Multi-Agent SDLC Automation Platform

Phase 1 Implementation Task Breakdown: Foundation & Core Components

Document Version: 1.0
Target Timeline: Weeks 1-2 (Foundation Phase)
Status: Ready for Sprint Planning
Tech Stack: React 18.3 + TypeScript + Next.js 14 + Tailwind CSS + Radix UI

Executive Summary

This document provides a granular, actionable breakdown of Phase 1 implementation tasks for the Multi-Agent SDLC Automation Platform UI. Each task includes clear acceptance criteria, estimated effort, dependencies, and technical specifications aligned with the design system documented in the UI/UX specification.

Phase 1 Goal: Establish the foundational architecture, design system implementation, and core navigation components that all subsequent features will build upon.

Project Setup & Environment Configuration

TASK 1.1: Initialize Next.js 14 Project with TypeScript

Priority: P0 (Blocking)
Estimated Effort: 2 hours
Dependencies: None

Detailed Steps:

1. **Create Next.js project with App Router:**
   npx create-next-app@14 sdlc-automation-ui --typescript --tailwind --app --no-src-dir
   cd sdlc-automation-ui
2. **Configure TypeScript strict mode** (tsconfig.json):
   {
   "compilerOptions": {
   "strict": true,
   "noUncheckedIndexedAccess": true,
   "noImplicitReturns": true,
   "noFallthroughCasesInSwitch": true,
   "esModuleInterop": true,
   "skipLibCheck": true,
   "forceConsistentCasingInFileNames": true,
   "module": "esnext",
   "moduleResolution": "bundler",
   "resolveJsonModule": true,

"isolatedModules": true,
"jsx": "preserve",
"incremental": true,
"paths": {
"@/*": ["./*"],
"@/components/*": ["./components/*"],
"@/lib/*": ["./lib/*"],
"@/types/*": ["./types/*"]
}
}
}

3. **Install essential dependencies:**
   npm install zustand @tanstack/react-query recharts
   npm install -D @types/node
4. **Create directory structure:**
   sdlc-automation-ui/
   ├── app/
   │ ├── layout.tsx
   │ ├── page.tsx (Dashboard)
   │ ├── requirements/
   │ │ └── page.tsx
   │ ├── pipeline/
   │ │ └── page.tsx
   │ ├── telemetry/
   │ │ └── page.tsx
   │ └── settings/
   │ └── page.tsx
   ├── components/
   │ ├── ui/ # Reusable UI primitives
   │ ├── layout/ # Navigation, sidebar
   │ ├── features/ # Feature-specific components
   │ └── icons/ # SVG icon components
   ├── lib/
   │ ├── utils.ts # Utility functions
   │ └── constants.ts # App-wide constants
   ├── types/
   │ ├── requirements.ts
   │ └── agent.ts
   ├── stores/
   │ └── ui.ts # Zustand store
   └── styles/
   └── globals.css

Acceptance Criteria:

- [x] Next.js 14 project runs without errors on `localhost:3000`
- [x] TypeScript strict mode enabled with zero compilation errors
- [x] All path aliases resolve correctly
- [x] Directory structure matches specification
- [x] `package.json` includes all required dependencies

Validation Command:
npm run build && npm run type-check

TASK 1.2: Configure Tailwind CSS with Design System Tokens

Priority: P0 (Blocking)
Estimated Effort: 3 hours
Dependencies: TASK 1.1

Detailed Steps:

1. **Update `tailwind.config.ts` with complete design system:**

```
import type { Config } from 'tailwindcss'
const config: Config = {
content: [
'./pages//*.{js,ts,jsx,tsx,mdx}',
'./components//.{js,ts,jsx,tsx,mdx}',
'./app/**/.{js,ts,jsx,tsx,mdx}',
],
darkMode: 'class',
theme: {
extend: {
colors: {
// Background layers
'bg-primary': '#0d1117',
'bg-secondary': '#161b22',
'bg-tertiary': '#21262d',
'bg-elevated': '#2d333b',

    // Borders
    'border-primary': '#30363d',
    'border-secondary': '#373e47',

    // Text
    'text-primary': '#e6edf3',
    'text-secondary': '#8d96a0',
    'text-tertiary': '#636e7b',

    // Accents
    'accent-primary': '#58a6ff',
    'accent-secondary': '#1f6feb',
    'accent-tertiary': '#388bfd',

    // Agent colors
    'agent-finalize': '#58a6ff',
```

```
    'agent-design': '#bc8cff',
    'agent-code': '#3fb950',
    'agent-test': '#d29922',

    // Semantic
    success: '#3fb950',
    warning: '#d29922',
    error: '#f85149',
    info: '#58a6ff',
    purple: '#bc8cff',
    teal: '#56d4dd',
  },
  fontFamily: {
    sans: [
      '-apple-system',
      'BlinkMacSystemFont',
      '"Segoe UI"',
      '"Noto Sans"',
      'Helvetica',
      'Arial',
      'sans-serif',
    ],
    mono: [
      '"Berkeley Mono"',
      '"SF Mono"',
      '"Courier New"',
      'monospace',
    ],
  },
  fontSize: {
    xs: ['11px', { lineHeight: '1.5' }],
    sm: ['12px', { lineHeight: '1.5' }],
    base: ['14px', { lineHeight: '1.6' }],
    md: ['14px', { lineHeight: '1.6' }],
    lg: ['16px', { lineHeight: '1.5' }],
    xl: ['18px', { lineHeight: '1.4' }],
    '2xl': ['20px', { lineHeight: '1.3' }],
    '3xl': ['24px', { lineHeight: '1.3' }],
```

```
      '4xl': ['30px', { lineHeight: '1.2' }],
      'display': ['32px', { lineHeight: '1.2' }],
    },
    spacing: {
      xs: '4px',
      sm: '8px',
      md: '16px',
      lg: '24px',
      xl: '32px',
    },
    borderRadius: {
      sm: '6px',
      DEFAULT: '8px',
      md: '8px',
      lg: '12px',
      full: '9999px',
    },
    boxShadow: {
      sm: '0 2px 8px rgba(0, 0, 0, 0.2)',
      DEFAULT: '0 4px 12px rgba(0, 0, 0, 0.3)',
      md: '0 4px 12px rgba(0, 0, 0, 0.3)',
      lg: '0 8px 24px rgba(0, 0, 0, 0.4)',
    },
    transitionDuration: {
      fast: '150ms',
      normal: '250ms',
    },
  },
},
plugins: [],
}
export default config
```

2. **Create `styles/globals.css` with base styles:**

```
@tailwind base;
@tailwind components;
@tailwind utilities;
@layer base {
  * {
    @apply box-border;
  }
```

```
html {
@apply text-base font-sans text-text-primary bg-bg-primary;
-webkit-font-smoothing: antialiased;
}
body {
@apply m-0 p-0 overflow-x-hidden;
}
/* Custom scrollbar */
::-webkit-scrollbar {
@apply w-2 h-2;
}
::-webkit-scrollbar-track {
@apply bg-bg-primary;
}
::-webkit-scrollbar-thumb {
@apply bg-bg-tertiary rounded;
}
::-webkit-scrollbar-thumb:hover {
@apply bg-bg-elevated;
}
}
@layer components {
/* Component-specific utilities will be added here */
}
```

3. **Create utility helper file** (`lib/utils.ts`):
```
import { type ClassValue, clsx } from 'clsx'
import { twMerge } from 'tailwind-merge'
export function cn(...inputs: ClassValue[]) {
return twMerge(clsx(inputs))
}
// Install dependencies
// npm install clsx tailwind-merge
```

4. **Test design tokens:**
Create `app/design-test/page.tsx`:
```
export default function DesignTest() {
return (
<div className="p-lg space-y-md">
```

Design System Test

Typography and colors

```
<div className="grid grid-cols-4 gap-md">
  <div className="bg-agent-finalize p-md rounded">Finalize</div>
  <div className="bg-agent-design p-md rounded">Design</div>
  <div className="bg-agent-code p-md rounded">Code</div>
  <div className="bg-agent-test p-md rounded">Test</div>
```

```
    </div>
  </div>
```

)
}

Acceptance Criteria:

- [x] All design system colors accessible via Tailwind classes
- [x] Typography scale matches specification (14px base)
- [x] Spacing system uses 8px grid (4px increments)
- [x] Dark mode theme renders correctly
- [x] Custom scrollbar styles applied
- [x] `cn()` utility function merges classes without conflicts
- [x] Design test page displays all color tokens correctly

Validation:

- Visual inspection of `/design-test` route
- No Tailwind warnings in console
- All custom classes resolve in DevTools

TASK 1.3: Install and Configure Radix UI Primitives

Priority: P0 (Blocking)
Estimated Effort: 2 hours
Dependencies: TASK 1.2

Detailed Steps:

1. **Install Radix UI packages:**
   npm install @radix-ui/react-dialog
   @radix-ui/react-dropdown-menu
   @radix-ui/react-tabs
   @radix-ui/react-tooltip
   @radix-ui/react-alert-dialog
   @radix-ui/react-select
   @radix-ui/react-switch
   @radix-ui/react-slot
2. **Create Radix wrapper components** (`components/ui/`):
   Dialog Component (`components/ui/dialog.tsx`):
   'use client'
   import * as React from 'react'
   import * as DialogPrimitive from '@radix-ui/react-dialog'
   import { cn } from '@/lib/utils'
   const Dialog = DialogPrimitive.Root
   const DialogTrigger = DialogPrimitive.Trigger
   const DialogContent = React.forwardRef<
   React.ElementRef<typeof DialogPrimitive.Content>,
   React.ComponentPropsWithoutRef<typeof DialogPrimitive.Content>

```
    (({ className, children, ...props }, ref) => (
    <DialogPrimitive.Portal>
    <DialogPrimitive.Overlay className="fixed inset-0 z-50 bg-black/50" />
    <DialogPrimitive.Content
    ref={ref}
    className={cn(
    'fixed left-[50%] top-[50%] z-50 w-full max-w-[800px]',
    'translate-x-[-50%] translate-y-[-50%]',
    'bg-bg-elevated rounded-lg p-lg shadow-lg',
    className
    )}
    {...props}
```

```
  {children}
</DialogPrimitive.Content>
```

```
</DialogPrimitive.Portal>
))
DialogContent.displayName = 'DialogContent'
export { Dialog, DialogTrigger, DialogContent }
```

Tabs Component (components/ui/tabs.tsx):

```
'use client'
import * as React from 'react'
import * as TabsPrimitive from '@radix-ui/react-tabs'
import { cn } from '@/lib/utils'
const Tabs = TabsPrimitive.Root
const TabsList = React.forwardRef<
React.ElementRef<typeof TabsPrimitive.List>,
React.ComponentPropsWithoutRef<typeof TabsPrimitive.List>
    (({ className, ...props }, ref) => (
    <TabsPrimitive.List
    ref={ref}
    className={cn('flex gap-sm', className)}
    {...props}
    />
    ))
    TabsList.displayName = 'TabsList'

const TabsTrigger = React.forwardRef<
React.ElementRef<typeof TabsPrimitive.Trigger>,
React.ComponentPropsWithoutRef<typeof TabsPrimitive.Trigger>
```

```
  (({ className, ...props }, ref) => (
  <TabsPrimitive.Trigger
  ref={ref}
  className={cn(
  'px-md py-sm rounded-md transition-fast',
  'text-text-secondary hover:bg-bg-tertiary hover:text-text-primary',
  'data-[state=active]:bg-accent-secondary data-[state=active]:text-white',
  className
  )}
  {...props}
  />
  ))
  TabsTrigger.displayName = 'TabsTrigger'
```

const TabsContent = TabsPrimitive.Content
export { Tabs, TabsList, TabsTrigger, TabsContent }

3. **Create component documentation:**
// components/ui/README.md

UI Components

Radix UI Primitives

All components are built on Radix UI for accessibility compliance:

- WAI-ARIA compliant
- Keyboard navigation
- Screen reader support

Usage

import { Dialog, DialogTrigger, DialogContent } from '@/components/ui/dialog'
import { Tabs, TabsList, TabsTrigger, TabsContent } from '@/components/ui/tabs'

Acceptance Criteria:

- [x] All Radix packages installed without peer dependency warnings
- [x] Dialog, Tabs components styled with design system tokens
- [x] Components accept className prop for custom styling
- [x] TypeScript types properly inferred
- [x] Keyboard navigation works (Tab, Enter, Escape)
- [x] ARIA attributes present in rendered HTML

Validation:

- Test dialog open/close with Escape key
- Tab through interactive elements
- Inspect HTML for ARIA attributes

Core Component Development

TASK 2.1: Build Button Component System

Priority: P0 (Blocking)
Estimated Effort: 3 hours
Dependencies: TASK 1.2, TASK 1.3

Detailed Steps:

1. **Create button component** (components/ui/button.tsx):

```
import * as React from 'react'
import { Slot } from '@radix-ui/react-slot'
import { cva, type VariantProps } from 'class-variance-authority'
import { cn } from '@/lib/utils'
// Install CVA: npm install class-variance-authority
const buttonVariants = cva(
// Base styles
'inline-flex items-center justify-center rounded-md transition-fast',
'font-medium cursor-pointer disabled:opacity-50 disabled:pointer-events-none',
{
variants: {
variant: {
primary: 'bg-accent-primary text-white hover:bg-accent-tertiary active:scale-[0.98]',
secondary: 'bg-bg-tertiary text-text-primary border border-border-primary hover:bg-
bg-elevated hover:border-border-secondary',
outline: 'bg-transparent text-text-primary border border-border-primary hover:bg-
bg-tertiary',
ghost: 'bg-transparent hover:bg-bg-tertiary',
},
size: {
sm: 'h-7 px-3 text-sm',
default: 'h-8 px-md text-base',
lg: 'h-10 px-5 text-lg',
icon: 'h-8 w-8',
},
},
defaultVariants: {
variant: 'primary',
size: 'default',
},
}
)
export interface ButtonProps
extends React.ButtonHTMLAttributes<HTMLButtonElement>,
VariantProps<typeof buttonVariants> {
asChild?: boolean
}
const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
({ className, variant, size, asChild = false, ...props }, ref) => {
const Comp = asChild ? Slot : 'button'
```

return (
<Comp
className={cn(buttonVariants({ variant, size, className }))}
ref={ref}
{...props}
/>
)
}
)
Button.displayName = 'Button'
export { Button, buttonVariants }

2. **Create Storybook-style showcase** (app/components-test/page.tsx):
import { Button } from '@/components/ui/button'
export default function ComponentTest() {
return (

Button Variants

Primary Secondary Outline Ghost

```
  <section>
    <h2 className="text-xl font-semibold mb-md">Button Sizes</h2>
    <div className="flex gap-md items-center">
      <Button size="sm">Small</Button>
      <Button size="default">Default</Button>
      <Button size="lg">Large</Button>
      <Button size="icon">□</Button>
    </div>
  </section>

  <section>
    <h2 className="text-xl font-semibold mb-md">Button States</h2>
    <div className="flex gap-md">
      <Button disabled>Disabled</Button>
      <Button variant="secondary" disabled>Disabled Secondary</Button>
    </div>
  </section>
</div>
```

)
}

Acceptance Criteria:

- [x] All 4 variants render correctly (primary, secondary, outline, ghost)
- [x] All 4 sizes render correctly (sm, default, lg, icon)
- [x] Hover states darken background by ~10%
- [x] Active state scales to 0.98
- [x] Disabled state shows 50% opacity and no pointer events
- [x] TypeScript autocomplete works for variant/size props
- [x] Component accepts all standard button HTML attributes
- [x] Focus-visible shows blue outline (accessibility)

Validation:

- Visual inspection at `/components-test`
- Tab to buttons and verify focus ring
- Test hover/active states in all variants

TASK 2.2: Build Input Field Components

Priority: P0 (Blocking)
Estimated Effort: 4 hours
Dependencies: TASK 1.2

Detailed Steps:

1. **Create text input component** (`components/ui/input.tsx`):
   ```
   import * as React from 'react'
   import { cn } from '@/lib/utils'
   export interface InputProps
   extends React.InputHTMLAttributes<HTMLInputElement> {
   error?: boolean
   }
   const Input = React.forwardRef<HTMLInputElement, InputProps>(
   ({ className, type, error, ...props }, ref) => {
   return (
   <input
   type={type}
   className={cn(
   'w-full h-10 px-md rounded-md transition-fast',
   'bg-bg-tertiary border border-border-primary',
   'text-text-primary placeholder:text-text-tertiary',
   'hover:bg-bg-elevated hover:border-border-secondary',
   'focus:outline-none focus:border-accent-primary focus:ring-3 focus:ring-accent-primary/20',
   error && 'border-error focus:border-error focus:ring-error/20',
   'disabled:opacity-50 disabled:cursor-not-allowed',
   className
   )}
   ref={ref}
   {...props}
   />
   )
   }
   ```

```
)
Input.displayName = 'Input'
export { Input }
```

2. **Create textarea component** (`components/ui/textarea.tsx`):

```
import * as React from 'react'
import { cn } from '@/lib/utils'
export interface TextareaProps
extends React.TextareaHTMLAttributes<HTMLTextAreaElement> {
error?: boolean
}
const Textarea = React.forwardRef<HTMLTextAreaElement, TextareaProps>(
({ className, error, ...props }, ref) => {
return (
<textarea
className={cn(
'w-full min-h-[120px] px-md py-3 rounded-md resize-y transition-fast',
'bg-bg-tertiary border border-border-primary',
'text-text-primary placeholder:text-text-tertiary',
'font-mono text-sm leading-relaxed',
'hover:bg-bg-elevated hover:border-border-secondary',
'focus:outline-none focus:border-accent-primary focus:ring-3 focus:ring-accent-
primary/20',
error && 'border-error focus:border-error focus:ring-error/20',
'disabled:opacity-50 disabled:cursor-not-allowed',
className
)}
ref={ref}
{...props}
/>
)
}
)
Textarea.displayName = 'Textarea'
export { Textarea }
```

3. **Create label component** (`components/ui/label.tsx`):

```
import * as React from 'react'
import * as LabelPrimitive from '@radix-ui/react-label'
import { cn } from '@/lib/utils'
// npm install @radix-ui/react-label
const Label = React.forwardRef<
React.ElementRef<typeof LabelPrimitive.Root>,
React.ComponentPropsWithoutRef<typeof LabelPrimitive.Root> & {
required?: boolean
}
```

```
  (({ className, required, children, ...props }, ref) => (
  <LabelPrimitive.Root
  ref={ref}
  className={cn(
  'block text-sm font-medium text-text-secondary mb-sm',
  className
  )}
  {...props}
```

```
{children}
{required && <span className="text-error ml-1">*</span>}
```

```
</LabelPrimitive.Root>
))
Label.displayName = 'Label'
export { Label }
```

4. **Create form field group component** (components/ui/form-field.tsx):

```
import * as React from 'react'
import { Label } from './label'
import { cn } from '@/lib/utils'
interface FormFieldProps {
label: string
required?: boolean
error?: string
helperText?: string
children: React.ReactNode
className?: string
}
export function FormField({
label,
required,
error,
helperText,
children,
className,
}: FormFieldProps) {
return (
<div className={cn('space-y-sm', className)}>
{label}
{children}
{error && (
{error}


)}
{helperText && !error && (
{helperText}
```

```
)}
</div>
)
}
```

5. **Add to component test page:**
   ```
   // Add to app/components-test/page.tsx
   import { Input } from '@/components/ui/input'
   import { Textarea } from '@/components/ui/textarea'
   import { FormField } from '@/components/ui/form-field'
   ```

   ## Form Fields

   

Acceptance Criteria:

- [x] Input fields have 40px height
- [x] Textarea has 120px min-height and vertical resize
- [x] Textarea uses monospace font
- [x] Hover states change border color
- [x] Focus states show blue ring without browser default outline
- [x] Error state shows red border and ring
- [x] Label displays red asterisk for required fields
- [x] FormField groups label, input, and error message
- [x] All components forward refs correctly
- [x] Placeholder text uses text-tertiary color

Validation:

- Tab through form fields
- Type in inputs and verify styling
- Test error state rendering

TASK 2.3: Build Card Component

Priority: P1 (High)
Estimated Effort: 2 hours
Dependencies: TASK 1.2

Detailed Steps:

1. **Create card component** (`components/ui/card.tsx`):
   ```
   import * as React from 'react'
   import { cn } from '@/lib/utils'
   const Card = React.forwardRef<
   HTMLDivElement,
   React.HTMLAttributes<HTMLDivElement>
   ```

```
  (({ className, ...props }, ref) => (
  <div
  ref={ref}
  className={cn(
  'bg-bg-secondary rounded-lg border border-border-primary shadow-sm',
  'transition-shadow hover:shadow-md',
  className
  )}
  {...props}
  />
  ))
  Card.displayName = 'Card'

const CardHeader = React.forwardRef<
HTMLDivElement,
React.HTMLAttributes<HTMLDivElement>
  (({ className, ...props }, ref) => (
  <div
  ref={ref}
  className={cn('p-lg border-b border-border-primary', className)}
  {...props}
  />
  ))
  CardHeader.displayName = 'CardHeader'

const CardBody = React.forwardRef<
HTMLDivElement,
React.HTMLAttributes<HTMLDivElement>
  (({ className, ...props }, ref) => (
  <div
  ref={ref}
  className={cn('p-lg', className)}
  {...props}
  />
  ))
  CardBody.displayName = 'CardBody'

const CardFooter = React.forwardRef<
HTMLDivElement,
React.HTMLAttributes<HTMLDivElement>
  (({ className, ...props }, ref) => (
  <div
  ref={ref}
  className={cn('p-lg border-t border-border-primary', className)}
  {...props}
  />
  ))
  CardFooter.displayName = 'CardFooter'

export { Card, CardHeader, CardBody, CardFooter }
```

2. **Create stat card component** (components/ui/stat-card.tsx):
```
import * as React from 'react'
```

import { Card, CardBody } from './card'
import { cn } from '@/lib/utils'
interface StatCardProps {
label: string
value: string | number
change?: string
icon: React.ReactNode
iconColor?: string
className?: string
}
export function StatCard({
label,
value,
change,
icon,
iconColor = 'bg-accent-primary/20 text-accent-primary',
className,
}: StatCardProps) {
return (
<Card className={cn('min-w-[320px]', className)}>
<CardBody>

{label}

<div className={cn('w-8 h-8 rounded-md flex items-center justify-center text-base', iconColor)}>
{icon}

```
      <div className="text-4xl font-bold text-text-primary mb-sm">
        {value}
      </div>

      {change && (
        <div className="text-xs text-success">
          {change}
        </div>
      )}
    </CardBody>
  </Card>
```

)
}

3. **Add to component test:**
import { Card, CardHeader, CardBody } from '@/components/ui/card'

import { StatCard } from '@/components/ui/stat-card'

Cards

Card Title

Card content goes here

Acceptance Criteria:

- [x] Card has bg-secondary background
- [x] Border is 1px solid border-primary
- [x] Border radius is 12px (rounded-lg)
- [x] Hover state elevates shadow (shadow-md)
- [x] CardHeader/CardFooter have border separators
- [x] StatCard displays icon in colored background
- [x] StatCard value is 32px font size, weight 700
- [x] Label text is uppercase, 11px, tracking-wide

Validation:

- Hover over cards to see shadow elevation
- Verify spacing matches 16px/24px grid

Navigation & Layout Components

TASK 3.1: Build Top Navbar Component

Priority: P0 (Blocking)
Estimated Effort: 4 hours
Dependencies: TASK 2.1

Detailed Steps:

1. **Create navbar component** (components/layout/navbar.tsx):
'use client'
import { useState } from 'react'
import Link from 'next/link'
import { usePathname } from 'next/navigation'
import { cn } from '@/lib/utils'
const NAV_ITEMS = [
{ label: 'Dashboard', href: '/' },
{ label: 'Requirements', href: '/requirements' },
{ label: 'Pipeline', href: '/pipeline' },
{ label: 'Telemetry', href: '/telemetry' },
{ label: 'Settings', href: '/settings' },
] as const
export function Navbar() {
const pathname = usePathname()
return (
<nav className="sticky top-0 z-50 w-full h-[60px] bg-bg-secondary border-b border-border-primary shadow-sm">

{/* Left Section /}

{/ Logo */}

Application
</Link>

```
        {/* Navigation Tabs */}
        <div className="flex gap-sm">
          {NAV_ITEMS.map((item) => {
            const isActive = pathname === item.href
            return (
              <Link
                key={item.href}
                href={item.href}
                className={cn(
                  'px-md py-sm rounded-md text-sm transition-fast',
                  isActive
                    ? 'bg-accent-secondary text-white'
                    : 'text-text-secondary hover:bg-bg-tertiary hover:text-text-prima
                )}
              >
                {item.label}
              </Link>
            )
          })}
        </div>
      </div>

      {/* Right Section */}
      <div className="flex items-center gap-md">
        {/* Status Indicator */}
        <div className="flex items-center gap-sm px-md py-sm bg-bg-tertiary
          <span className="w-2 h-2 rounded-full bg-success animate-pulse" />
          <span>All Systems Operational</span>
        </div>

        {/* Cost Badge */}
        <div className="px-md py-sm bg-bg-tertiary rounded-md text-xs font-
```

```
          $0.42 / $10.00 daily
        </div>
      </div>
    </div>
  </nav>
```

)
}

2. **Add pulse animation to globals.css:**
   @layer utilities {
   @keyframes pulse {
   0%, 100% { opacity: 1; }
   50% { opacity: 0.5; }
   }
   .animate-pulse {
   animation: pulse 2s cubic-bezier(0.4, 0, 0.6, 1) infinite;
   }
   }
3. **Update root layout** (app/layout.tsx):
   import { Navbar } from '@/components/layout/navbar'
   import './globals.css'
   export const metadata = {
   title: 'Multi-Agent SDLC Automation',
   description: 'Automated requirements finalization and agent orchestration',
   }
   export default function RootLayout({
   children,
   }: {
   children: React.ReactNode
   }) {
   return (

   {children}

)
   }

Acceptance Criteria:

- [x] Navbar is sticky with z-index 50
- [x] Height exactly 60px
- [x] Logo is 36×36px with gradient background
- [x] Active tab has blue background
- [x] Inactive tabs have gray text, hover shows tertiary bg
- [x] Status dot pulses with 2s animation
- [x] Cost badge uses monospace font
- [x] usePathname correctly identifies active route
- [x] All transitions are 150ms

Validation:

- Navigate between routes and verify active state
- Check sticky behavior on scroll
- Verify status dot animation

TASK 3.2: Build Collapsible Sidebar Component

Priority: P0 (Blocking)
Estimated Effort: 5 hours
Dependencies: TASK 2.1, TASK 3.1

Detailed Steps:

1. **Create Zustand store for sidebar state** (stores/ui.ts):
   import { create } from 'zustand'
   import { persist } from 'zustand/middleware'
   interface UIState {
   sidebarOpen: boolean
   toggleSidebar: () => void
   setSidebarOpen: (open: boolean) => void
   }
   export const useUIStore = create<UIState>()(
   persist(
   (set) => ({
   sidebarOpen: true,
   toggleSidebar: () => set((state) => ({ sidebarOpen: !state.sidebarOpen })),
   setSidebarOpen: (open) => set({ sidebarOpen: open }),
   }),
   {
   name: 'ui-storage',
   }
)
)
2. **Create agent types** (types/agent.ts):
   export type AgentType = 'finalize' | 'design' | 'code' | 'test'
   export type AgentStatus = 'production' | 'coming_soon' | 'error'
   export interface Agent {
   id: string
   type: AgentType
   name: string
   icon: string
   status: AgentStatus
   statusLabel: string
   }
   export const AGENTS: Agent[] = [
   {
   id: 'finalize',
   type: 'finalize',
   name: 'Requirements Finalizer',
   icon: '',

status: 'production',
statusLabel: 'Production Ready',
},
{
id: 'design',
type: 'design',
name: 'Design Generator',
icon: '',
status: 'coming_soon',
statusLabel: 'Coming Soon',
},
{
id: 'code',
type: 'code',
name: 'Code Generator',
icon: '⚡',
status: 'coming_soon',
statusLabel: 'Coming Soon',
},
{
id: 'test',
type: 'test',
name: 'Test Generator',
icon: '',
status: 'coming_soon',
statusLabel: 'Coming Soon',
},
]

3. **Create sidebar component** (components/layout/sidebar.tsx):
'use client'
import { useState } from 'react'
import { cn } from '@/lib/utils'
import { AGENTS, type Agent } from '@/types/agent'
import { Button } from '@/components/ui/button'
export function Sidebar() {
const [selectedAgent, setSelectedAgent] = useState<string>('finalize')
return (
<aside className="w-[280px] bg-bg-secondary border-r border-border-primary p-lg overflow-y-auto">
{/* Active Agents Section */}
<section className="mb-xl">

Active Agents

```
<div className="space-y-sm">
  {AGENTS.map((agent) => (
    <AgentCard
      key={agent.id}
```

```
          agent={agent}
          isActive={selectedAgent === agent.id}
          onClick={() => setSelectedAgent(agent.id)}
        />
      ))}
    </div>
  </section>

  {/* Quick Actions Section */}
  <section>
    <h3 className="text-xs uppercase tracking-wide text-text-tertiary mb-n
      Quick Actions
    </h3>

    <div className="space-y-sm">
      <Button variant="secondary" className="w-full justify-start">
        <span className="mr-sm">+</span>
        New Requirement
      </Button>
      <Button variant="secondary" className="w-full justify-start">
        <span className="mr-sm">□</span>
        View Analytics
      </Button>
      <Button variant="secondary" className="w-full justify-start">
        <span className="mr-sm">⚙</span>
        Configure Agents
      </Button>
    </div>
  </section>
</aside>
```

```
)
}
interface AgentCardProps {
agent: Agent
isActive: boolean
onClick: () => void
}
function AgentCard({ agent, isActive, onClick }: AgentCardProps) {
const agentColors = {
```

```
finalize: 'bg-agent-finalize',
design: 'bg-agent-design',
code: 'bg-agent-code',
test: 'bg-agent-test',
}
return (
<div
onClick={onClick}
className={cn(
'relative p-md rounded-md border cursor-pointer transition-all',
'hover:bg-bg-elevated hover:border-border-secondary hover:translate-x-1',
isActive
? 'bg-bg-elevated border-accent-primary'
: 'bg-bg-tertiary border-border-primary'
)}
>
{/* Active Indicator */}
{isActive && (

)}
```

```
    <div className="flex items-center gap-sm">
      {/* Agent Icon */}
      <div className={cn('w-6 h-6 rounded flex items-center justify-center te
        {agent.icon}
      </div>

      {/* Agent Info */}
      <div className="flex-1 min-w-0">
        <div className="text-sm font-medium text-text-primary truncate">
          {agent.name}
        </div>
        <div className="text-xs text-text-tertiary">
          {agent.statusLabel}
        </div>
      </div>
    </div>
  </div>
```

```
)
}
```

4. **Update layout to include sidebar:**

```
// app/layout.tsx
import { Sidebar } from '@/components/layout/sidebar'
```

```
export default function RootLayout({ children }: { children: React.ReactNode }) {
return (



{children}


)
}
```

Acceptance Criteria:

- [x] Sidebar width is 280px
- [x] Sidebar is scrollable independently
- [x] Agent cards show 3px left border when active
- [x] Agent cards translate 4px right on hover
- [x] Agent icons have correct background colors
- [x] Quick action buttons span full width
- [x] Section titles are 11px uppercase
- [x] Zustand store persists sidebar state
- [x] Mobile: sidebar collapses (responsive behavior)

Validation:

- Click agent cards and verify active state
- Hover over cards to see translation
- Refresh page and verify selected agent persists

TASK 3.3: Create Page Layouts and Routing

Priority: P1 (High)
Estimated Effort: 3 hours
Dependencies: TASK 3.1, TASK 3.2

Detailed Steps:

1. **Create content layout wrapper** (`components/layout/page-layout.tsx`):
```
import * as React from 'react'
import { cn } from '@/lib/utils'
interface PageLayoutProps {
title: string
subtitle?: string
actions?: React.ReactNode
children: React.ReactNode
className?: string
}
export function PageLayout({
title,
subtitle,
actions,
children,
className,
```

}: PageLayoutProps) {
return (

{/* Page Header */}
{title}

{subtitle && (
{subtitle}

)}

{actions &&
{actions}
}

```
    {/* Page Body */}
    <div className={cn('flex-1 p-lg overflow-y-auto', className)}>
      {children}
    </div>
  </div>
```

)
}

2. **Create dashboard page** (app/page.tsx):
import { PageLayout } from '@/components/layout/page-layout'
import { StatCard } from '@/components/ui/stat-card'
export default function DashboardPage() {
return (
<PageLayout title="Multi-Agent SDLC Dashboard" subtitle="Automated requirements finalization, design generation, and code synthesis" >
{/* Stats Grid */}

```
    {/* Placeholder for Pipeline Visualization */}
    <div className="bg-bg-secondary border border-border-primary rounde
      <h2 className="text-xl font-semibold mb-lg">Agent Pipeline Flow</h2>
      <p className="text-text-secondary">Pipeline visualization will be impl
    </div>
  </PageLayout>
```

)
}

3. **Create requirements page** (app/requirements/page.tsx):
   import { PageLayout } from '@/components/layout/page-layout'
   import { Button } from '@/components/ui/button'
   export default function RequirementsPage() {
   return (
   <PageLayout
   title="Process New Requirements"
   actions={
   ▶ Run Agent Pipeline
   }
   >
   Requirements processing UI will be implemented in Phase 2

   </PageLayout>
)
   }
4. **Create remaining placeholder pages:**
   // app/pipeline/page.tsx
   import { PageLayout } from '@/components/layout/page-layout'
   export default function PipelinePage() {
   return (
   Pipeline management will be implemented in Phase 3

)
   }
   // app/telemetry/page.tsx
   import { PageLayout } from '@/components/layout/page-layout'
   export default function TelemetryPage() {
   return (
   Telemetry and observability will be implemented in Phase 3

)
   }
   // app/settings/page.tsx
   import { PageLayout } from '@/components/layout/page-layout'
   export default function SettingsPage() {
   return (
   Settings interface will be implemented in Phase 3

)
   }

Acceptance Criteria:

- [x] All 5 routes render without errors
- [x] Navbar active state updates on route change

- [x] PageLayout provides consistent header structure
- [x] Page titles are 24px, weight 600
- [x] Subtitles are 14px, text-secondary
- [x] Dashboard displays 4 stat cards
- [x] Stat cards are responsive (1/2/4 columns)
- [x] All pages use consistent spacing (16px/24px)

Validation:

- Navigate through all routes
- Verify responsive behavior at mobile/tablet/desktop
- Check heading hierarchy (H1 for page title)

Testing & Documentation

TASK 4.1: Write Component Tests

Priority: P2 (Medium)
Estimated Effort: 4 hours
Dependencies: TASK 2.1, TASK 2.2, TASK 2.3

Detailed Steps:

1. **Install testing dependencies:**
   npm install -D vitest @testing-library/react @testing-library/jest-dom @testing-library/user-event jsdom
2. **Configure Vitest** (vitest.config.ts):
   import { defineConfig } from 'vitest/config'
   import react from '@vitejs/plugin-react'
   import path from 'path'
   export default defineConfig({
   plugins: [react()],
   test: {
   environment: 'jsdom',
   setupFiles: ['./vitest.setup.ts'],
   globals: true,
   },
   resolve: {
   alias: {
   '@': path.resolve(__dirname, './'),
   },
   },
   })
3. **Create setup file** (vitest.setup.ts):
   import '@testing-library/jest-dom'
4. **Write button component tests** (components/ui/__tests__/button.test.tsx):
   import { describe, it, expect, vi } from 'vitest'
   import { render, screen } from '@testing-library/react'
   import userEvent from '@testing-library/user-event'
   import { Button } from '../button'

describe('Button', () => {
it('renders with default variant', () => {
render(Click me)
const button = screen.getByRole('button', { name: /click me/i })
expect(button).toBeInTheDocument()
expect(button).toHaveClass('bg-accent-primary')
})
it('renders all variants correctly', () => {
const { rerender } = render(Primary)
expect(screen.getByRole('button')).toHaveClass('bg-accent-primary')

```
rerender(<Button variant="secondary">Secondary</Button>)
expect(screen.getByRole('button')).toHaveClass('bg-bg-tertiary')

rerender(<Button variant="outline">Outline</Button>)
expect(screen.getByRole('button')).toHaveClass('bg-transparent')
```

})
it('calls onClick handler', async () => {
const handleClick = vi.fn()
render(Click)

```
await userEvent.click(screen.getByRole('button'))
expect(handleClick).toHaveBeenCalledTimes(1)
```

})
it('is disabled when disabled prop is true', async () => {
const handleClick = vi.fn()
render(Disabled)

```
const button = screen.getByRole('button')
expect(button).toBeDisabled()

await userEvent.click(button)
expect(handleClick).not.toHaveBeenCalled()
```

})
})

5. **Write input component tests** (components/ui/__tests__/input.test.tsx):
   import { describe, it, expect } from 'vitest'
   import { render, screen } from '@testing-library/react'
   import userEvent from '@testing-library/user-event'
   import { Input } from '../input'
   describe('Input', () => {
   it('renders with placeholder', () => {

render( )
expect(screen.getByPlaceholderText('Enter text...')).toBeInTheDocument()
})
it('accepts user input', async () => {
render( )
const input = screen.getByRole('textbox')

```
await userEvent.type(input, 'Hello world')
expect(input).toHaveValue('Hello world')
```

})
it('applies error styling when error prop is true', () => {
render( )
const input = screen.getByRole('textbox')
expect(input).toHaveClass('border-error')
})
it('forwards ref correctly', () => {
const ref = { current: null }
render( )
expect(ref.current).toBeInstanceOf(HTMLInputElement)
})
})

6. **Add test scripts to `package.json`:**
{
"scripts": {
"test": "vitest",
"test:ui": "vitest --ui",
"test:coverage": "vitest --coverage"
}
}

Acceptance Criteria:

- [x] Vitest runs without configuration errors
- [x] Button component has 100% test coverage
- [x] Input component has 100% test coverage
- [x] All tests pass (`npm test`)
- [x] Tests use Testing Library best practices
- [x] User interactions are tested with userEvent
- [x] Accessibility attributes are verified

Validation:
npm test

All tests should pass

TASK 4.2: Create Component Documentation

Priority: P2 (Medium)
Estimated Effort: 3 hours
Dependencies: TASK 2.1, TASK 2.2, TASK 2.3

Detailed Steps:

1. **Create comprehensive README** (components/README.md):

Component Library Documentation

Overview

This component library is built with React 18, TypeScript, Tailwind CSS, and Radix UI primitives. All components follow the design system defined in the UI/UX specification document.

Design Principles

- **Accessibility First**: WCAG 2.1 AA compliant
- **Type Safety**: Full TypeScript coverage
- **Composable**: Small, focused components
- **Consistent**: Follows 8px spacing grid

Component Categories

UI Primitives (components/ui/)

- Button
- Input
- Textarea
- Label
- Card
- Tabs
- Dialog

Layout Components (components/layout/)

- Navbar
- Sidebar
- PageLayout

Feature Components (components/features/)

- StatCard
- AgentCard
- (Future: PipelineFlow, JSONViewer, LogViewer)

Usage Examples

Button

```tsx
import { Button } from '@/components/ui/button'
Submit
```

Props:

- variant: 'primary' | 'secondary' | 'outline' | 'ghost'
- size: 'sm' | 'default' | 'lg' | 'icon'
- disabled: boolean
- asChild: boolean (renders as Slot for composition)

Input

```tsx
import { Input } from '@/components/ui/input'
```

Props:

- error: boolean (applies error styling)
- All standard HTML input attributes

FormField

```tsx
import { FormField } from '@/components/ui/form-field'
import { Input } from '@/components/ui/input'
<FormField
label="Email Address"
required
error={errors.email?.message}
helperText="We'll never share your email"
/>
</FormField>
```

Styling Guidelines

Color Usage

- Primary actions: bg-accent-primary
- Surfaces: bg-bg-secondary
- Borders: border-border-primary
- Text: text-text-primary / text-text-secondary

Spacing

- Use Tailwind spacing utilities: `p-md`, `gap-lg`, `mb-sm`
- Follows 8px grid: xs=4px, sm=8px, md=16px, lg=24px, xl=32px

Responsive Design

- Mobile first approach
- Breakpoints: sm=640px, md=768px, lg=1024px, xl=1280px

Testing

All components have unit tests in `__tests__/` directories.
Run tests:
```bash
npm test
```

Accessibility

- All interactive elements are keyboard navigable
- ARIA attributes included where needed
- Focus indicators visible
- Color contrast meets WCAG AA standards

2. **Create Storybook-style component showcase** (`app/showcase/page.tsx`):
```
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Textarea } from '@/components/ui/textarea'
import { FormField } from '@/components/ui/form-field'
import { Card, CardHeader, CardBody } from '@/components/ui/card'
import { StatCard } from '@/components/ui/stat-card'
export default function ComponentShowcase() {
return (
<div className="p-lg space-y-xl max-w-7xl mx-auto">
# Component Showcase

Interactive examples of all UI components in the design system

{/* Buttons */}
<section>
  <h2 className="text-2xl font-semibold mb-md">Buttons</h2>
  <div className="space-y-md">
    <div>
      <h3 className="text-sm font-medium text-text-secondary mb-sm">Va
      <div className="flex gap-md">
        <Button variant="primary">Primary</Button>
        <Button variant="secondary">Secondary</Button>
```

```
        <Button variant="outline">Outline</Button>
        <Button variant="ghost">Ghost</Button>
      </div>
    </div>

    <div>
      <h3 className="text-sm font-medium text-text-secondary mb-sm">Si
      <div className="flex gap-md items-center">
        <Button size="sm">Small</Button>
        <Button size="default">Default</Button>
        <Button size="lg">Large</Button>
        <Button size="icon">□</Button>
      </div>
    </div>

    <div>
      <h3 className="text-sm font-medium text-text-secondary mb-sm">St
      <div className="flex gap-md">
        <Button>Normal</Button>
        <Button disabled>Disabled</Button>
      </div>
    </div>
  </div>
</section>

{/* Form Fields */}
<section>
  <h2 className="text-2xl font-semibold mb-md">Form Fields</h2>
  <div className="space-y-md max-w-lg">
    <FormField label="Text Input" required helperText="Enter your projec
      <Input placeholder="E-commerce Platform" />
    </FormField>

    <FormField label="With Error" error="This field is required">
      <Input error placeholder="Invalid input..." />
    </FormField>

    <FormField label="Textarea">
```

```
      <Textarea placeholder="Describe your requirements..." />
    </FormField>
  </div>
</section>

{/* Cards */}
<section>
  <h2 className="text-2xl font-semibold mb-md">Cards</h2>
  <div className="grid grid-cols-2 gap-md">
    <Card>
      <CardHeader>
        <h3 className="font-semibold">Basic Card</h3>
      </CardHeader>
      <CardBody>
        <p className="text-text-secondary">Card content goes here</p>
      </CardBody>
    </Card>

    <StatCard
      label="Metric Example"
      value="98.5%"
      change="↑ 5% increase"
      icon="✓"
      iconColor="bg-success/20 text-success"
    />
  </div>
</section>

{/* Color Palette */}
<section>
  <h2 className="text-2xl font-semibold mb-md">Color Palette</h2>
  <div className="grid grid-cols-6 gap-md">
    <ColorSwatch color="bg-accent-primary" label="Accent Primary" />
    <ColorSwatch color="bg-agent-finalize" label="Agent Finalize" />
    <ColorSwatch color="bg-agent-design" label="Agent Design" />
    <ColorSwatch color="bg-agent-code" label="Agent Code" />
    <ColorSwatch color="bg-agent-test" label="Agent Test" />
    <ColorSwatch color="bg-success" label="Success" />
```

```
    </div>
  </section>
</div>
```

)
}
function ColorSwatch({ color, label }: { color: string; label: string }) {
return (

<div className={${color} h-16 rounded-md mb-sm} />
{label}

)
}

Acceptance Criteria:

- [x] README covers all existing components
- [x] Usage examples are copy-pasteable
- [x] Props are documented with types
- [x] Styling guidelines are clear
- [x] Showcase page renders all components
- [x] Showcase is accessible via /showcase route
- [x] All examples are interactive

Validation:

- Navigate to /showcase route
- Copy code examples and verify they work
- Check README formatting in GitHub/IDE

Deployment & CI/CD

TASK 5.1: Configure Vercel Deployment

Priority: P1 (High)
Estimated Effort: 2 hours
Dependencies: TASK 1.1, all component tasks

Detailed Steps:

1. **Create Vercel configuration** (vercel.json):
{
"buildCommand": "npm run build",
"outputDirectory": ".next",
"devCommand": "npm run dev",
"installCommand": "npm install",
"framework": "nextjs",

```
"regions": ["iad1"]
}
```

2. **Add environment variable template** (`.env.example`):

```
# Application
NEXT_PUBLIC_APP_NAME="Multi-Agent SDLC Automation"
NEXT_PUBLIC_APP_URL="http://localhost:3000"

# API (Future)
NEXT_PUBLIC_API_URL=""
API_KEY=""

# Feature Flags (Future)
NEXT_PUBLIC_ENABLE_PIPELINE=false
NEXT_PUBLIC_ENABLE_TELEMETRY=false
```

3. **Update `.gitignore`:**

```
# Dependencies
node_modules/

# Next.js
.next/
out/

# Environment
.env
.env.local
.env.production

# Testing
coverage/

# IDE
.vscode/
.idea/

# OS
.DS_Store
Thumbs.db
```

4. **Create deployment checklist** (`DEPLOYMENT.md`):

Deployment Checklist

Pre-Deployment

- [ ] All tests passing (`npm test`)
- [ ] TypeScript compiles without errors (`npm run type-check`)
- [ ] Build succeeds locally (`npm run build`)
- [ ] Lighthouse score > 90
- [ ] No console errors in production build

Vercel Setup

1. Connect GitHub repository to Vercel
2. Configure build settings:
   - Framework: Next.js
   - Build command: `npm run build`
   - Output directory: `.next`
3. Set environment variables in Vercel dashboard
4. Enable automatic deployments on `main` branch

Post-Deployment

- [ ] Verify all routes load correctly
- [ ] Test responsive behavior on mobile/tablet
- [ ] Check Vercel Analytics for performance
- [ ] Monitor Vercel logs for errors

Rollback Procedure

1. Go to Vercel dashboard > Deployments
2. Find previous stable deployment
3. Click "..." > "Promote to Production"

5. **Add deployment scripts to `package.json`:**

```
{
"scripts": {
"dev": "next dev",
"build": "next build",
"start": "next start",
"lint": "next lint",
"type-check": "tsc --noEmit",
"test": "vitest",
"test:coverage": "vitest --coverage",
"preview": "next build && next start"
}
}
```

Acceptance Criteria:

- [x] `vercel.json` configured correctly
- [x] `.env.example` includes all required variables
- [x] `.gitignore` excludes sensitive files
- [x] Build succeeds without errors

- [x] Preview deployment works on Vercel
- [x] All routes are accessible in production
- [x] Static assets load correctly

Validation:

Local build test

npm run build
npm run start

Visit http://localhost:3000 and test all routes

Vercel CLI test (optional)

npx vercel --prod

Summary & Next Steps

Phase 1 Completion Criteria

All tasks complete when:

1. ✅ Next.js 14 project runs with TypeScript strict mode
2. ✅ Tailwind CSS configured with complete design system
3. ✅ Radix UI primitives installed and wrapped
4. ✅ Button, Input, Textarea, Label, Card components built
5. ✅ Navbar with active route highlighting
6. ✅ Collapsible sidebar with agent cards
7. ✅ All 5 routes created (Dashboard, Requirements, Pipeline, Telemetry, Settings)
8. ✅ Dashboard displays 4 stat cards
9. ✅ Component tests written and passing
10. ✅ Component documentation complete
11. ✅ Vercel deployment configured
12. ✅ Preview deployment successful

Estimated Total Effort

Task Category	Hours
Project Setup	7h
Core Components	9h
Navigation & Layout	12h
Testing & Documentation	7h
Deployment	2h
Total	**37 hours**

Recommended Sprint: 2 weeks with 2 developers (20 hours/week each)

Phase 2 Preview

Next implementation phase will include:

- Requirements Processing Form (Task 2.1)
- JSON Viewer with Syntax Highlighting (Task 2.2)
- Alert/Toast Notification System (Task 2.3)
- API Integration for Finalize Agent (Task 2.4)
- Output Tabbed Interface (Task 2.5)

Technical Debt & Future Improvements

To address in future phases:

- [ ] Implement keyboard shortcuts (Cmd+K command palette)
- [ ] Add dark/light mode toggle
- [ ] Optimize bundle size (code splitting)
- [ ] Add E2E tests with Playwright
- [ ] Implement error boundaries
- [ ] Add loading states for async operations
- [ ] Set up Sentry for error tracking
- [ ] Configure Vercel Analytics

Document End
This task breakdown provides actionable, granular steps for Phase 1 implementation. Each task includes clear acceptance criteria, estimated effort, and validation procedures to ensure quality and consistency with the design specification.