# 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>  
 )  
}