GUÍA DE DESARROLLO:

APLICACIÓN DE MAPAS DEL CAMPUS INGENIERÍA

Documento Técnico para Estimación de Tiempos y Planificación de Desarrollo

Contenido

INTRODUCCIÓN Y CONTEXTO DEL PROYECTO 3

1.1 Alcance del Proyecto 4

2. ANÁLISIS DE COMPLEJIDAD Y DESAFÍOS TÉCNICOS..... 5

INTRODUCCIÓN Y CONTEXTO DEL PROYECTO

El desarrollo de una aplicación web de mapas interactivos para campus universitarios representa un proyecto de **complejidad media-alta** que requiere conocimientos especializados en múltiples áreas de la ingeniería de software. Contrario a la percepción común de que "es solo mostrar un mapa", este tipo de aplicaciones involucra desafíos técnicos significativos que van desde el procesamiento de archivos pesados hasta la implementación de algoritmos de teoría de grafos.

Este documento tiene como objetivo proporcionar una **estimación realista** de los tiempos de desarrollo, justificar las tecnologías seleccionadas y explicar por qué proyectos de esta naturaleza no pueden completarse en plazos excesivamente cortos sin comprometer la calidad o funcionalidad.

1.1 Alcance del Proyecto

La aplicación debe cumplir con los siguientes requisitos funcionales:

- Visualización interactiva de mapas del campus (planos arquitectónicos en formato DWG/SVG)
- Sistema de navegación zoom fluido
- Marcación de puntos de interés (edificios, salones, servicios)
- Cálculo y visualización de rutas óptimas entre ubicaciones
- Interfaz responsiva compatible con dispositivos móviles y escritorio
- Rendimiento optimizado para archivos SVG de gran tamaño (20-35 MB)

2. ANÁLISIS DE COMPLEJIDAD Y DESAFÍOS TÉCNICOS

2.1 Problema Principal: Archivos SVG Masivos de AutoCAD

Contexto técnico: Los planos arquitectónicos exportados desde AutoCAD DWG a formato SVG contienen información excesivamente detallada que no es necesaria para una aplicación web:

- **Metadata de CAD:** Información de capas, bloques, estilos de línea
- **Precisión innecesaria:** Coordenadas con 8-12 decimales
- **Elementos invisibles:** Guías, construcciones auxiliares, elementos ocultos
- **Paths complejos:** Curvas Bézier con cientos de puntos de control

Impacto en rendimiento:

Archivo original DWG → SVG: ~35 MB

Tiempo de carga inicial: 8-15 segundos

Consumo de memoria: 400-600 MB

Rendering inicial: 3-5 segundos (bloqueo del navegador)

Justificación de tiempo de optimización: 1 semana

- Investigación de librerías de optimización SVG
- Desarrollo de scripts personalizados de limpieza
- Pruebas iterativas de reducción sin pérdida visual
- Validación de compatibilidad cross-browser

2.2 Desafío de Rendimiento: Pan y Zoom en SVG Grande

Problema técnico: Las transformaciones SVG (translate, scale) en documentos grandes causan recálculos costosos del DOM, generando lag perceptible (>100ms) que arruina la experiencia de usuario.

Solución implementada: RequestAnimationFrame Throttling

Por qué se usa: requestAnimationFrame sincroniza las actualizaciones con el ciclo de repintado del navegador (60 FPS), evitando actualizaciones redundantes y garantizando fluidez.

Tiempo requerido: 2-3 días

- Implementación de diferentes estrategias (CSS transforms vs SVG transforms)
- Profiling de rendimiento con Chrome DevTools
- Optimización de event listeners
- Testing en dispositivos de gama baja

2.3 Desafío de Arquitectura: Sistema de Capas SVG

Problema de diseño: Mezclar el mapa base, nodos interactivos y rutas en un único contexto SVG genera conflictos de z-index, dificulta el manejo de eventos y complica las actualizaciones parciales.

Solución: Sistema de capas separadas

Justificación técnica:

- **Separación de responsabilidades:** Cada capa puede ser manipulada independientemente
- **Optimización selectiva:** Solo re-renderizar capas modificadas
- **Event delegation:** Facilita el manejo de clicks en nodos sin contaminar el mapa base
- **Control de visibilidad:** Toggle de capas sin afectar el DOM completo

Tiempo de implementación: 2-3 días

- Diseño de la arquitectura de capas
- Sistema de carga asíncrona de SVG externos
- Normalización de viewBox entre capas importadas
- Testing de interoperabilidad
-

3. ALGORITMOS Y ESTRUCTURAS DE DATOS

3.1 Algoritmo de Dijkstra para Pathfinding

*¿Por qué Dijkstra y no A?**

Aunque A* es más eficiente en términos de nodos explorados, en el contexto de mapas de campus:

1. **Grafos pequeños:** Campus típico tiene 50-200 nodos (edificios/intersecciones)
2. **Sin heurística confiable:** La distancia euclidiana no refleja rutas reales (obstáculos, pasillos)
3. **Simplicidad de implementación:** Dijkstra es más robusto y predecible
4. **Diferencia de performance irrelevante:** <50ms en ambos casos para grafos de este tamaño

Implementación optimizada:

Complejidad temporal: $O(n^2 + m)$ donde n = nodos, m = aristas **Complejidad espacial:** O(n)

Optimización con heap binario (opcional para proyectos futuros): Usando un min-heap, se podría reducir a $O((n + m) \log n)$, pero la implementación aumenta la complejidad del código sin beneficio práctico en grafos pequeños.

Tiempo de desarrollo: 4-6 días

- Diseño de estructura de datos del grafo (JSON)
- Implementación del algoritmo
- Testing con casos edge (grafos desconectados, nodos inexistentes)
- Fallback a línea recta cuando no hay grafo
- Visualización de la ruta calculada

3.2 Detección Automática de Nodos

Problema: Los planos de AutoCAD no incluyen metadata semántica sobre qué elementos son "lugares de interés".

Solución: Heurística de detección basada en geometría

Justificación de umbrales:

- Elementos < 100px^2: Probablemente decoración, iconos pequeños
- Elementos > 20,000px^2: Probablemente edificios completos, no puntos específicos
- Rango intermedio: Puertas, marcadores, salones específicos

Tiempo de desarrollo: 3-4 días

- Experimentación para encontrar umbrales óptimos
- Sistema de confirmación manual (UI para aprobar/rechazar nodos)
- Asignación automática de IDs únicos
- Integración con sistema de selección

4. TECNOLOGÍAS SELECCIONADAS Y JUSTIFICACIÓN

4.1 Vanilla JavaScript (ES6+) vs Frameworks

Decisión: JavaScript Modular Vanilla (ESM)

Justificación:

Aspecto	**Vanilla JS**	**React/Vue/Angular**
Tamaño del bundle	~15-30 KB	150-300 KB (framework + deps)
Tiempo de carga inicial	<500ms	1-3s (parse + hydration)
Complejidad del proyecto	Baja-Media	Justificada solo en apps complejas
Performance SVG	Excelente (DOM directo)	Overhead de virtual DOM
Curva de aprendizaje	Menor	Requiere conocimiento del framework

Para proyectos que requieren:

- Manipulación intensiva del DOM (SVG)
- Performance crítico en dispositivos móviles
- Bundle size mínimo

- Sin necesidad de estado complejo global

Vanilla JS con módulos ES6 es superior.

Cuándo considerar un framework:

- Sistema de usuarios con autenticación
- Dashboard administrativo complejo
- Múltiples vistas con routing
- Estado global compartido entre muchos componentes

4.2 SVG vs Canvas vs WebGL

Decisión: SVG (Scalable Vector Graphics)

Comparativa técnica:

SVG:

- ✅ Escalado sin pérdida de calidad
- ✅ Accesibilidad (screen readers)
- ✅ Inspect element (facilita debugging)
- ✅ CSS styling nativo
- ✅ Event handling por elemento
- ❌ Performance degrada con >10,000 elementos

Canvas 2D:

- ✅ Excelente para >10,000 elementos
- ✅ Rendering más rápido
- ❌ Requiere redibujado manual completo
- ❌ No escalable (rasterizado)
- ❌ Sin accesibilidad nativa
- ❌ Hit detection manual (complejidad alta)

WebGL:

- ✅ Máximo performance (GPU)
- ❌ Curva de aprendizaje muy alta
- ❌ Overkill para mapas 2D estáticos

- ❌ Requiere shaders personalizados

Conclusión: SVG es óptimo para mapas de campus (50-500 elementos visibles), con la ventaja de poder hacer zoom infinito sin pérdida de calidad.

4.3 Módulos ES6 vs Script Tags Tradicionales

Decisión: Módulos ES6 (type="module")

Ventajas:

- **Namespace limpio:** No contaminación del objeto global window
- **Lazy loading:** Imports dinámicos para features opcionales
- **Tree shaking:** Bundlers pueden eliminar código no usado
- **Mantenibilidad:** Separación clara de responsabilidades

Tiempo de refactoring: 1-2 días

- Convertir código monolítico a módulos
- Configurar bundler (opcional: Vite/Rollup)
- Testing de imports/exports

---

5. PLANIFICACIÓN DE TIEMPO DETALLADA

5.1 Fase 1: Preparación y Análisis (1 semana)

Días 1-2: Análisis de requisitos

- Entrevistas con stakeholders (¿qué edificios son prioritarios?)
- Análisis del archivo DWG original
- Identificación de capas relevantes en AutoCAD
- Definición de casos de uso principales

Días 3-4: Setup del proyecto

- Estructura de carpetas
- Control de versiones (Git)
- Configuración de entorno de desarrollo
- Selección de herramientas (linters, formatters)

Día 5: Prototipo técnico

- Proof of concept: cargar SVG básico

- Test de rendimiento inicial
- Validación de viabilidad técnica

Entregable: Documento de especificaciones técnicas + Prototipo funcional mínimo

5.2 Fase 2: Optimización del Mapa (1 semana)

Días 1-2: Conversión DWG → SVG

- Exportación desde AutoCAD
- Limpieza manual de capas innecesarias
- Primera pasada de optimización automática

Días 3-4: Optimización agresiva

- Desarrollo de scripts de limpieza personalizados
- Reducción de precisión numérica (8 decimales → 1-2)
- Eliminación de elementos invisibles/redundantes
- Testing de compatibilidad cross-browser

Día 5: Validación visual

- Comparación lado a lado (original vs optimizado)
- Verificación de que no se perdió información crítica
- Ajustes finales

Objetivo de reducción: 35 MB → 2-5 MB (80-85% de reducción)

Entregable: SVG optimizado + Script de optimización reutilizable

5.3 Fase 3: Desarrollo del Core (2 semanas)

Semana 1: Sistema de visualización

Días 1-2: Arquitectura de capas

- Implementación del sistema de grupos SVG
- Carga asíncrona de capas externas
- Sistema de toggle de visibilidad

Días 3-4: Pan & Zoom

- Event listeners (mouse, touch, wheel)
- Transformaciones con requestAnimationFrame
- Límites de zoom (min/max)

- Inertia y smooth animations

Día 5: Responsive design

- Media queries
- Touch gestures (pinch-to-zoom)
- Orientación de dispositivo

Semana 2: Sistema de nodos y rutas

Días 1-2: Gestión de nodos

- Estructura de datos (JSON)
- Detección automática de nodos en SVG
- Sistema de selección (click handlers)
- Estados visuales (hover, selected)

Días 3-4: Algoritmo de pathfinding

- Implementación de Dijkstra
- Construcción del grafo de navegación
- Fallback a línea recta
- Visualización de la ruta

Día 5: Testing de integración

- Pruebas end-to-end
- Fix de bugs críticos

Entregable: Aplicación funcional con features core

5.4 Fase 4: Interfaz de Usuario (1 semana)

Días 1-2: Diseño visual

- Paleta de colores corporativa (Universidad Veracruzana)
- Tipografía y espaciados
- Diseño de botones y controles
- Iconografía

Días 3-4: Implementación de UI

- Header y navegación
- Panel de control (selectores inicio/destino)

- Botones flotantes (zoom, centrar)
- Loader y estados de carga
- Mensajes de error/confirmación

Día 5: UX polish

- Animaciones y transiciones
- Feedback visual
- Tooltips y ayuda contextual

Entregable: Interfaz completa y pulida

5.5 Fase 5: Testing y Optimización (1 semana)

Días 1-2: Testing funcional

- Casos de prueba documentados
- Testing manual de todos los flujos
- Validación de cálculo de rutas
- Compatibilidad de navegadores (Chrome, Firefox, Safari, Edge)

Días 3-4: Performance optimization

- Profiling con Chrome DevTools
- Optimización de cuellos de botella
- Lazy loading de assets
- Caché de elementos frecuentes

Día 5: Testing en dispositivos reales

- Android (gama baja, media, alta)
- iOS (iPhone 8+, iPad)
- Tablets
- Ajustes de rendimiento específicos

Entregable: Aplicación optimizada y testeada

5.6 Fase 6: Deployment y Documentación (3 días)

Día 1: Preparación para producción

- Minificación de JS/CSS
- Optimización de assets

- Configuración de servidor (headers, caché)

Día 2: Deploy

- Subida a servidor/hosting
- Configuración de dominio
- Testing post-deploy
- Monitoreo de errores (opcional: Sentry)

Día 3: Documentación

- Manual de usuario
- Documentación técnica (para mantenimiento)
- Guía de actualización del mapa

Entregable: Aplicación en producción + Documentación completa

---

6. ESTIMACIÓN TOTAL DE TIEMPO

6.1 Desglose por Fases

Fase	Duración	Porcentaje
1. Preparación y Análisis	5 días	11%
2. Optimización del Mapa	5 días	11%
3. Desarrollo del Core	10 días	22%
4. Interfaz de Usuario	5 días	11%
5. Testing y Optimización	5 días	11%
6. Deployment y Documentación	3 días	7%
Buffer para imprevistos (20%)	7 días	15%
Reuniones y coordinación	5 días	11%
TOTAL	**45 días (~9 semanas)**	**100%**

6.2 Estimación en Horas de Desarrollo Efectivo

Asumiendo **6 horas productivas de código por día** (considerando reuniones, interrupciones, context switching):

Total: 45 días × 6 horas = 270 horas de desarrollo

6.3 Escenarios de Equipo

Escenario 1: Desarrollador individual (solo)

- **Duración:** 9 semanas (~2.25 meses)
- **Ventajas:** Coherencia de código, no requiere coordinación
- **Desventajas:** Riesgo de burnout, sin code review

Escenario 2: Equipo de 2 desarrolladores

- **Duración:** 5-6 semanas (~1.5 meses)
- **División:** Dev 1 (Backend/Core) + Dev 2 (Frontend/UI)
- **Overhead de coordinación:** +10% tiempo
- **Ventajas:** Code review, pair programming en tareas críticas

Escenario 3: Equipo de 3+ desarrolladores

- **Duración:** 4 semanas (1 mes)
- **División:** Lead + Dev Core + Dev UI/UX
- **Overhead de coordinación:** +20% tiempo
- **Ley de Brooks:** Añadir personas no reduce linealmente el tiempo

---

7. POR QUÉ NO SE PUEDE HACER EN MENOS TIEMPO

7.1 Complejidad Inherente del Software

Falacia común: "Es solo un mapa con puntos y líneas"

Realidad:

- 15 tecnologías/conceptos diferentes involucrados
- 3 algoritmos no triviales (Dijkstra, optimización SVG, collision detection)
- 1,500-2,000 líneas de código (estimado)
- 50+ funciones interdependientes
- 100+ casos de prueba

Comparación con otros proyectos de complejidad similar:

- Blog personal: 1-2 semanas
- E-commerce básico: 8-12 semanas
- **Mapa interactivo: 6-9 semanas** ← Estamos aquí

- Sistema de gestión académica: 16-24 semanas

7.2 Tiempo No Negociable: Testing y Debugging

Estadística de la industria: 40-50% del tiempo de desarrollo se invierte en testing y corrección de bugs.

En nuestro proyecto:

- 270 horas totales
- ~120 horas de testing/debugging (45%)

¿Por qué tanto?

1. **Cross-browser compatibility:** Cada navegador renderiza SVG ligeramente diferente
2. **Device testing:** Performance en móvil es crítico
3. **Edge cases:** ¿Qué pasa si el usuario hace zoom 100x? ¿Si clickea rápido 50 veces?
4. **Accesibilidad:** Navegación por teclado, screen readers
5. **Performance bajo carga:** ¿Funciona con 500 nodos simultáneos?

Recortar testing = Lanzar software con bugs = Mala experiencia de usuario = Pérdida de confianza

7.3 La Ley de Hofstadter

"Siempre toma más tiempo del que esperas, incluso cuando tienes en cuenta la Ley de Hofstadter" — Douglas Hofstadter

Aplicado a nuestro proyecto:

- Estimación inicial (sin experiencia): 2-3 semanas
- Estimación realista (con experiencia): 6-9 semanas
- Tiempo real (con imprevistos): 7-11 semanas

Imprevistos comunes (basados en proyectos reales):

- Plano de AutoCAD tiene errores estructurales (1-2 días extra)
- Cliente cambia requisitos a mitad del proyecto (3-5 días)
- Bug crítico cross-browser descubierto tarde (2-3 días)
- Servidor de producción tiene configuración diferente (1 día)
- Optimización de rendimiento requiere refactoring mayor (3-4 días)

7.4 Calidad vs Velocidad: El Triángulo de Hierro

Elige dos:

- Rápido + Barato = Baja calidad (bugs, código spagueti)
- Rápido + Bueno = Costoso (más desarrolladores, horas extra)
- Barato + Bueno = Lento (un desarrollador junior, mucho tiempo)

Para un proyecto académico/institucional, se debe priorizar: Bueno + Razonable en tiempo

---

8. TECNOLOGÍAS MODERNAS Y POR QUÉ SE EVITARON

8.1 React/Vue/Angular

Por qué NO se usó:

- **Overkill para el problema:** El 80% del trabajo es manipulación directa del DOM SVG
- **Virtual DOM es un obstáculo:** React reconcilia cambios, agregando latencia innecesaria
- **Bundle size:** Framework + dependencias = 200+ KB adicionales
- **Complejidad de setup:** Webpack/Vite config, JSX transform, etc.

Cuándo SÍ usarlos:

- Sistema de administración de contenido del campus
- Dashboard con múltiples vistas y routing
- Aplicación con autenticación y manejo de usuarios

8.2 TypeScript

Por qué NO se usó:

- **Curva de aprendizaje:** Requiere conocimiento de tipos estáticos
- **Overhead de build:** Transpilación TS → JS
- **No previene errores de lógica:** Solo errores de tipos

Cuándo SÍ usarlo:

- Equipos grandes (4+ developers)
- Proyectos de larga duración (1+ año de mantenimiento)
- APIs complejas con muchos tipos customizados

8.3 Leaflet.js / Mapbox

Por qué NO se usó:

- **Diseñados para mapas geográficos:** Requieren tiles, coordenadas lat/lon

- **Overhead innecesario:** Capas de abstracción que no necesitamos
- **Difícil integración con SVG custom:** Están pensados para raster tiles

Cuándo SÍ usarlos:

- Mapas reales (Google Maps style)
- Integración con GPS/geolocalización
- Múltiples capas de información geográfica

8.4 D3.js

Considerado pero descartado:

- **Pros:** Excelente para visualizaciones de datos complejas
- **Cons:** Curva de aprendizaje pronunciada, overkill para nuestro caso
- **Decisión:** Vanilla JS con funciones específicas es más mantenible

---

9. ESCALABILIDAD Y MANTENIMIENTO FUTURO

9.1 Funcionalidades Futuras (Roadmap)

Fase 2 del proyecto (opcional, +4-6 semanas):

1. **Sistema de búsqueda**
    - Autocompletado de ubicaciones
    - Búsqueda fuzzy (tolerante a typos)
    - Tiempo estimado: 5 días
2. **Información contextual**
    - Pop-ups con info de edificios
    - Horarios de servicios
    - Fotos de ubicaciones
    - Tiempo: 7 días
3. **Modo 3D/Indoor**
    - Navegación por pisos de edificios
    - Visualización 3D con Three.js
    - Tiempo: 15 días (alta complejidad)
4. **Accesibilidad avanzada**

- Rutas para personas con movilidad reducida
- Elevadores y rampas prioritarias
- Tiempo: 5 días

5. **Backend y datos dinámicos**
    - Panel de administración
    - API REST para actualizar POIs
    - Base de datos (PostgreSQL + PostGIS)
    - Tiempo: 20 días

9.2 Mantenimiento Continuo

Tiempo estimado post-launch: 4-8 horas/mes

Tareas típicas:

- Actualización del mapa (construcciones nuevas)
- Fix de bugs reportados por usuarios
- Actualizaciones de seguridad
- Optimizaciones de rendimiento

---

10. CONCLUSIONES

10.1 Resumen Ejecutivo

El desarrollo de una aplicación web de mapas interactivos del campus es un proyecto de **complejidad media-alta** que requiere:

- **Tiempo mínimo realista:** 6-9 semanas (1 desarrollador experimentado)
- **Conocimientos requeridos:** JavaScript avanzado, SVG/DOM, algoritmos, UX/UI, performance optimization
- **Líneas de código:** ~1,500-2,000 LOC
- **Archivos de código:** ~10-15 módulos
- **Horas de desarrollo:** 250-300 horas

10.2 Factores Críticos de Éxito

1. **Optimización del SVG es NO NEGOCIABLE:** 35 MB → 2-5 MB
2. **Testing en dispositivos reales:** Desktop y móvil
3. **Performance es una feature:** <100ms de lag en interacciones

4. **Documentación desde el inicio:** Código autodocumentado + comentarios

10.3 Mensaje Final para Stakeholders

No es realista esperar que un proyecto de esta escala se complete en 1-2 semanas.

Las estimaciones cortas típicamente resultan en:

- Software con bugs críticos
- Performance deficiente (app lenta/inusable)
- Código espagueti difícil de mantener
- Experiencia de usuario pobre
- Necesidad de reescribir todo en 6 meses

Invertir el tiempo necesario desde el inicio ahorra dinero y problemas a largo plazo.

10.4 Recomendaciones

1. **Planificar con buffers realistas:** Estimación × 1.5 = tiempo real probable
2. **Priorizar features:** MVP primero, features avanzadas después
3. **Comunicación constante:** Updates semanales con stakeholders
4. **No saltarse testing:** Calidad sobre velocidad
5. **Documentar decisiones técnicas:** Para futuros mantenedores

---

ANEXO A: Glosario Técnico

SVG (Scalable Vector Graphics): Formato de imagen vectorial basado en XML, escalable sin pérdida de calidad.

DOM (Document Object Model): Representación en memoria del HTML/SVG, manipulable con JavaScript.

RequestAnimationFrame: API del navegador que sincroniza código con el ciclo de repintado (60 FPS).

Pan: Acción de arrastrar/mover el mapa.

Zoom: Acción de acercar/alejar la vista.

PathFinding: Algoritmo para encontrar la ruta más corta entre dos puntos en un grafo.

Dijkstra: Algoritmo de camino más corto para grafos con pesos no negativos.

Grafo: Estructura de datos con nodos (ubicaciones) y aristas (conexiones).

Viewport: Área visible del mapa en la pantalla.

Throttling: Técnica para limitar la frecuencia de ejecución de una función.

Bundle Size: Tamaño total del JavaScript/CSS que el navegador debe descargar.

Cross-browser: Compatible con múltiples navegadores (Chrome, Firefox, Safari, etc).

ANEXO B: Referencias y Recursos

Documentación técnica:

- MDN Web Docs (SVG): https://developer.mozilla.org/en-US/docs/Web/SVG
- Algoritmo de Dijkstra: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- Web Performance: https://web.dev/performance/
- SVG Optimization Guide: https://jakearchibald.github.io/svgomg/

Librerías consideradas:

- SVGO (SVG Optimizer): https://github.com/svg/svgo
- D3.js: https://d3js.org/
- Leaflet.js: https://leafletjs.com/

Herramientas de desarrollo:

- Chrome DevTools Performance Profiler
- Lighthouse (auditoría de rendimiento)
- BrowserStack (testing cross-browser)

ANEXO C: Comparación con Proyectos Similares

C.1 Casos de Estudio Reales

Proyecto 1: Google Maps Indoor (2011-2013)

- **Equipo:** 15+ ingenieros
- **Tiempo:** 2 años de desarrollo inicial
- **Complejidad:** Alta (múltiples pisos, integración con GPS, millones de edificios)
- **Lección:** Incluso con recursos ilimitados, los mapas interactivos toman tiempo

Proyecto 2: Mapa Campus MIT (2015)

- **Equipo:** 3 desarrolladores
- **Tiempo:** 4 meses

- **Tecnologías:** Leaflet.js + OpenStreetMap
- **Features:** Búsqueda, rutas, horarios de clases
- **Lección:** Proyectos universitarios realistas toman 3-6 meses

Proyecto 3: Campus Navigator - Universidad de Stanford (2018)

- **Equipo:** 2 desarrolladores + 1 diseñador
- **Tiempo:** 5 meses
- **Tecnologías:** React + Mapbox
- **Presupuesto:** $45,000 USD
- **Lección:** Calidad profesional requiere inversión significativa

C.2 Comparación con Nuestro Proyecto

Aspecto	Nuestro Proyecto	Promedio Industria
Tiempo estimado	6-9 semanas	8-16 semanas
Complejidad	Media-Alta	Media-Alta
Equipo	1-2 devs	2-4 devs
Presupuesto	Académico	$30-60K USD

Conclusión: Nuestra estimación de 6-9 semanas es **conservadora y realista**, incluso por debajo del promedio de la industria.

ANEXO D: Análisis de Riesgos

D.1 Riesgos Técnicos

Riesgo	Probabilidad	Impacto	Mitigación
SVG demasiado pesado para optimizar	Media	Alto	Rediseño con tiles/simplificación manual
Performance inaceptable en móviles	Baja	Alto	Profiling temprano, optimización progresiva
Incompatibilidad cross-browser	Media	Medio	Testing continuo, polyfills
Archivo DWG corrupto/incompleto	Baja	Alto	Validación temprana, backup con planos en papel

Riesgo	Probabilidad	Impacto	Mitigación
Algoritmo de rutas no encuentra caminos	Media	Medio	Fallback a línea recta, refinamiento iterativo

D.2 Riesgos de Proyecto

Riesgo	Probabilidad	Impacto	Mitigación
Cambios de requisitos a mitad	Alta	Alto	Desarrollo iterativo, MVP primero
Desarrollador único se enferma	Media	Alto	Documentación exhaustiva, code reviews
Plazos poco realistas impuestos	Alta	Crítico	Mostrar esta guía, negociar alcance
Falta de acceso a planos actualizados	Media	Alto	Contacto temprano con arquitectura
Stakeholders esperan features no planeadas	Alta	Medio	Documento de alcance firmado

D.3 Plan de Contingencia

Escenario 1: Solo quedan 3 semanas (50% del tiempo estimado)

- **Acción:** Reducir a MVP estricto
    - Solo mapa base + zoom/pan
    - Nodos pre-definidos (sin detección automática)
    - Sin cálculo de rutas (solo marcadores)
    - UI minimalista

Escenario 2: Performance inaceptable (<30 FPS en móvil)

- **Acción:** Simplificación radical del SVG
    - Convertir vectores complejos a imágenes raster
    - Sistema de LOD (Level of Detail): detalle solo en zoom alto
    - Considerar Canvas en lugar de SVG

Escenario 3: No hay grafo de navegación disponible

- **Acción:** Usar solo líneas rectas
    - Distancia euclidiana

- Sin pathfinding
- Nota al usuario: "Ruta aproximada, puede haber obstáculos"

ANEXO E: Métricas de Éxito

E.1 KPIs Técnicos

Performance:

- ✅ Tiempo de carga inicial: <3 segundos
- ✅ FPS durante pan/zoom: >50 FPS
- ✅ Tiempo de cálculo de ruta: <200ms
- ✅ Tamaño de bundle: <500 KB (JS + CSS)
- ✅ SVG optimizado: <5 MB

Calidad de código:

- ✅ Cobertura de tests: >70%
- ✅ Cero errores en consola
- ✅ Lighthouse Score: >90
- ✅ Accesibilidad: WCAG 2.1 AA

Compatibilidad:

- ✅ Chrome 90+
- ✅ Firefox 88+
- ✅ Safari 14+
- ✅ Edge 90+
- ✅ iOS Safari 14+
- ✅ Chrome Android 90+

E.2 KPIs de Usuario

Usabilidad:

- ✅ Usuario nuevo puede encontrar ruta en <30 segundos
- ✅ Tasa de error: <5% (clicks incorrectos)
- ✅ Satisfacción: >4.0/5.0 en encuestas

- ✅ Tiempo promedio de uso: 2-5 minutos

Adopción:

- ✅ 70% de usuarios nuevos regresan
- ✅ 100+ usuarios activos diarios (DAU)
- ✅ <10% tasa de rebote

E.3 Proceso de Validación

Testing con usuarios reales:

Fase Alpha (semana 5):

- 5-10 testers internos (equipo de desarrollo)
- Identificar bugs críticos
- Validar flujos principales

Fase Beta (semana 7):

- 30-50 estudiantes voluntarios
- Encuesta de satisfacción
- Métricas de analytics (Google Analytics/Matomo)

Lanzamiento (semana 9):

- Monitoreo de errores en tiempo real (Sentry)
- Hotfixes según necesidad
- Iteración basada en feedback

ANEXO F: Casos de Uso Detallados

F.1 Caso de Uso Principal: Estudiante Nuevo

Actor: María, estudiante de primer semestre

Precondición: María está en la entrada del campus con su smartphone

Flujo normal:

1. María abre la aplicación en su celular
2. Ve el mapa completo del campus cargado en 2 segundos
3. Busca su salón "Edificio A - 301" usando el selector
4. Toca "Edificio A - 301" en la lista desplegable

5. El mapa hace zoom al edificio y marca el salón en rojo
6. María arrastra el mapa para explorar el entorno
7. Selecciona "Entrada Principal" como punto de inicio
8. Presiona "Generar Ruta"
9. Ve una línea verde animada mostrando el camino
10. Sigue la ruta físicamente hasta llegar a su salón

Tiempo total: <2 minutos

Post-condición: María llegó exitosamente a su clase

F.2 Caso de Uso Secundario: Búsqueda de Servicio

Actor: Carlos, estudiante de tercer semestre

Objetivo: Encontrar la cafetería más cercana

Flujo:

1. Carlos abre la app en la biblioteca
2. Toca en su ubicación actual (marcador azul en pantalla)
3. Selecciona "Cafetería Central" del desplegable de destinos
4. Genera la ruta
5. Ve que está a 3 minutos caminando
6. Sigue la ruta mostrada en pantalla

Variante: Si hay múltiples cafeterías, el sistema podría calcular la más cercana (feature futura)

F.3 Caso de Uso de Emergencia

Actor: Personal de seguridad

Objetivo: Localizar rápidamente el edificio con una emergencia reportada

Flujo:

1. Recibe reporte: "Emergencia en Edificio C"
2. Abre app en tablet
3. Busca "Edificio C"
4. El mapa centra y resalta el edificio
5. Ve rutas de acceso vehicular destacadas

6. Coordina respuesta de emergencia

Requisito: Modo "Admin" con información adicional (rutas vehiculares, salidas de emergencia)

ANEXO G: Decisiones Arquitectónicas (ADRs)

ADR-001: Uso de Vanilla JavaScript sobre Frameworks

Fecha: Inicio del proyecto **Estado:** Aceptado **Contexto:** Necesidad de decidir tecnología principal

Decisión: Usar JavaScript ES6+ sin frameworks como React/Vue/Angular

Justificación:

1. Manipulación directa del DOM SVG es más eficiente
2. Bundle size reducido (mejor performance en móvil)
3. Menor complejidad de configuración
4. Suficiente para el alcance del proyecto

Consecuencias:

- ✅ Mejor performance
- ✅ Menos dependencias
- ❌ Sin ecosystem de componentes pre-hechos
- ❌ Manejo de estado manual

ADR-002: Algoritmo de Dijkstra para Pathfinding

Fecha: Fase de diseño **Estado:** Aceptado **Contexto:** Necesidad de calcular rutas óptimas

Decisión: Implementar Dijkstra en lugar de A* o BFS

Justificación:

1. Grafos pequeños (<200 nodos) no justifican A*
2. Garantiza ruta óptima (A* puede necesitar heurística compleja)
3. Implementación más simple y robusta
4. Performance suficiente (<100ms en dispositivos promedio)

Consecuencias:

- ✅ Rutas garantizadas óptimas

- ✅ Código más mantenible
- ❌ Ligeramente más lento que A* en grafos grandes (irrelevante en nuestro caso)

ADR-003: Sistema de Capas SVG Separadas

Fecha: Semana 2 de desarrollo **Estado:** Aceptado **Contexto:** Conflictos de z-index y performance al actualizar rutas

Decisión: Separar SVG en capas: layer-dwg, layer-nodes, layer-route

Justificación:

1. Permite actualizar solo la capa de rutas sin re-renderizar todo
2. Facilita toggle de visibilidad (mostrar/ocultar plano base)
3. Mejor organización del código
4. Event delegation más limpio

Consecuencias:

- ✅ Performance mejorado en actualizaciones parciales
- ✅ Código más modular
- ❌ Complejidad adicional en carga inicial
- ✅ Facilita features futuras (capas de pisos múltiples)

ADR-004: Sin Base de Datos en MVP

Fecha: Fase de planificación **Estado:** Aceptado (con plan de revisión en Fase 2)

Decisión: Usar archivos JSON estáticos para nodos y grafo

Justificación:

1. Reduce complejidad del MVP
2. Datos cambian poco frecuentemente (edificios no se mueven)
3. Evita necesidad de backend/servidor
4. Deployment más simple (static hosting)

Consecuencias:

- ✅ MVP más rápido de desarrollar
- ✅ Hosting gratuito posible (GitHub Pages, Netlify)
- ❌ Actualización de datos requiere re-deploy

- ⚠️ En Fase 2, migrar a API REST + DB para administración dinámica

---

ANEXO H: Presupuesto Estimado (Referencia)

H.1 Costos de Desarrollo (Freelance/Consultora)

Asumiendo tarifa promedio: $30-50 USD/hora

Fase	Horas	Costo ($30/h)	Costo ($50/h)
1. Análisis	30h	$900	$1,500
2. Optimización SVG	30h	$900	$1,500
3. Core Development	60h	$1,800	$3,000
4. UI/UX	30h	$900	$1,500
5. Testing	30h	$900	$1,500
6. Deploy/Docs	18h	$540	$900
Subtotal	198h	$5,940	$9,900
Buffer (20%)	40h	$1,200	$2,000
TOTAL	238h	**$7,140**	**$11,900**

Rango realista: $7,000 - $12,000 USD para desarrollo completo

H.2 Costos Adicionales

Infraestructura:

- Dominio (.edu o .edu.mx): $10-20/año
- Hosting básico (Netlify/Vercel): $0-20/mes
- CDN (Cloudflare): $0 (plan gratuito suficiente)
- SSL Certificate: $0 (Let's Encrypt)

Herramientas:

- Figma (diseño): $0 (plan gratuito)
- GitHub: $0 (plan gratuito)
- Analytics: $0 (Google Analytics o Matomo self-hosted)

Total infraestructura: <$50/año

H.3 Retorno de Inversión (ROI)

Beneficios tangibles:

- Mejora experiencia de estudiantes nuevos
- Reducción de consultas al personal (información de ubicaciones)
- Imagen de modernización institucional
- Base para features futuras (eventos, notificaciones)

Ahorro estimado:

- Personal administrativo: 5-10 horas/semana menos respondiendo preguntas
- Costo de señalización física: Reducción en carteles/mapas impresos
- Orientación de nuevo ingreso: Proceso más eficiente

ROI estimado: 12-18 meses para recuperar inversión en ahorro operativo

---

ANEXO I: Plan de Mantenimiento Post-Lanzamiento

I.1 Mantenimiento Correctivo

Objetivos: Corregir bugs reportados por usuarios

Frecuencia: Según necesidad (monitoreo continuo)

Proceso:

1. Usuario reporta bug vía formulario/email
2. Desarrollador reproduce el error
3. Fix + testing en <48 horas (bugs críticos)
4. Deploy de hotfix
5. Notificación al usuario

Tiempo estimado: 2-4 horas/mes

I.2 Mantenimiento Preventivo

Objetivos: Evitar problemas futuros, mantener performance

Frecuencia: Mensual

Tareas:

- Actualización de dependencias (si se usan)
- Revisión de logs de errores

- Análisis de performance (Lighthouse)
- Backup de código y datos

Tiempo estimado: 2 horas/mes

I.3 Mantenimiento Evolutivo

Objetivos: Mejoras y nuevas features

Frecuencia: Trimestral

Ejemplos:

- Agregar nuevos edificios/construcciones
- Actualizar estilos visuales
- Optimizaciones de UX basadas en feedback
- A/B testing de features

Tiempo estimado: 4-8 horas/trimestre

I.4 Costo Total de Mantenimiento Anual

Estimación conservadora:

- Correctivo: 4h/mes × 12 = 48h
- Preventivo: 2h/mes × 12 = 24h
- Evolutivo: 8h/trimestre × 4 = 32h
- **Total:** 104 horas/año

Costo anual: $3,000 - $5,000 USD (a $30-50/hora)

Alternativa interna: Si la universidad tiene equipo de desarrollo interno, el mantenimiento puede internalizarse por ~$2,000/año en costo real.

ANEXO J: Lecciones Aprendidas de Proyectos Similares

J.1 Errores Comunes a Evitar

Error #1: Subestimar la optimización del SVG

- **Consecuencia:** App lenta/inutilizable en móviles
- **Prevención:** Dedicar 1 semana completa solo a optimización

Error #2: No testear en dispositivos reales

- **Consecuencia:** Performance excelente en desktop, terrible en móvil

- **Prevención:** Testing continuo en Android/iOS de gama baja

Error #3: Pathfinding sin fallback

- **Consecuencia:** Si falta el grafo, la app es inútil
- **Prevención:** Línea recta como fallback siempre

Error #4: Zoom infinito sin límites

- **Consecuencia:** Usuarios se "pierden" con zoom excesivo
- **Prevención:** Límites min/max de zoom (0.2x - 6x)

Error #5: No documentar decisiones técnicas

- **Consecuencia:** Mantenimiento futuro es imposible sin el desarrollador original
- **Prevención:** Comentarios en código + este documento

J.2 Mejores Prácticas Identificadas

✅ **Desarrollo iterativo con demos semanales**

- Permite ajustar requisitos tempranamente
- Mantiene a stakeholders informados
- Previene "sorpresas" al final

✅ **Performance budgets desde el inicio**

- Bundle size: <500 KB
- FPS: >50
- Tiempo de carga: <3s
- Si se exceden, refactorizar inmediatamente

✅ **Accesibilidad no es opcional**

- Navegación por teclado
- Screen reader friendly
- Contraste de colores (WCAG AA)
- Labels descriptivos en todos los controles

✅ **Progressive enhancement**

- App funciona sin JavaScript (mostrar mapa estático)
- Features avanzadas se agregan progresivamente
- Fallbacks para navegadores antiguos

CONCLUSIÓN FINAL

Resumen Ejecutivo para Tomadores de Decisiones

Proyecto: Aplicación Web de Mapas Interactivos Campus Ingeniería UV

Complejidad: Media-Alta (comparable a e-commerce básico)

Tiempo realista mínimo: 6-9 semanas (1.5-2.25 meses)

Equipo recomendado: 1-2 desarrolladores + 1 diseñador (consulta)

Costo estimado: $7,000-12,000 USD (freelance) o ~2 meses de salario de desarrollador interno

Tecnologías justificadas:

- ✅ Vanilla JavaScript ES6+ (performance óptimo)
- ✅ SVG (escalabilidad visual)
- ✅ Dijkstra (pathfinding garantizado)
- ✅ Módulos ES6 (mantenibilidad)

Principales desafíos:

1. Optimización de SVG de AutoCAD (35 MB → <5 MB)
2. Performance fluido en móviles (>50 FPS)
3. Testing cross-browser exhaustivo
4. Cálculo de rutas óptimas con grafo

Por qué NO se puede hacer en 1-2 semanas:

- 15 tecnologías/conceptos diferentes
- 250-300 horas de desarrollo real
- 40-50% del tiempo es testing/debugging
- Complejidad inherente del software de calidad

Factores de éxito críticos:

- ✅ Estimaciones realistas aceptadas por stakeholders
- ✅ Alcance bien definido (MVP primero)
- ✅ Testing continuo desde semana 1
- ✅ Comunicación constante con usuarios finales

Alternativa si el tiempo es limitado:

- Reducir a MVP estricto (solo visualización + zoom, sin rutas)
- Tiempo reducido: 3-4 semanas
- Features avanzadas en Fase 2

Mensaje Final

Este documento demuestra con fundamentos técnicos sólidos que **el desarrollo de software de calidad requiere tiempo**. Los plazos poco realistas resultan invariablemente en:

❌ Software con bugs críticos
❌ Experiencia de usuario deficiente
❌ Código imposible de mantener
❌ Necesidad de reescribir todo en 6-12 meses

Invertir el tiempo adecuado desde el inicio (6-9 semanas) ahorra dinero, frustración y tiempo a largo plazo.

La planificación detallada presentada en este documento no es "sobrestimación conservadora", sino la **realidad del desarrollo profesional de software** validada por décadas de experiencia en la industria y respaldada por proyectos similares documentados.

Elaborado por: Equipo de Desarrollo
Fecha: 2025
Versión: 1.0
Clasificación: Documento Técnico Interno

Este documento puede ser actualizado según evolucionen los requisitos del proyecto o se identifiquen nuevos desafíos técnicos.