

Plan de Refactor Completo - Control Tower Dashboard



Resumen Ejecutivo

Dado el estado crítico del proyecto actual (riesgo 8/10), se propone un **REFACTOR COMPLETO** en lugar de mejoras graduales. Este enfoque permitirá:

- ✓ Eliminar toda la deuda técnica de una vez
- ✓ Implementar mejores prácticas desde el inicio
- ✓ Crear una base sólida y escalable
- ✓ Reducir tiempo de desarrollo a largo plazo
- ✓ Mejorar la experiencia del desarrollador



Objetivos del Refactor

- Arquitectura Limpia y Escalable
- Type Safety Completo con TypeScript
- Performance Optimizado desde el Inicio
- Testing Integrado
- Developer Experience de Primera Clase
- Documentación Completa



Nueva Arquitectura Propuesta

Stack Tecnológico Recomendado (Con Restricciones de Infraestructura)

Core:

- Vite 5.1+ (Build tool - REQUERIDO)
- TypeScript 5.3+ (strict mode)
- React 18.2+
- React Router 6.22+

Estado:

- Redux Toolkit 2.2+ (REQUERIDO - pero bien estructurado)
- RTK Query (para server state)
- Redux Persist (para persistencia)

Estilos:

- Tailwind CSS 3.4+
- CVA (Class Variance Authority)
- Radix UI (componentes headless)

Testing:

- Vitest
- React Testing Library
- Playwright (E2E)

Calidad:

- ESLint (con reglas estrictas)
- Prettier
- Husky + lint-staged
- Commitlint

Performance:

- @loadable/component (code splitting)
- React Query (cache management)
- Comlink (web workers)

Estructura de Carpetas Propuesta

```
src/
├── app/                                # Core de la aplicación
│   ├── router/                        # Configuración de rutas
│   │   ├── routes.tsx
│   │   ├── PrivateRoute.tsx
│   │   └── LazyRoutes.tsx
│   ├── store/                        # Redux store configuration
│   │   ├── index.ts
│   │   ├── rootReducer.ts
│   │   └── middleware.ts
│   └── providers/                    # App providers
│       ├── AppProvider.tsx
│       └── ThemeProvider.tsx
├── features/                         # Funcionalidades por dominio (feature-sliced)
│   ├── auth/
│   │   ├── components/
│   │   ├── hooks/
│   │   ├── services/
│   │   ├── store/                    # Redux slice específico
│   │   │   ├── authSlice.ts
│   │   │   └── authApi.ts (RTK Query)
│   │   └── types/
│   └── dashboard/
```

```
| | | | components/
| | | | | hooks/
| | | | | services/
| | | | | store/
| | | | | | dashboardSlice.ts
| | | | | | dashboardApi.ts
| | | | | types/
| | | monitoring/
| | | planning/
|
| | shared/ # Código compartido
| | | components/
| | | | ui/ # Componentes base (Button, Card, etc)
| | | | charts/ # Componentes de gráficos
| | | | | layouts/ # Layout components
| | | | hooks/
| | | | lib/ # Utilidades y configuraciones
| | | | services/ # Servicios compartidos
| | | | types/ # Tipos globales
|
| | pages/ # Páginas de la aplicación
| | | Portal.tsx
| | | Dashboard.tsx
| | | Monitoring.tsx
| | | [feature]/
|
| | styles/
| | | globals.css # Solo estilos globales de Tailwind
|
| | tests/ # Tests globales y utilidades
| | | e2e/
| | | integration/
| | | utils/
```



Proceso de Migración

Fase 0: Preparación (1 semana)

```
graph LR
  A[Análisis Completo] --> B[Backup Código Actual]
  B --> C[Setup Nuevo Proyecto]
  C --> D[Configuración Base]
  D --> E[CI/CD Pipeline]
```

Tareas:

- [] Crear nuevo repositorio/branch para refactor
- [] Setup Vite con TypeScript strict
- [] Configurar Redux Toolkit con estructura modular
- [] Configurar ESLint, Prettier, Husky
- [] Setup testing framework (Vitest)
- [] Configurar CI/CD pipeline
- [] Crear documentación inicial

Fase 1: Core Infrastructure (2 semanas)

```
graph TB
  A[Authentication] --> B[Authorization]
  B --> C[API Layer]
  C --> D[State Management]
  D --> E[Error Handling]
  E --> F[Logging System]
```

Implementación:

1. Sistema de Autenticación/Autorización

```
// src/features/auth/services/auth.service.ts
export class AuthService {
  private static instance: AuthService;

  static getInstance(): AuthService {
    if (!this.instance) {
      this.instance = new AuthService();
    }
    return this.instance;
  }

  async login(credentials: LoginCredentials): Promise<User> {
    // Implementación con manejo de errores robusto
  }

  async logout(): Promise<void> {
    // Limpieza completa de estado
  }

  async refreshToken(): Promise<Token> {
    // Auto-refresh con retry logic
  }
}
```

```
}  
}
```

2. API Layer con RTK Query

```
// src/shared/services/api/baseApi.ts  
import { createApi, fetchBaseQuery, retry } from '@reduxjs/toolkit/query/react';  
import type { RootState } from '@app/store';  
  
const baseQuery = fetchBaseQuery({  
  baseUrl: import.meta.env.VITE_API_URL,  
  prepareHeaders: (headers, { getState }) => {  
    const token = (getState() as RootState).auth.token;  
    if (token) {  
      headers.set('authorization', `Bearer ${token}`);  
    }  
    return headers;  
  },  
});  
  
const baseQueryWithRetry = retry(baseQuery, { maxRetries: 3 });  
  
export const baseApi = createApi({  
  reducerPath: 'api',  
  baseQuery: baseQueryWithRetry,  
  tagTypes: ['Dashboard', 'Monitoring', 'Planning'],  
  endpoints: () => ({}),  
});  
  
// src/features/dashboard/store/dashboardApi.ts  
import { baseApi } from '@shared/services/api/baseApi';  
  
export const dashboardApi = baseApi.injectEndpoints({  
  endpoints: (builder) => ({  
    getDashboardData: builder.query({  
      query: (filters) => ({  
        url: '/dashboard',  
        params: filters,  
      }),  
      providesTags: ['Dashboard'],  
      transformResponse: (response) => transformDashboardData(response),  
      keepUnusedDataFor: 5 * 60, // 5 minutos  
    }),  
  }),  
});  
  
export const { useGetDashboardDataQuery } = dashboardApi;
```

3. State Management con Redux Toolkit (Bien Estructurado)

```
// src/features/dashboard/store/dashboardSlice.ts
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { dashboardApi } from '../dashboardApi';

interface DashboardState {
  filters: FilterState;
  selectedArea: string | null;
  ui: {
    sidebarOpen: boolean;
    activeTab: string;
  };
}

const initialState: DashboardState = {
  filters: defaultFilters,
  selectedArea: null,
  ui: {
    sidebarOpen: true,
    activeTab: 'overview',
  },
};

export const dashboardSlice = createSlice({
  name: 'dashboard',
  initialState,
  reducers: {
    setFilters: (state, action: PayloadAction<Partial<FilterState>>) => {
      state.filters = { ...state.filters, ...action.payload };
    },
    setSelectedArea: (state, action: PayloadAction<string | null>) => {
      state.selectedArea = action.payload;
    },
    toggleSidebar: (state) => {
      state.ui.sidebarOpen = !state.ui.sidebarOpen;
    },
    reset: () => initialState,
  },
  extraReducers: (builder) => {
    // Sincronizar con API responses si es necesario
    builder.addMatcher(
      dashboardApi.endpoints.getDashboardData.matchFulfilled,
      (state, action) => {
        // Update state based on API response if needed
      }
    );
  },
});

export const { setFilters, setSelectedArea, toggleSidebar, reset } = dashboardSlice.actions;

// Selectors memoizados con createSelector
import { createSelector } from '@reduxjs/toolkit';
import type { RootState } from '@app/store';

export const selectDashboard = (state: RootState) => state.dashboard;
export const selectFilters = createSelector(
  selectDashboard,
  (dashboard) => dashboard.filters
);
```

```

);
export const selectFilteredData = createSelector(
  [selectDashboard, (state: RootState, data: any[]) => data],
  (dashboard, data) => applyFilters(data, dashboard.filters)
);

```

4. Redux Store Modular

```

// src/app/store/index.ts
import { configureStore } from '@reduxjs/toolkit';
import { setupListeners } from '@reduxjs/toolkit/query';
import {
  persistStore,
  persistReducer,
  FLUSH,
  REHYDRATE,
  PAUSE,
  PERSIST,
  PURGE,
  REGISTER,
} from 'redux-persist';
import storage from 'redux-persist/lib/storage';

import { rootReducer } from '../rootReducer';
import { baseApi } from '@shared/services/api/baseApi';

const persistConfig = {
  key: 'root',
  storage,
  whitelist: ['auth', 'preferences'], // Solo persistir lo necesario
};

const persistedReducer = persistReducer(persistConfig, rootReducer);

export const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
      },
    })
    .concat(baseApi.middleware)
    .concat(customMiddleware),
  devTools: import.meta.env.DEV,
});

export const persistor = persistStore(store);

setupListeners(store.dispatch);

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

// src/app/store/rootReducer.ts

```

```
import { combineReducers } from '@reduxjs/toolkit';

import { baseApi } from '@shared/services/api/baseApi';

// Feature slices
import { authSlice } from '@features/auth/store/authSlice';
import { dashboardSlice } from '@features/dashboard/store/dashboardSlice';
import { monitoringSlice } from '@features/monitoring/store/monitoringSlice';

export const rootReducer = combineReducers({
  // API
  [baseApi.reducerPath]: baseApi.reducer,

  // Features
  auth: authSlice.reducer,
  dashboard: dashboardSlice.reducer,
  monitoring: monitoringSlice.reducer,
});
```

Fase 2: Componentes UI Base (2 semanas)

Sistema de Diseño con Radix UI + CVA

```
// src/shared/components/ui/button.tsx
import { cva, type VariantProps } from 'class-variance-authority';
import { forwardRef } from 'react';

const buttonVariants = cva(
  'inline-flex items-center justify-center rounded-md font-medium transition-colors focus-visible:outline-none focus-visible:ring-2 disabled:pointer-events-none disabled:opacity-50',
  {
    variants: {
      variant: {
        default: 'bg-primary text-primary-foreground hover:bg-primary/90',
        destructive: 'bg-destructive text-destructive-foreground hover:bg-destructive/90',
        outline: 'border border-input bg-background hover:bg-accent hover:text-accent-foreground',
        secondary: 'bg-secondary text-secondary-foreground hover:bg-secondary/80',
        ghost: 'hover:bg-accent hover:text-accent-foreground',
        link: 'text-primary underline-offset-4 hover:underline',
      },
      size: {
        default: 'h-10 px-4 py-2',
        sm: 'h-9 rounded-md px-3',
        lg: 'h-11 rounded-md px-8',
        icon: 'h-10 w-10',
      },
    },
    defaultVariants: {
      variant: 'default',
      size: 'default',
    },
  }
);
```



```

);

export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement>,
    VariantProps<typeof buttonVariants> {}

export const Button = forwardRef<HTMLButtonElement, ButtonProps>(
  ({ className, variant, size, ...props }, ref) => {
    return (
      <button
        className={cn(buttonVariants({ variant, size, className }))}
        ref={ref}
        {...props}
      />
    );
  }
);

Button.displayName = 'Button';

```

Componentes de Gráficos Optimizados

```

// src/shared/components/charts/line-chart.tsx
import { lazy, Suspense, memo } from 'react';
import loadable from '@loadable/component';

// Lazy load con Vite para optimización de bundle
const Plot = loadable(() => import('react-plotly.js'), {
  fallback: <ChartSkeleton />
});

export const LineChart = memo(({ data, config }: LineChartProps) => {
  return (
    <Suspense fallback={<ChartSkeleton />}>
      <Plot
        data={data}
        layout={config.layout}
        config={config.options}
        style={{ width: '100%', height: '100%' }}
      />
    </Suspense>
  );
});

// Alternativa con dynamic imports de Vite
export const DynamicChart = lazy(() =>
  import('./chart-implementations').then(module => ({
    default: module.LineChart
  })))
);

```

Router Configuration con Code Splitting

```
// src/app/router/routes.tsx
import { lazy } from 'react';
import { RouteObject } from 'react-router-dom';

// Lazy loading de páginas con Vite
const Portal = lazy(() => import('@pages/Portal'));
const Dashboard = lazy(() => import('@pages/Dashboard'));
const Monitoring = lazy(() => import('@pages/Monitoring'));
const Planning = lazy(() => import('@pages/Planning'));
const Reliability = lazy(() => import('@pages/Reliability'));

export const routes: RouteObject[] = [
  {
    path: '/',
    element: <RootLayout />,
    children: [
      {
        index: true,
        element: <Navigate to="/portal" replace />,
      },
      {
        path: 'portal',
        element: (
          <Suspense fallback={<PageLoader />}>
            <Portal />
          </Suspense>
        ),
      },
      {
        path: 'dashboard',
        element: (
          <PrivateRoute>
            <Suspense fallback={<PageLoader />}>
              <Dashboard />
            </Suspense>
          </PrivateRoute>
        ),
      },
      // ... más rutas
    ],
  },
];

// src/app/router/AppRouter.tsx
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
import { routes } from './routes';

const router = createBrowserRouter(routes, {
  basename: import.meta.env.BASE_URL,
});

export function AppRouter() {
```

```
    return <RouterProvider router={router} />;  
  }  
}
```

Fase 3: Migración de Features (3-4 semanas)

Estrategia de Migración por Módulos

```
graph LR  
  A[Portal] --> B[Dashboard]  
  B --> C[Monitoring]  
  C --> D[Planning]  
  D --> E[Reliability]  
  E --> F[General Stop]
```

Ejemplo: Migración del Dashboard con Redux Toolkit

```
// src/pages/Dashboard.tsx  
import { Suspense } from 'react';  
import { DashboardSkeleton } from '@features/dashboard/components/dashboard-skeleton';  
import { DashboardContent } from '@features/dashboard/components/dashboard-content';  
import { ErrorBoundary } from '@shared/components/error-boundary';  
  
export default function DashboardPage() {  
  return (  
    <ErrorBoundary fallback=<DashboardError />>  
      <Suspense fallback=<DashboardSkeleton />>  
        <DashboardContent />  
      </Suspense>  
    </ErrorBoundary>  
  );  
}  
  
// src/features/dashboard/components/dashboard-content.tsx  
import { useAppSelector, useAppDispatch } from '@app/store/hooks';  
import { selectFilters, setFilters } from '../../store/dashboardSlice';  
import { useGetDashboardDataQuery } from '../../store/dashboardApi';  
  
export function DashboardContent() {  
  const dispatch = useAppDispatch();  
  const filters = useAppSelector(selectFilters);  
  
  const { data, isLoading, error } = useGetDashboardDataQuery(filters, {  
    pollingInterval: 60000, // Refresh cada minuto  
    refetchOnMountOrArgChange: true,  
  });  
});
```

```

    if (error) throw error; // Manejado por ErrorBoundary

    const handleFilterChange = (newFilters: Partial<FilterState>) => {
      dispatch(setFilters(newFilters));
    };

    return (
      <div className="container mx-auto p-6">
        <DashboardHeader />
        <DashboardFilters
          filters={filters}
          onChange={handleFilterChange}
        />
        <DashboardMetrics data={data} loading={isLoading} />
        <DashboardCharts data={data} loading={isLoading} />
      </div>
    );
  }
}

// src/app/store/hooks.ts
import { useDispatch, useSelector, TypedUseSelectorHook } from 'react-redux';
import type { RootState, AppDispatch } from './index';

export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;

```

Fase 4: Testing & Calidad (1 semana)

Testing Strategy

```

// src/features/dashboard/__tests__/dashboard.test.tsx
import { render, screen, waitFor } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import DashboardPage from '../components/dashboard-page';

describe('Dashboard', () => {
  const queryClient = new QueryClient({
    defaultOptions: { queries: { retry: false } }
  });

  it('should render dashboard with data', async () => {
    render(
      <QueryClientProvider client={queryClient}>
        <DashboardPage />
      </QueryClientProvider>
    );

    await waitFor(() => {
      expect(screen.getByText('Dashboard')).toBeInTheDocument();
    });
  });
});

```

```
// More assertions...

});

it('should filter data when filter is applied', async () => {
  const user = userEvent.setup();
  // Test implementation
});
});
```

E2E Testing

```
// tests/e2e/dashboard.spec.ts
import { test, expect } from '@playwright/test';

test.describe('Dashboard Flow', () => {
  test('should load and display dashboard data', async ({ page }) => {
    await page.goto('/dashboard');

    // Wait for data to load
    await page.waitForSelector('[data-testid="dashboard-metrics"]');

    // Verify key elements
    await expect(page.locator('h1')).toContainText('Dashboard');
    await expect(page.locator('[data-testid="metric-card"]')).toHaveCount(4);

    // Test filtering
    await page.click('[data-testid="filter-button"]');
    await page.selectOption('[data-testid="area-select"]', 'production');
    await expect(page.locator('[data-testid="filtered-results"]')).toBeVisible();
  });
});
```

Fase 5: Optimización & Performance (1 semana)

Bundle Optimization con Vite

```
// vite.config.ts
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import { visualizer } from 'rollup-plugin-visualizer';
import { compression } from 'vite-plugin-compression2';
import tsconfigPaths from 'vite-tsconfig-paths';

export default defineConfig({
  plugins: [
    react(),
    visualizer(),
    compression(),
    tsconfigPaths(),
  ],
});
```

```

    tsconfigPaths(),
    compression({
      algorithm: 'gzip',
      exclude: [/\. (br)$ /, /\. (gz)$ /],
    }),
    visualizer({
      template: 'treemap',
      open: true,
      gzipSize: true,
      brotliSize: true,
    }),
  ],

  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          'vendor-react': ['react', 'react-dom', 'react-router-dom'],
          'vendor-redux': ['@reduxjs/toolkit', 'react-redux', 'redux-persist'],
          'vendor-ui': ['@mui/material', '@emotion/react', '@emotion/styled'],
          'vendor-charts': ['plotly.js-dist', 'react-plotly.js'],
          'vendor-utils': ['axios', 'date-fns', 'lodash-es'],
        },
      },
    },
    chunkSizeWarningLimit: 1000,
    sourcemap: true,
    minify: 'terser',
    terserOptions: {
      compress: {
        drop_console: true,
        drop_debugger: true,
      },
    },
  },

  optimizeDeps: {
    include: ['react', 'react-dom', '@reduxjs/toolkit'],
    exclude: ['@vite/client', '@vite/env'],
  },
});

```

Main App Setup con Vite + Redux

```

// src/main.tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import { PersistGate } from 'redux-persist/integration/react';
import { store, persistor } from '@app/store';
import { AppProvider } from '@app/providers/AppProvider';
import { AppRouter } from '@app/router/AppRouter';
import '@styles/globals.css';

```

```
// Performance monitoring
if (import.meta.env.PROD) {
  import('@shared/lib/monitoring/performance').then(({ initPerformanceMonitoring }) => {
    initPerformanceMonitoring();
  });
}

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <Provider store={store}>
      <PersistGate loading={<AppLoader />} persistor={persistor}>
        <AppProvider>
          <AppRouter />
        </AppProvider>
      </PersistGate>
    </Provider>
  </React.StrictMode>
);
```

Performance Monitoring

```
// src/shared/lib/monitoring/performance.ts
import { getCLS, getFID, getFCP, getLCP, getTTFB } from 'web-vitals';

export function initPerformanceMonitoring() {
  if (typeof window !== 'undefined') {
    getCLS(console.log);
    getFID(console.log);
    getFCP(console.log);
    getLCP(console.log);
    getTTFB(console.log);
  }
}
```

Comparación: Refactor Gradual vs Completo

Aspecto	Refactor Gradual	Refactor Completo
Tiempo Total	12-16 semanas	8-10 semanas
Riesgo	Alto (mezcla código viejo/nuevo)	Bajo (base limpia)
Esfuerzo	Mayor (mantener dos códigos)	Menor (enfoque único)
Calidad Final	Comprometida	Óptima

Aspecto	Refactor Gradual	Refactor Completo
Deuda Técnica	Persiste parcialmente	Eliminada completamente
Testing	Difícil de implementar	Integrado desde inicio
Team Morale	Baja (frustración continua)	Alta (proyecto nuevo)

Plan de Implementación

Semana 1: Setup y Core

- ☐ Crear nuevo proyecto Vite + React
- ☐ Configurar TypeScript strict
- ☐ Setup Redux Toolkit con estructura modular
- ☐ Configurar RTK Query para API calls
- ☐ Setup testing framework (Vitest)
- ☐ CI/CD pipeline

Semanas 2-3: Infrastructure

- ☐ API client con interceptors
- ☐ State management setup
- ☐ Error boundaries
- ☐ Logging system
- ☐ Sistema de componentes base

Semanas 4-6: Migración Features

- ☐ Portal page
- ☐ Dashboard completo
- ☐ Monitoring module
- ☐ Planning module
- ☐ Reliability module

Semana 7: Testing

- ☐ Unit tests (>80% coverage)
- ☐ Integration tests

- ☐ E2E tests críticos
- ☐ Performance testing

Semana 8: Optimización

- ☐ Bundle optimization
- ☐ Performance tuning
- ☐ SEO y accesibilidad
- ☐ Documentation

Semana 9-10: Deployment

- ☐ Staging deployment
 - ☐ User acceptance testing
 - ☐ Production deployment
 - ☐ Monitoring setup
 - ☐ Handover y training
-

Beneficios Técnicos del Refactor Completo

Inversión de Tiempo

- **Desarrollo:** 8-10 semanas con equipo dedicado
- **Testing:** 1 semana adicional
- **Deployment:** 1 semana

Mejoras Esperadas (Año 1)

- **Reducción bugs:** -70%
- **Velocidad desarrollo:** +40%
- **Mantenimiento:** -50% tiempo requerido
- **Onboarding:** -60% tiempo de incorporación
- **Performance:** +50% mejora en métricas

Retorno Técnico: Productividad 2.5x en el primer año

Checklist de Decisión

¿Cuándo hacer Refactor Completo?

- [x] Deuda técnica > 60% del código
- [x] Múltiples anti-patterns críticos
- [x] Performance issues sistemáticos
- [x] Dificultad extrema para nuevas features
- [x] Bugs recurrentes en producción
- [x] Team frustrado con codebase
- [x] Tiempo de mantenimiento excede tiempo de desarrollo nuevo
- [x] Ventana de tiempo disponible

Resultado: 8/8  PROCEDER CON REFACTOR COMPLETO

Métricas de Éxito

Technical KPIs

Métrica	Actual	Target	Mejora
Bundle Size	~5MB	<1.5MB	-70%
First Load	>5s	<2s	-60%
Lighthouse Score	~50	>90	+80%
Type Coverage	0%	100%	+100%
Test Coverage	0%	>80%	+80%

Business KPIs

Métrica	Actual	Target	Mejora
Bugs/Sprint	15-20	<5	-75%
Feature Velocity	2/sprint	5/sprint	+150%

Métrica	Actual	Target	Mejora
Deploy Frequency	Weekly	Daily	+500%
MTTR	4h	<1h	-75%

Conclusión

Un **REFACTOR COMPLETO** es la decisión correcta para este proyecto porque:

- El tiempo de mantener el código actual supera el tiempo del refactor
- La deuda técnica está en nivel crítico (8/10)
- Un enfoque gradual prolongaría el sufrimiento
- Las mejoras en productividad son claras y medibles (2.5x año 1)
- Mejorará significativamente la moral del equipo

Recomendación Final:

PROCEDER INMEDIATAMENTE con el Refactor Completo usando la arquitectura propuesta con Next.js 14+, TypeScript strict, y las mejores prácticas modernas.

El equipo debería: 1. Obtener aprobación de stakeholders 2. Asignar recursos dedicados (no parciales) 3. Establecer fecha de freeze del código actual 4. Comenzar con el nuevo proyecto en paralelo 5. Planificar migración gradual de usuarios

Documento preparado para decisión ejecutiva Fecha: 2025-01-07 Próximos pasos: Aprobación y kick-off meeting