avoid Zustand unless needed |
| **HTTP Client** | fetch (native) | built-in | API calls | Server Actions preferred, fallback to fetch |
| **Date Handling** | date-fns | 2.x | Date manipulation | Lighter than moment.js |
| **Markdown** | react-markdown | 8.x | Render markdown comments | Comments, rich text display |
| **Icons** | Lucide React | latest | Icon library | Tree-shakeable, modern design |
| **Hosting** | Vercel | latest | Next.js hosting | Optimized for Next.js, edge functions |
| **CI/CD** | GitHub Actions | built-in | Automated testing, deployment | Free for public repos, Vercel integration |
| **Monitoring** | Vercel Analytics | built-in | Performance monitoring | Built-in, zero-config |
| **Error Tracking** | Sentry (optional) | latest | Error monitoring | Production error tracking |

### Key Dependencies

{  
 "dependencies": {  
 "next": "^14.0.0",  
 "react": "^18.0.0",  
 "react-dom": "^18.0.0",  
 "typescript": "^5.0.0",  
 "@supabase/supabase-js": "^2.38.0",  
 "@supabase/ssr": "^0.0.10",  
 "tailwindcss": "^3.4.0",  
 "react-hook-form": "^7.49.0",  
 "zod": "^3.22.4",  
 "@hookform/resolvers": "^3.3.4",  
 "recharts": "^2.10.0",  
 "@react-pdf/renderer": "^3.1.14",  
 "date-fns": "^2.30.0",  
 "react-markdown": "^8.0.7",  
 "lucide-react": "^0.303.0",  
 "class-variance-authority": "^0.7.0",  
 "clsx": "^2.0.0",  
 "tailwind-merge": "^2.2.0"  
 },  
 "devDependencies": {  
 "@types/node": "^20.10.0",  
 "@types/react": "^18.2.0",  
 "@types/react-dom": "^18.2.0",  
 "eslint": "^8.56.0",  
 "eslint-config-next": "^14.0.0",  
 "prettier": "^3.1.0",  
 "prettier-plugin-tailwindcss": "^0.5.0",  
 "@playwright/test": "^1.40.0",  
 "vitest": "^1.1.0",  
 "@testing-library/react": "^14.1.0"  
 }  
}

## Data Architecture

### Database Schema Overview

**Philosophy:** Hybrid approach combining normalized relational tables with JSONB flexibility.

#### 15 Core Tables

1. **municipalities** - Multi-city support (single city for MVP)
2. **departments** - Organizational units (Water & Field Services, Parks & Rec, IT, etc.)
3. **fiscal\_years** - Reference table for fiscal year periods
4. **users** - User profiles and roles (extends Supabase auth.users)
5. **strategic\_plans** - The 3-year strategic planning document
6. **strategic\_goals** - 3-5 major goals per plan
7. **initiatives** - Individual strategic initiatives (heart of the plan)
8. **initiative\_budgets** - Normalized budget tracking for aggregation
9. **initiative\_kpis** - Performance metrics / Key Performance Indicators
10. **quarterly\_milestones** - Implementation timeline tracking
11. **initiative\_dependencies** - Dependencies between initiatives (many-to-many)
12. **initiative\_collaborators** - Cross-departmental initiatives (many-to-many)
13. **comments** - Collaborative feedback and discussion
14. **audit\_logs** - Track all changes for accountability
15. **document\_embeddings** - Vector embeddings for RAG/AI (Phase 2)

#### Entity Relationship Diagram

municipalities  
 ↓ (1:many)  
departments ──→ users (many:1)  
 ↓ (1:many)  
strategic\_plans ──→ fiscal\_years (3-year span)  
 ↓ (1:many)  
strategic\_goals  
 ↓ (1:many)  
initiatives ←──→ initiatives (dependencies, many:many)  
 ↓ ↓ (many:many via junction)  
 ↓ departments (collaborative initiatives)  
 ↓  
 ├─→ initiative\_budgets (normalized financial tracking)  
 ├─→ initiative\_kpis (performance metrics)  
 ├─→ quarterly\_milestones (implementation timeline)  
 ├─→ comments (feedback and discussion)  
 └─→ audit\_logs (change tracking)

### TypeScript Data Models

**Core Type Definitions** (types/database.ts):

// Auto-generated from Supabase CLI  
export type Database = {  
 public: {  
 Tables: {  
 strategic\_plans: {  
 Row: {  
 id: string  
 department\_id: string  
 start\_fiscal\_year\_id: string  
 end\_fiscal\_year\_id: string  
 title: string  
 status: 'draft' | 'under\_review' | 'approved' | 'active' | 'archived'  
 version: string  
 executive\_summary: string | null  
 department\_vision: string | null  
 swot\_analysis: Json | null  
 environmental\_scan: Json | null  
 benchmarking\_data: Json | null  
 total\_investment\_amount: number  
 approved\_by: string | null  
 approved\_at: string | null  
 published\_at: string | null  
 created\_by: string  
 created\_at: string  
 updated\_at: string  
 }  
 Insert: Omit<Row, 'id' | 'created\_at' | 'updated\_at'>  
 Update: Partial<Insert>  
 }  
 initiatives: {  
 Row: {  
 id: string  
 strategic\_goal\_id: string  
 lead\_department\_id: string  
 fiscal\_year\_id: string  
 initiative\_number: string  
 name: string  
 priority\_level: 'NEED' | 'WANT' | 'NICE\_TO\_HAVE'  
 rank\_within\_priority: number  
 description: string  
 rationale: string | null  
 expected\_outcomes: Json | null  
 status: 'not\_started' | 'in\_progress' | 'at\_risk' | 'completed' | 'deferred'  
 financial\_analysis: Json | null  
 roi\_analysis: Json | null  
 cost\_benefit\_analysis: Json | null  
 total\_year\_1\_cost: number  
 total\_year\_2\_cost: number  
 total\_year\_3\_cost: number  
 responsible\_party: string | null  
 created\_at: string  
 updated\_at: string  
 }  
 Insert: Omit<Row, 'id' | 'created\_at' | 'updated\_at'>  
 Update: Partial<Insert>  
 }  
 // ... other tables  
 }  
 Views: {  
 // Materialized views for dashboards  
 }  
 Functions: {  
 // RLS helper functions  
 }  
 }  
}

**Application-Level Types** (types/models.ts):

// Strategic Plan with relationships  
export interface StrategicPlanWithRelations {  
 id: string  
 department: Department  
 start\_fiscal\_year: FiscalYear  
 end\_fiscal\_year: FiscalYear  
 title: string  
 status: PlanStatus  
 goals: StrategicGoalWithInitiatives[]  
 total\_investment\_amount: number  
 created\_by: User  
 created\_at: string  
 updated\_at: string  
}  
  
// Initiative with full context  
export interface InitiativeWithContext {  
 id: string  
 strategic\_goal: StrategicGoal  
 lead\_department: Department  
 fiscal\_year: FiscalYear  
 initiative\_number: string  
 name: string  
 priority\_level: PriorityLevel  
 rank\_within\_priority: number  
 description: string  
 status: InitiativeStatus  
 financial\_analysis: FinancialAnalysis  
 roi\_analysis: ROIAnalysis  
 budgets: InitiativeBudget[]  
 kpis: InitiativeKPI[]  
 milestones: QuarterlyMilestone[]  
 collaborators: InitiativeCollaborator[]  
 comments: Comment[]  
}  
  
// JSONB Structures  
export interface FinancialAnalysis {  
 year\_1?: YearBudget  
 year\_2?: YearBudget  
 year\_3?: YearBudget  
 funding\_sources: FundingSource[]  
}  
  
export interface YearBudget {  
 personnel\_costs: number  
 equipment\_technology: number  
 professional\_services: number  
 training\_development: number  
 materials\_supplies: number  
 other\_costs: number  
 total: number  
}  
  
export interface FundingSource {  
 source: string // "General Fund", "EPA Grant", etc.  
 amount: number  
 status: 'secured' | 'requested' | 'pending' | 'projected'  
}  
  
export interface ROIAnalysis {  
 financial: {  
 projected\_annual\_savings: number  
 projected\_revenue\_generation: number  
 payback\_period\_months: number  
 three\_year\_net\_impact: number  
 }  
 non\_financial: {  
 service\_quality\_improvement?: string  
 efficiency\_gains?: string  
 risk\_reduction?: string  
 citizen\_satisfaction?: string  
 employee\_impact?: string  
 }  
}

### Data Access Patterns

**Server Component Data Fetching**:

// app/dashboard/page.tsx (Server Component)  
import { createServerSupabaseClient } from '@/lib/supabase/server'  
  
export default async function DashboardPage() {  
 const supabase = createServerSupabaseClient()  
  
 // RLS automatically filters by user's department  
 const { data: plans } = await supabase  
 .from('strategic\_plans')  
 .select(`  
 \*,  
 department:departments(\*),  
 start\_fiscal\_year:fiscal\_years!start\_fiscal\_year\_id(\*),  
 goals:strategic\_goals(  
 \*,  
 initiatives(\*)  
 )  
 `)  
 .eq('status', 'active')  
 .order('created\_at', { ascending: false })  
  
 return <DashboardView plans={plans} />  
}

**Client Component Data Fetching** (rare, prefer Server Components):

// components/InitiativeForm.tsx (Client Component)  
'use client'  
  
import { createBrowserSupabaseClient } from '@/lib/supabase/client'  
import { useState, useEffect } from 'react'  
  
export function InitiativeForm() {  
 const supabase = createBrowserSupabaseClient()  
 const [goals, setGoals] = useState([])  
  
 useEffect(() => {  
 supabase  
 .from('strategic\_goals')  
 .select('\*')  
 .then(({ data }) => setGoals(data || []))  
 }, [])  
  
 return <form>...</form>  
}

**Server Actions for Mutations**:

// app/actions/initiatives.ts  
'use server'  
  
import { createServerSupabaseClient } from '@/lib/supabase/server'  
import { revalidatePath } from 'next/cache'  
import { z } from 'zod'  
  
const initiativeSchema = z.object({  
 strategic\_goal\_id: z.string().uuid(),  
 name: z.string().min(5),  
 priority\_level: z.enum(['NEED', 'WANT', 'NICE\_TO\_HAVE']),  
 description: z.string().min(20),  
 // ... other fields  
})  
  
export async function createInitiative(formData: FormData) {  
 const supabase = createServerSupabaseClient()  
  
 // Parse and validate  
 const parsed = initiativeSchema.safeParse(Object.fromEntries(formData))  
 if (!parsed.success) {  
 return { error: parsed.error.flatten() }  
 }  
  
 // Insert (RLS enforces department access)  
 const { data, error } = await supabase  
 .from('initiatives')  
 .insert(parsed.data)  
 .select()  
 .single()  
  
 if (error) {  
 return { error: error.message }  
 }  
  
 // Revalidate cache  
 revalidatePath('/dashboard')  
 revalidatePath(`/initiatives/${data.id}`)  
  
 return { data }  
}

## 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)).

## Frontend Architecture

### Next.js App Router Structure

app/  
├── (auth)/ # Auth group (different layout)  
│ ├── login/  
│ └── signup/  
├── (dashboard)/ # Dashboard group (authenticated)  
│ ├── dashboard/  
│ ├── plans/  
│ │ ├── page.tsx # List all plans  
│ │ ├── [id]/  
│ │ │ ├── page.tsx # Plan detail (Server Component)  
│ │ │ ├── edit/  
│ │ │ │ └── page.tsx # Plan edit form  
│ │ │ └── goals/  
│ │ │ └── [goalId]/  
│ │ │ └── initiatives/  
│ │ │ └── [initiativeId]/  
│ │ │ └── page.tsx  
│ │ └── new/  
│ │ └── page.tsx # Create new plan  
│ ├── initiatives/  
│ ├── budgets/  
│ ├── reports/  
│ └── settings/  
├── admin/ # Admin pages (separate layout)  
│ ├── users/  
│ ├── departments/  
│ └── audit-logs/  
├── api/ # Route handlers (minimal use)  
│ └── export-pdf/  
│ └── route.ts  
├── actions/ # Server Actions (mutations)  
│ ├── plans.ts  
│ ├── initiatives.ts  
│ └── budgets.ts  
├── layout.tsx # Root layout  
└── page.tsx # Home page

### Component Architecture

**Atomic Design Pattern**:

components/  
├── ui/ # shadcn/ui primitives  
│ ├── button.tsx  
│ ├── input.tsx  
│ ├── dialog.tsx  
│ └── ...  
├── atoms/ # Basic building blocks  
│ ├── StatusBadge.tsx  
│ ├── PriorityIcon.tsx  
│ └── CurrencyDisplay.tsx  
├── molecules/ # Composed components  
│ ├── InitiativeCard.tsx  
│ ├── BudgetSummary.tsx  
│ └── CommentThread.tsx  
├── organisms/ # Complex features  
│ ├── InitiativeForm.tsx  
│ ├── BudgetTable.tsx  
│ ├── DashboardCharts.tsx  
│ └── PlanHeader.tsx  
├── templates/ # Page layouts  
│ ├── DashboardLayout.tsx  
│ ├── PlanLayout.tsx  
│ └── ReportLayout.tsx  
└── providers/ # Context providers  
 ├── AuthProvider.tsx  
 ├── ThemeProvider.tsx  
 └── SupabaseProvider.tsx

### Server Components vs. Client Components

**Default to Server Components** (fetch data, render static content):

// components/PlanList.tsx (Server Component)  
import { createServerSupabaseClient } from '@/lib/supabase/server'  
  
export async function PlanList() {  
 const supabase = createServerSupabaseClient()  
 const { data: plans } = await supabase  
 .from('strategic\_plans')  
 .select('\*')  
  
 return (  
 <div>  
 {plans.map(plan => (  
 <PlanCard key={plan.id} plan={plan} />  
 ))}  
 </div>  
 )  
}

**Use Client Components** only for interactivity:

// components/InitiativeForm.tsx (Client Component)  
'use client'  
  
import { useForm } from 'react-hook-form'  
import { zodResolver } from '@hookform/resolvers/zod'  
import { createInitiative } from '@/app/actions/initiatives'  
  
export function InitiativeForm() {  
 const form = useForm({  
 resolver: zodResolver(initiativeSchema),  
 })  
  
 async function onSubmit(data) {  
 const result = await createInitiative(data)  
 if (result.error) {  
 form.setError('root', { message: result.error })  
 } else {  
 // Success feedback  
 }  
 }  
  
 return <form onSubmit={form.handleSubmit(onSubmit)}>...</form>  
}

### State Management Strategy

**Minimize client-side state**. Prefer:

1. **Server state** (database, Server Components)
2. **URL state** (searchParams, route params)
3. **Form state** (React Hook Form)
4. **Local state** (useState for UI-only state)

**Only use Context** for truly global client state:

// app/providers/AuthProvider.tsx  
'use client'  
  
import { createContext, useContext } from 'react'  
import { User } from '@supabase/supabase-js'  
  
const AuthContext = createContext<{ user: User | null }>({ user: null })  
  
export function AuthProvider({  
 children,  
 user  
}: {  
 children: React.ReactNode  
 user: User | null  
}) {  
 return (  
 <AuthContext.Provider value={{ user }}>  
 {children}  
 </AuthContext.Provider>  
 )  
}  
  
export const useAuth = () => useContext(AuthContext)

### Form Handling Pattern

**Standard Form with React Hook Form + Zod**:

'use client'  
  
import { useForm } from 'react-hook-form'  
import { zodResolver } from '@hookform/resolvers/zod'  
import { z } from 'zod'  
import { createInitiative } from '@/app/actions/initiatives'  
import { Input } from '@/components/ui/input'  
import { Button } from '@/components/ui/button'  
  
const initiativeSchema = z.object({  
 name: z.string().min(5, 'Name must be at least 5 characters'),  
 priority\_level: z.enum(['NEED', 'WANT', 'NICE\_TO\_HAVE']),  
 description: z.string().min(20),  
})  
  
type InitiativeFormData = z.infer<typeof initiativeSchema>  
  
export function InitiativeForm({ goalId }: { goalId: string }) {  
 const form = useForm<InitiativeFormData>({  
 resolver: zodResolver(initiativeSchema),  
 defaultValues: {  
 name: '',  
 priority\_level: 'NEED',  
 description: '',  
 },  
 })  
  
 async function onSubmit(data: InitiativeFormData) {  
 const result = await createInitiative({ ...data, strategic\_goal\_id: goalId })  
 if (result.error) {  
 form.setError('root', { message: result.error })  
 }  
 }  
  
 return (  
 <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">  
 <Input  
 {...form.register('name')}  
 placeholder="Initiative name"  
 />  
 {form.formState.errors.name && (  
 <p className="text-sm text-red-600">{form.formState.errors.name.message}</p>  
 )}  
  
 <select {...form.register('priority\_level')}>  
 <option value="NEED">NEED</option>  
 <option value="WANT">WANT</option>  
 <option value="NICE\_TO\_HAVE">NICE TO HAVE</option>  
 </select>  
  
 <Button type="submit" disabled={form.formState.isSubmitting}>  
 {form.formState.isSubmitting ? 'Creating...' : 'Create Initiative'}  
 </Button>  
 </form>  
 )  
}

## Backend Architecture

### Supabase RLS (Row-Level Security)

**Core security layer**. All data access is controlled at the database level via RLS policies.

#### RLS Helper Functions

-- Get current user's municipality  
CREATE OR REPLACE FUNCTION auth.user\_municipality\_id()  
RETURNS uuid  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT municipality\_id FROM public.users WHERE id = auth.uid();  
$$;  
  
-- Get current user's department  
CREATE OR REPLACE FUNCTION auth.user\_department\_id()  
RETURNS uuid  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT department\_id FROM public.users WHERE id = auth.uid();  
$$;  
  
-- Get current user's role  
CREATE OR REPLACE FUNCTION auth.user\_role()  
RETURNS text  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT role FROM public.users WHERE id = auth.uid();  
$$;  
  
-- Check if user is admin or city manager  
CREATE OR REPLACE FUNCTION auth.is\_admin\_or\_manager()  
RETURNS boolean  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT role IN ('admin', 'city\_manager') FROM public.users WHERE id = auth.uid();  
$$;

#### RLS Policy Examples

**Strategic Plans - Department Scoped**:

-- Department users can view their own plans  
CREATE POLICY "Users can view own department plans"  
ON strategic\_plans  
FOR SELECT  
USING (  
 department\_id = auth.user\_department\_id()  
 OR auth.is\_admin\_or\_manager()  
);  
  
-- Department directors can create plans  
CREATE POLICY "Directors can create plans"  
ON strategic\_plans  
FOR INSERT  
WITH CHECK (  
 department\_id = auth.user\_department\_id()  
 AND auth.user\_role() IN ('admin', 'department\_director')  
);  
  
-- Department directors can update their drafts  
CREATE POLICY "Directors can update own drafts"  
ON strategic\_plans  
FOR UPDATE  
USING (  
 department\_id = auth.user\_department\_id()  
 AND status IN ('draft', 'under\_review')  
 AND auth.user\_role() IN ('admin', 'department\_director')  
);  
  
-- City Manager can approve plans  
CREATE POLICY "City Manager can approve plans"  
ON strategic\_plans  
FOR UPDATE  
USING (auth.user\_role() = 'city\_manager')  
WITH CHECK (  
 status IN ('approved', 'active')  
 AND auth.user\_role() = 'city\_manager'  
);

**Initiatives - Cross-Department Collaboration**:

-- Users can view initiatives from their department or collaborating departments  
CREATE POLICY "Users can view accessible initiatives"  
ON initiatives  
FOR SELECT  
USING (  
 lead\_department\_id = auth.user\_department\_id()  
 OR EXISTS (  
 SELECT 1 FROM initiative\_collaborators  
 WHERE initiative\_id = initiatives.id  
 AND department\_id = auth.user\_department\_id()  
 )  
 OR auth.is\_admin\_or\_manager()  
);

**Comments - Universal Access**:

-- Any authenticated user can view comments on entities they can access  
CREATE POLICY "Users can view comments"  
ON comments  
FOR SELECT  
USING (  
 CASE entity\_type  
 WHEN 'strategic\_plan' THEN EXISTS (  
 SELECT 1 FROM strategic\_plans  
 WHERE id = comments.entity\_id  
 AND (department\_id = auth.user\_department\_id() OR auth.is\_admin\_or\_manager())  
 )  
 WHEN 'initiative' THEN EXISTS (  
 SELECT 1 FROM initiatives  
 WHERE id = comments.entity\_id  
 AND (lead\_department\_id = auth.user\_department\_id() OR auth.is\_admin\_or\_manager())  
 )  
 ELSE false  
 END  
);  
  
-- Users can create comments  
CREATE POLICY "Authenticated users can create comments"  
ON comments  
FOR INSERT  
WITH CHECK (auth.uid() IS NOT NULL);

### Database Triggers

**Audit Log Trigger**:

CREATE OR REPLACE FUNCTION audit\_trigger\_function()  
RETURNS TRIGGER  
LANGUAGE plpgsql  
AS $$  
BEGIN  
 IF (TG\_OP = 'DELETE') THEN  
 INSERT INTO audit\_logs (  
 table\_name,  
 record\_id,  
 action,  
 old\_values,  
 changed\_by,  
 changed\_at  
 ) VALUES (  
 TG\_TABLE\_NAME,  
 OLD.id,  
 'delete',  
 to\_jsonb(OLD),  
 auth.uid(),  
 now()  
 );  
 RETURN OLD;  
 ELSIF (TG\_OP = 'UPDATE') THEN  
 INSERT INTO audit\_logs (  
 table\_name,  
 record\_id,  
 action,  
 old\_values,  
 new\_values,  
 changed\_by,  
 changed\_at  
 ) VALUES (  
 TG\_TABLE\_NAME,  
 NEW.id,  
 'update',  
 to\_jsonb(OLD),  
 to\_jsonb(NEW),  
 auth.uid(),  
 now()  
 );  
 RETURN NEW;  
 ELSIF (TG\_OP = 'INSERT') THEN  
 INSERT INTO audit\_logs (  
 table\_name,  
 record\_id,  
 action,  
 new\_values,  
 changed\_by,  
 changed\_at  
 ) VALUES (  
 TG\_TABLE\_NAME,  
 NEW.id,  
 'insert',  
 to\_jsonb(NEW),  
 auth.uid(),  
 now()  
 );  
 RETURN NEW;  
 END IF;  
END;  
$$;  
  
-- Apply to strategic\_plans  
CREATE TRIGGER strategic\_plans\_audit  
AFTER INSERT OR UPDATE OR DELETE ON strategic\_plans  
FOR EACH ROW EXECUTE FUNCTION audit\_trigger\_function();  
  
-- Apply to initiatives  
CREATE TRIGGER initiatives\_audit  
AFTER INSERT OR UPDATE OR DELETE ON initiatives  
FOR EACH ROW EXECUTE FUNCTION audit\_trigger\_function();

### Database Indexes

**Performance-critical indexes**:

-- Strategic Plans  
CREATE INDEX idx\_strategic\_plans\_department\_status  
ON strategic\_plans(department\_id, status);  
  
CREATE INDEX idx\_strategic\_plans\_fiscal\_year  
ON strategic\_plans(start\_fiscal\_year\_id);  
  
-- Initiatives  
CREATE INDEX idx\_initiatives\_goal  
ON initiatives(strategic\_goal\_id);  
  
CREATE INDEX idx\_initiatives\_priority\_rank  
ON initiatives(priority\_level, rank\_within\_priority);  
  
CREATE INDEX idx\_initiatives\_status  
ON initiatives(status);  
  
-- Initiative Budgets  
CREATE INDEX idx\_initiative\_budgets\_fiscal\_year  
ON initiative\_budgets(fiscal\_year\_id, funding\_source);  
  
-- Comments  
CREATE INDEX idx\_comments\_entity  
ON comments(entity\_type, entity\_id);  
  
-- Audit Logs  
CREATE INDEX idx\_audit\_logs\_table\_record  
ON audit\_logs(table\_name, record\_id);  
  
CREATE INDEX idx\_audit\_logs\_changed\_at  
ON audit\_logs(changed\_at DESC);

## Security Architecture

### Authentication Flow

**Supabase Auth with Email/Password** (MVP):

// lib/supabase/auth.ts  
import { createBrowserSupabaseClient } from './client'  
  
export async function signUp(email: string, password: string) {  
 const supabase = createBrowserSupabaseClient()  
 const { data, error } = await supabase.auth.signUp({  
 email,  
 password,  
 options: {  
 emailRedirectTo: `${location.origin}/auth/callback`,  
 },  
 })  
 return { data, error }  
}  
  
export async function signIn(email: string, password: string) {  
 const supabase = createBrowserSupabaseClient()  
 const { data, error } = await supabase.auth.signInWithPassword({  
 email,  
 password,  
 })  
 return { data, error }  
}  
  
export async function signOut() {  
 const supabase = createBrowserSupabaseClient()  
 const { error } = await supabase.auth.signOut()  
 return { error }  
}

**Protected Route Middleware**:

// middleware.ts  
import { createMiddlewareSupabaseClient } from '@/lib/supabase/middleware'  
import { NextResponse } from 'next/server'  
import type { NextRequest } from 'next/server'  
  
export async function middleware(request: NextRequest) {  
 const response = NextResponse.next()  
 const supabase = createMiddlewareSupabaseClient(request, response)  
  
 const { data: { session } } = await supabase.auth.getSession()  
  
 // Redirect to login if not authenticated  
 if (!session && !request.nextUrl.pathname.startsWith('/login')) {  
 return NextResponse.redirect(new URL('/login', request.url))  
 }  
  
 // Redirect to dashboard if authenticated and trying to access login  
 if (session && request.nextUrl.pathname.startsWith('/login')) {  
 return NextResponse.redirect(new URL('/dashboard', request.url))  
 }  
  
 return response  
}  
  
export const config = {  
 matcher: [  
 '/dashboard/:path\*',  
 '/plans/:path\*',  
 '/initiatives/:path\*',  
 '/admin/:path\*',  
 ],  
}

### Authorization (Role-Based Access Control)

**7 User Roles**:

1. **admin** - Full access to all data, user management
2. **department\_director** - Create/edit plans for their department
3. **staff** - Edit plans if granted access by director
4. **city\_manager** - View all plans, approve plans, city-wide dashboards
5. **finance** - View all budgets, comment on budgets
6. **council** - View approved/active plans (read-only)
7. **public** - View published plans only

**Role Check Utility**:

// lib/auth/roles.ts  
import { Database } from '@/types/database'  
  
type UserRole = Database['public']['Tables']['users']['Row']['role']  
  
export function canEditPlan(  
 userRole: UserRole,  
 userDepartmentId: string,  
 planDepartmentId: string,  
 planStatus: string  
): boolean {  
 if (userRole === 'admin') return true  
 if (userRole === 'city\_manager') return false  
 if (userDepartmentId !== planDepartmentId) return false  
 if (planStatus === 'approved' || planStatus === 'active') return false  
 return ['department\_director', 'staff'].includes(userRole)  
}  
  
export function canApprovePlan(userRole: UserRole): boolean {  
 return ['admin', 'city\_manager'].includes(userRole)  
}  
  
export function canViewAllDepartments(userRole: UserRole): boolean {  
 return ['admin', 'city\_manager', 'finance'].includes(userRole)  
}

### Data Validation

**Server-Side Validation with Zod**:

// lib/validation/initiative.ts  
import { z } from 'zod'  
  
export const initiativeSchema = z.object({  
 strategic\_goal\_id: z.string().uuid('Invalid goal ID'),  
 name: z.string()  
 .min(5, 'Name must be at least 5 characters')  
 .max(200, 'Name too long'),  
 priority\_level: z.enum(['NEED', 'WANT', 'NICE\_TO\_HAVE']),  
 rank\_within\_priority: z.number().int().positive(),  
 description: z.string().min(20, 'Description too short'),  
 financial\_analysis: z.object({  
 year\_1: z.object({  
 personnel\_costs: z.number().nonnegative(),  
 equipment\_technology: z.number().nonnegative(),  
 professional\_services: z.number().nonnegative(),  
 training\_development: z.number().nonnegative(),  
 materials\_supplies: z.number().nonnegative(),  
 other\_costs: z.number().nonnegative(),  
 total: z.number().nonnegative(),  
 }).refine(  
 (data) => {  
 const sum = data.personnel\_costs +  
 data.equipment\_technology +  
 data.professional\_services +  
 data.training\_development +  
 data.materials\_supplies +  
 data.other\_costs  
 return Math.abs(sum - data.total) < 0.01  
 },  
 { message: 'Budget categories must sum to total' }  
 ),  
 funding\_sources: z.array(  
 z.object({  
 source: z.string(),  
 amount: z.number().positive(),  
 status: z.enum(['secured', 'requested', 'pending', 'projected']),  
 })  
 ).min(1, 'At least one funding source required'),  
 }),  
})  
  
export type InitiativeInput = z.infer<typeof initiativeSchema>

### Security Best Practices

1. **Never expose database credentials** - Use environment variables
2. **RLS is mandatory** - Enable RLS on all tables
3. **Validate all inputs** - Use Zod schemas server-side
4. **Sanitize JSONB data** - Prevent JSON injection
5. **Audit all changes** - Use audit\_logs table
6. **Use HTTPS only** - Enforce in production
7. **Rate limiting** - Supabase provides built-in rate limiting
8. **Content Security Policy** - Configure in next.config.js

## Project Structure

### Complete Directory Structure

strategic-planning-system/  
├── .github/  
│ └── workflows/  
│ ├── ci.yml # Run tests on PR  
│ └── deploy.yml # Deploy to Vercel  
├── app/ # Next.js App Router  
│ ├── (auth)/  
│ │ ├── login/  
│ │ │ └── page.tsx  
│ │ ├── signup/  
│ │ │ └── page.tsx  
│ │ └── layout.tsx  
│ ├── (dashboard)/  
│ │ ├── dashboard/  
│ │ │ ├── page.tsx # Main dashboard  
│ │ │ └── loading.tsx  
│ │ ├── plans/  
│ │ │ ├── page.tsx # List plans  
│ │ │ ├── [id]/  
│ │ │ │ ├── page.tsx # Plan detail  
│ │ │ │ ├── edit/  
│ │ │ │ │ └── page.tsx  
│ │ │ │ └── goals/  
│ │ │ │ └── [goalId]/  
│ │ │ │ └── initiatives/  
│ │ │ │ └── [initiativeId]/  
│ │ │ │ └── page.tsx  
│ │ │ └── new/  
│ │ │ └── page.tsx  
│ │ ├── initiatives/  
│ │ ├── budgets/  
│ │ ├── reports/  
│ │ └── layout.tsx  
│ ├── admin/  
│ │ ├── users/  
│ │ ├── departments/  
│ │ ├── audit-logs/  
│ │ └── layout.tsx  
│ ├── api/  
│ │ └── export-pdf/  
│ │ └── route.ts  
│ ├── actions/ # Server Actions  
│ │ ├── plans.ts  
│ │ ├── initiatives.ts  
│ │ └── budgets.ts  
│ ├── layout.tsx # Root layout  
│ ├── page.tsx # Home page  
│ └── globals.css # Global styles  
├── components/  
│ ├── ui/ # shadcn/ui components  
│ ├── atoms/  
│ ├── molecules/  
│ ├── organisms/  
│ ├── templates/  
│ └── providers/  
├── lib/  
│ ├── supabase/  
│ │ ├── client.ts # Browser client  
│ │ ├── server.ts # Server client  
│ │ ├── middleware.ts # Middleware client  
│ │ └── auth.ts # Auth helpers  
│ ├── auth/  
│ │ └── roles.ts # Role utilities  
│ ├── validation/  
│ │ ├── plan.ts  
│ │ ├── initiative.ts  
│ │ └── budget.ts  
│ ├── utils/  
│ │ ├── cn.ts # Class merger  
│ │ ├── currency.ts  
│ │ └── date.ts  
│ └── pdf/  
│ └── generator.ts # PDF export logic  
├── types/  
│ ├── database.ts # Auto-generated Supabase types  
│ └── models.ts # Application types  
├── supabase/  
│ ├── migrations/ # Database migrations  
│ │ ├── 20250109000001\_create\_core\_tables.sql  
│ │ ├── 20250109000002\_create\_planning\_tables.sql  
│ │ ├── 20250109000003\_create\_supporting\_tables.sql  
│ │ ├── 20250109000004\_create\_system\_tables.sql  
│ │ ├── 20250109000005\_enable\_rls\_policies.sql  
│ │ └── 20250109000006\_seed\_data.sql  
│ ├── config.toml # Supabase config  
│ └── seed.sql # Test data  
├── docs/ # Documentation  
│ ├── prd/ # Product Requirements  
│ ├── epics/ # User stories  
│ ├── architecture.md # This file  
│ └── database-schema-overview.md  
├── tests/  
│ ├── unit/  
│ ├── integration/  
│ └── e2e/  
├── public/  
│ ├── images/  
│ └── fonts/  
├── .env.local # Local environment variables  
├── .env.example # Example env file  
├── next.config.js # Next.js config  
├── tailwind.config.ts # Tailwind config  
├── tsconfig.json # TypeScript config  
├── package.json  
└── README.md

### Environment Variables

**.env.local**:

# Supabase  
NEXT\_PUBLIC\_SUPABASE\_URL=http://localhost:54321  
NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your-anon-key  
SUPABASE\_SERVICE\_ROLE\_KEY=your-service-role-key  
  
# App  
NEXT\_PUBLIC\_APP\_URL=http://localhost:3000  
  
# Optional: Third-party services  
SENTRY\_DSN=  
VERCEL\_ANALYTICS\_ID=

**.env.production** (Vercel):

NEXT\_PUBLIC\_SUPABASE\_URL=https://your-project.supabase.co  
NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your-prod-anon-key  
SUPABASE\_SERVICE\_ROLE\_KEY=your-prod-service-role-key  
NEXT\_PUBLIC\_APP\_URL=https://strategic-planning.yourdomain.com

## Development Workflow

### Local Development Setup

# 1. Clone repository  
git clone https://github.com/your-org/strategic-planning-system.git  
cd strategic-planning-system  
  
# 2. Install dependencies  
npm install  
  
# 3. Start Supabase locally  
npx supabase start  
  
# 4. Copy environment variables  
cp .env.example .env.local  
# Edit .env.local with Supabase credentials from `supabase status`  
  
# 5. Run database migrations  
npx supabase db push  
  
# 6. Generate TypeScript types  
npx supabase gen types typescript --local > types/database.ts  
  
# 7. Start development server  
npm run dev

### Git Workflow

**Branch Strategy**:

* main - Production-ready code
* develop - Integration branch
* feature/epic-1-story-1.1 - Feature branches

**Commit Convention**:

type(scope): subject  
  
[optional body]  
  
[optional footer]

Examples: - feat(initiatives): add create initiative form - fix(budgets): correct funding source validation - docs(readme): update setup instructions

### Code Review Checklist

* TypeScript compiles without errors
* All tests pass
* RLS policies tested for role
* Zod schema validation present
* No console.logs left in code
* Accessibility tested (keyboard navigation)
* Mobile responsive
* Error handling implemented

## Deployment Architecture

### Infrastructure Overview

┌─────────────────────────────────────────────────────────┐  
│ Vercel Edge Network │  
│ ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐ │  
│ │ US-East │ │ US-West │ │ Europe │ │ Asia │ │  
│ └─────────┘ └─────────┘ └─────────┘ └─────────┘ │  
│ Global CDN (Cached Assets) │  
└─────────────────────────────────────────────────────────┘  
 ▼  
┌─────────────────────────────────────────────────────────┐  
│ Next.js Application (Vercel) │  
│ ┌──────────────────────────────────────────────────┐ │  
│ │ Server Components (SSR) │ │  
│ │ - Render on-demand │ │  
│ │ - Direct Supabase access │ │  
│ └──────────────────────────────────────────────────┘ │  
│ ┌──────────────────────────────────────────────────┐ │  
│ │ Client Components (Browser) │ │  
│ │ - Minimal JavaScript │ │  
│ │ - Supabase client for real-time │ │  
│ └──────────────────────────────────────────────────┘ │  
└─────────────────────────────────────────────────────────┘  
 ▼  
┌─────────────────────────────────────────────────────────┐  
│ Supabase Cloud │  
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │  
│ │ PostgreSQL │ │ Auth & RLS │ │ Storage │ │  
│ │ (Primary) │ │ │ │ (Files) │ │  
│ └──────────────┘ └──────────────┘ └──────────────┘ │  
│ ┌──────────────┐ ┌──────────────┐ │  
│ │ PostgREST │ │ Realtime │ │  
│ │ (API) │ │ (WebSocket) │ │  
│ └──────────────┘ └──────────────┘ │  
└─────────────────────────────────────────────────────────┘

### Deployment Process

**Automatic Deployment via GitHub Actions**:

# .github/workflows/deploy.yml  
name: Deploy to Vercel  
  
on:  
 push:  
 branches: [main]  
 pull\_request:  
 branches: [main]  
  
jobs:  
 deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: '18'  
  
 - name: Install dependencies  
 run: npm ci  
  
 - name: Run tests  
 run: npm test  
  
 - name: Build  
 run: npm run build  
 env:  
 NEXT\_PUBLIC\_SUPABASE\_URL: ${{ secrets.SUPABASE\_URL }}  
 NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY: ${{ secrets.SUPABASE\_ANON\_KEY }}  
  
 - name: Deploy to Vercel  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID }}  
 vercel-args: '--prod'

**Database Migrations**:

# Run migrations on Supabase cloud  
npx supabase db push --db-url $DATABASE\_URL  
  
# Or via Supabase CLI (connected to project)  
npx supabase link --project-ref your-project-ref  
npx supabase db push

### Environment-Specific Configuration

**next.config.js**:

/\*\* @type {import('next').NextConfig} \*/  
const nextConfig = {  
 images: {  
 domains: ['your-project.supabase.co'],  
 },  
 async redirects() {  
 return [  
 {  
 source: '/',  
 destination: '/dashboard',  
 permanent: false,  
 has: [{ type: 'cookie', key: 'sb-access-token' }],  
 },  
 ]  
 },  
 async headers() {  
 return [  
 {  
 source: '/:path\*',  
 headers: [  
 {  
 key: 'X-Frame-Options',  
 value: 'DENY',  
 },  
 {  
 key: 'X-Content-Type-Options',  
 value: 'nosniff',  
 },  
 {  
 key: 'Referrer-Policy',  
 value: 'strict-origin-when-cross-origin',  
 },  
 ],  
 },  
 ]  
 },  
}  
  
module.exports = nextConfig

## Performance Strategy

### Performance Targets

* **First Contentful Paint (FCP)**: < 1.5s
* **Largest Contentful Paint (LCP)**: < 2.5s
* **Time to Interactive (TTI)**: < 3.5s
* **Total Blocking Time (TBT)**: < 300ms
* **Cumulative Layout Shift (CLS)**: < 0.1

### Optimization Techniques

**1. Server Components by Default**:

// ✅ Good: Server Component (no client JS)  
export default async function PlanPage({ params }) {  
 const plan = await fetchPlan(params.id) // Server-side  
 return <PlanView plan={plan} />  
}  
  
// ❌ Bad: Client Component when not needed  
'use client'  
export default function PlanPage({ params }) {  
 const [plan, setPlan] = useState(null)  
 useEffect(() => {  
 fetchPlan(params.id).then(setPlan)  
 }, [params.id])  
 return <PlanView plan={plan} />  
}

**2. Streaming with Suspense**:

import { Suspense } from 'react'  
  
export default function DashboardPage() {  
 return (  
 <div>  
 <Suspense fallback={<ChartsSkeleton />}>  
 <DashboardCharts /> {/\* Server Component, async data fetch \*/}  
 </Suspense>  
  
 <Suspense fallback={<TableSkeleton />}>  
 <InitiativeTable />  
 </Suspense>  
 </div>  
 )  
}

**3. Database Query Optimization**:

// ✅ Good: Select only needed columns  
const { data } = await supabase  
 .from('initiatives')  
 .select('id, name, status, total\_year\_1\_cost')  
 .eq('strategic\_goal\_id', goalId)  
  
// ❌ Bad: Select all columns  
const { data } = await supabase  
 .from('initiatives')  
 .select('\*')

**4. Caching Strategy**:

// Page-level caching (Next.js)  
export const revalidate = 60 // Revalidate every 60 seconds  
  
// Route segment config  
export const dynamic = 'force-static' // Static generation  
export const fetchCache = 'force-cache'

**5. Image Optimization**:

import Image from 'next/image'  
  
<Image  
 src="/department-logo.png"  
 alt="Department Logo"  
 width={200}  
 height={100}  
 priority // For above-the-fold images  
/>

**6. Bundle Size Optimization**:

// Dynamic imports for large components  
import dynamic from 'next/dynamic'  
  
const PDFExporter = dynamic(() => import('@/components/PDFExporter'), {  
 loading: () => <LoadingSpinner />,  
 ssr: false, // Only load on client  
})

## Testing Strategy

### Testing Pyramid

┌─────────────┐  
 │ E2E (10%) │ Playwright - Critical user flows  
 └─────────────┘  
 ┌───────────────┐  
 │ Integration │ Vitest - Component + DB  
 │ (30%) │  
 └───────────────┘  
 ┌─────────────────┐  
 │ Unit (60%) │ Vitest - Business logic  
 └─────────────────┘

### Unit Tests (Vitest)

// tests/unit/budget-validation.test.ts  
import { describe, it, expect } from 'vitest'  
import { validateBudget } from '@/lib/validation/budget'  
  
describe('validateBudget', () => {  
 it('should return valid for matching totals', () => {  
 const budget = {  
 year\_1: {  
 personnel\_costs: 75000,  
 equipment\_technology: 50000,  
 professional\_services: 25000,  
 training\_development: 10000,  
 materials\_supplies: 5000,  
 other\_costs: 0,  
 total: 165000,  
 },  
 funding\_sources: [  
 { source: 'General Fund', amount: 100000, status: 'secured' },  
 { source: 'EPA Grant', amount: 65000, status: 'pending' },  
 ],  
 }  
  
 const result = validateBudget(budget)  
 expect(result.valid).toBe(true)  
 })  
  
 it('should return error for mismatched totals', () => {  
 const budget = {  
 year\_1: {  
 personnel\_costs: 75000,  
 equipment\_technology: 50000,  
 professional\_services: 25000,  
 training\_development: 10000,  
 materials\_supplies: 5000,  
 other\_costs: 0,  
 total: 200000, // Wrong total  
 },  
 funding\_sources: [  
 { source: 'General Fund', amount: 100000, status: 'secured' },  
 ],  
 }  
  
 const result = validateBudget(budget)  
 expect(result.valid).toBe(false)  
 expect(result.error).toContain('Budget mismatch')  
 })  
})

### Integration Tests

// tests/integration/create-initiative.test.ts  
import { describe, it, expect, beforeAll, afterAll } from 'vitest'  
import { createClient } from '@supabase/supabase-js'  
import { createInitiative } from '@/app/actions/initiatives'  
  
describe('createInitiative', () => {  
 let supabase  
 let testGoalId  
  
 beforeAll(async () => {  
 supabase = createClient(  
 process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,  
 process.env.SUPABASE\_SERVICE\_ROLE\_KEY!  
 )  
  
 // Create test goal  
 const { data: goal } = await supabase  
 .from('strategic\_goals')  
 .insert({ title: 'Test Goal', ... })  
 .select()  
 .single()  
 testGoalId = goal.id  
 })  
  
 afterAll(async () => {  
 // Cleanup  
 await supabase  
 .from('strategic\_goals')  
 .delete()  
 .eq('id', testGoalId)  
 })  
  
 it('should create initiative successfully', async () => {  
 const formData = new FormData()  
 formData.append('strategic\_goal\_id', testGoalId)  
 formData.append('name', 'Test Initiative')  
 formData.append('priority\_level', 'NEED')  
 formData.append('description', 'This is a test initiative description.')  
  
 const result = await createInitiative(formData)  
  
 expect(result.error).toBeUndefined()  
 expect(result.data).toBeDefined()  
 expect(result.data.name).toBe('Test Initiative')  
 })  
})

### End-to-End Tests (Playwright)

// tests/e2e/create-plan.spec.ts  
import { test, expect } from '@playwright/test'  
  
test.describe('Strategic Plan Creation', () => {  
 test.beforeEach(async ({ page }) => {  
 // Login  
 await page.goto('/login')  
 await page.fill('[name="email"]', 'director@carrollton.gov')  
 await page.fill('[name="password"]', 'test-password')  
 await page.click('button[type="submit"]')  
 await page.waitForURL('/dashboard')  
 })  
  
 test('should create new strategic plan', async ({ page }) => {  
 // Navigate to create plan  
 await page.click('text=Create New Plan')  
 await expect(page).toHaveURL(/\/plans\/new/)  
  
 // Fill form  
 await page.fill('[name="title"]', 'FY2026-2028 Strategic Plan')  
 await page.fill('[name="executive\_summary"]', 'This is a test summary.')  
  
 // Submit  
 await page.click('button:has-text("Create Plan")')  
  
 // Verify redirect and success  
 await page.waitForURL(/\/plans\/[a-f0-9-]+/)  
 await expect(page.locator('h1')).toContainText('FY2026-2028 Strategic Plan')  
 })  
})

## Monitoring & Observability

### Logging Strategy

**Structured Logging with Pino** (optional):

// lib/logger.ts  
import pino from 'pino'  
  
export const logger = pino({  
 level: process.env.NODE\_ENV === 'production' ? 'info' : 'debug',  
 transport: process.env.NODE\_ENV !== 'production'  
 ? { target: 'pino-pretty' }  
 : undefined,  
})  
  
// Usage  
logger.info({ initiativeId, userId }, 'Initiative created successfully')  
logger.error({ error, initiativeId }, 'Failed to create initiative')

### Error Monitoring

**Sentry Integration**:

// lib/sentry.ts  
import \* as Sentry from '@sentry/nextjs'  
  
Sentry.init({  
 dsn: process.env.NEXT\_PUBLIC\_SENTRY\_DSN,  
 tracesSampleRate: 1.0,  
 environment: process.env.NODE\_ENV,  
})  
  
// Capture errors  
try {  
 await createInitiative(data)  
} catch (error) {  
 Sentry.captureException(error, {  
 tags: { action: 'create\_initiative' },  
 extra: { initiativeData: data },  
 })  
 throw error  
}

### Performance Monitoring

**Vercel Analytics** (built-in):

// app/layout.tsx  
import { Analytics } from '@vercel/analytics/react'  
  
export default function RootLayout({ children }) {  
 return (  
 <html>  
 <body>  
 {children}  
 <Analytics />  
 </body>  
 </html>  
 )  
}

## Coding Standards

### TypeScript Guidelines

1. **Strict mode enabled** - "strict": true in tsconfig.json
2. **No explicit any** - Use unknown if type is truly unknown
3. **Prefer interfaces over types** for object shapes
4. **Use type inference** when obvious
5. **Export types with implementations**

### Component Guidelines

1. **One component per file**
2. **Server Components by default** - Only add 'use client' when necessary
3. **Props interface named <ComponentName>Props**
4. **Destructure props in function signature**
5. **Use composition over inheritance**

**Example**:

// components/InitiativeCard.tsx  
interface InitiativeCardProps {  
 initiative: Initiative  
 onEdit?: (id: string) => void  
}  
  
export function InitiativeCard({ initiative, onEdit }: InitiativeCardProps) {  
 return (  
 <Card>  
 <CardHeader>  
 <CardTitle>{initiative.name}</CardTitle>  
 </CardHeader>  
 <CardContent>  
 <p>{initiative.description}</p>  
 </CardContent>  
 {onEdit && (  
 <CardFooter>  
 <Button onClick={() => onEdit(initiative.id)}>Edit</Button>  
 </CardFooter>  
 )}  
 </Card>  
 )  
}

### File Naming Conventions

* **Components**: PascalCase (InitiativeCard.tsx)
* **Utilities**: camelCase (formatCurrency.ts)
* **Types**: PascalCase (Initiative, User)
* **Constants**: SCREAMING\_SNAKE\_CASE (MAX\_INITIATIVES)
* **Routes**: kebab-case folders (/strategic-plans/)

## Error Handling

### Error Handling Patterns

**1. Server Actions**:

// app/actions/initiatives.ts  
'use server'  
  
import { z } from 'zod'  
  
export async function createInitiative(formData: FormData) {  
 try {  
 // Validate  
 const parsed = initiativeSchema.safeParse(Object.fromEntries(formData))  
 if (!parsed.success) {  
 return {  
 success: false,  
 error: {  
 type: 'validation',  
 message: 'Invalid input',  
 issues: parsed.error.flatten(),  
 },  
 }  
 }  
  
 // Insert  
 const { data, error } = await supabase  
 .from('initiatives')  
 .insert(parsed.data)  
 .select()  
 .single()  
  
 if (error) {  
 logger.error({ error }, 'Database error creating initiative')  
 return {  
 success: false,  
 error: {  
 type: 'database',  
 message: 'Failed to create initiative',  
 },  
 }  
 }  
  
 revalidatePath('/dashboard')  
 return { success: true, data }  
  
 } catch (error) {  
 logger.error({ error }, 'Unexpected error creating initiative')  
 Sentry.captureException(error)  
 return {  
 success: false,  
 error: {  
 type: 'unknown',  
 message: 'An unexpected error occurred',  
 },  
 }  
 }  
}

**2. Client Error Boundaries**:

// app/error.tsx  
'use client'  
  
import { useEffect } from 'react'  
import \* as Sentry from '@sentry/nextjs'  
  
export default function Error({  
 error,  
 reset,  
}: {  
 error: Error & { digest?: string }  
 reset: () => void  
}) {  
 useEffect(() => {  
 Sentry.captureException(error)  
 }, [error])  
  
 return (  
 <div className="flex min-h-screen items-center justify-center">  
 <div className="text-center">  
 <h2 className="text-2xl font-bold">Something went wrong!</h2>  
 <button  
 onClick={() => reset()}  
 className="mt-4 rounded bg-blue-600 px-4 py-2 text-white"  
 >  
 Try again  
 </button>  
 </div>  
 </div>  
 )  
}

**3. User-Friendly Error Messages**:

// lib/errors/messages.ts  
export const ERROR\_MESSAGES = {  
 validation: {  
 required: 'This field is required',  
 minLength: (min: number) => `Must be at least ${min} characters`,  
 email: 'Invalid email address',  
 },  
 database: {  
 duplicate: 'A record with this value already exists',  
 notFound: 'Record not found',  
 constraint: 'Cannot delete: record is referenced elsewhere',  
 },  
 auth: {  
 unauthorized: 'You do not have permission to perform this action',  
 unauthenticated: 'Please log in to continue',  
 },  
} as const

## Appendix

### Glossary

* **RLS** - Row-Level Security (database-level access control)
* **RSC** - React Server Components
* **BaaS** - Backend as a Service
* **JSONB** - PostgreSQL’s binary JSON data type
* **SWOT** - Strengths, Weaknesses, Opportunities, Threats analysis
* **KPI** - Key Performance Indicator
* **FY** - Fiscal Year

### Key Dependencies Documentation

* [Next.js 14 Docs](https://nextjs.org/docs)
* [Supabase Docs](https://supabase.com/docs)
* [React Hook Form](https://react-hook-form.com/)
* [Zod](https://zod.dev/)
* [TailwindCSS](https://tailwindcss.com/)
* [shadcn/ui](https://ui.shadcn.com/)

### Architecture Decision Records (ADRs)

**ADR-001: Hybrid Data Model** - **Decision**: Use normalized relational tables + JSONB - **Rationale**: Balance queryability with flexibility - **Status**: Approved

**ADR-002: Server Components First** - **Decision**: Default to Server Components, minimize client JS - **Rationale**: Better performance, SEO, security - **Status**: Approved

**ADR-003: Supabase RLS for Security** - **Decision**: Enforce all access control via RLS policies - **Rationale**: Defense in depth, prevents bypass - **Status**: Approved

**ADR-004: No Custom API Layer** - **Decision**: Use Supabase PostgREST, no custom REST/GraphQL - **Rationale**: Reduce complexity, leverage auto-generated API - **Status**: Approved

**Document Status:** ✅ Complete **Last Reviewed:** January 9, 2025 **Next Review:** February 9, 2025