# 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