Proyecto Vite + React + TypeScript + Tailwind + Zustand + Router

📂Estructura de Carpetas y Archivos

```
mission_management/           # Carpeta raíz del proyecto
├─ .husky/                    # Hooks de git (Husky)
│  └─ pre-commit              # Ejecuta lint-staged antes de cada commit
├─ node_modules/              # Dependencias instaladas
├─ public/                    # Archivos públicos (favicon, index.html, etc.)
│  └─ index.html
├─ src/                       # Código fuente
│  ├─ components/             # Componentes reutilizables
│  │  └─ Header.tsx
│  ├─ pages/                  # Páginas de la app (React Router)
│  │  ├─ Home.tsx
│  │  └─ About.tsx
│  ├─ store/                  # Zustand stores
│  │  └─ useCounterStore.ts
│  ├─ App.tsx                 # Componente principal con Routes
│  ├─ main.tsx                # Entrada de React, importa CSS y renderiza App
│  └─ index.css               # Tailwind imports
├─ tailwind.config.js         # Configuración de Tailwind CSS
├─ postcss.config.js          # Configuración de PostCSS
├─ tsconfig.json              # Configuración de TypeScript
├─ package.json               # Scripts y dependencias
└─ vite.config.ts             # Configuración de Vite
```

⚙️Scripts Principales (package.json)

```
"scripts": {
  "dev": "vite",                // Levanta el servidor de desarrollo
  "build": "vite build",        // Build optimizada para producción
  "preview": "vite preview",    // Previsualizar build
  "typecheck": "tsc --noEmit", // Verificar tipos TypeScript
  "lint": "eslint \"src/**/*.{ts,tsx,js,jsx}\"", // Revisar errores de código
  "format": "prettier --write \"src/**/*.{ts,tsx,js,jsx,json,css,md}\"", //
Formatear código
  "test": "vitest",             // Ejecutar tests
  "test:coverage": "vitest run --coverage" // Cobertura de tests
}
```

🧰Componentes y Funcionalidades

Componente/ Carpeta	Propósito
src/components/	Componentes reutilizables (botones, formularios, headers, etc.)
src/pages/	Páginas de la aplicación para React Router (Home, About, etc.)
src/store/	Zustand store(s) para manejo de estado global
App.tsx	Componente raíz con React Router configurado
main.tsx	Entrada de React, renderiza App y carga CSS
index.css	Importa Tailwind (`@tailwind base; @tailwind components; @tailwind utilities;`)
.husky/pre-commit	Hook de Git para ejecutar `lint-staged` antes de cada commit
tailwind.config.js	Configuración de rutas y plugins Tailwind
postcss.config.js	Configuración de PostCSS y autoprefixer

🛠️Comandos de Desarrollo

Comando	Propósito
`npm run dev`	Levanta servidor de desarrollo con hot reload
`npm run build`	Genera build optimizada para producción
`npm run preview`	Previsualiza la build final
`npm run lint`	Revisa errores de código con ESLint
`npm run format`	Formatea código automáticamente con Prettier
`npm run test`	Ejecuta tests unitarios con Vitest
`npm run test:coverage`	Genera reporte de cobertura de tests
`npm run typecheck`	Verifica tipos TypeScript sin generar archivos

🕰️Buenas Prácticas

- Mantener `.husky/pre-commit` para lint y formateo automático antes de commits.
- Usar ESLint + Prettier para consistencia de código.
- Escribir tests para componentes críticos y lógica de estado.
- Usar rutas y alias (`@/`) para imports limpios.
- Mantener `tailwind.config.js` actualizado con las rutas de los archivos donde se usan clases Tailwind.

- Ejecutar `npm run dev` y `npm run build` regularmente para detectar errores tempranos.
- Documentar la estructura y scripts en README.md para facilidad del equipo.

```
# Ejemplo de flujo típico
npm run lint       # Revisar errores
npm run format     # Formatear código
npm run test       # Ejecutar tests
npm run dev        # Desarrollar en hot reload
git add .
git commit -m "feat: nueva funcionalidad"
```