Estrategia Integrada para tu App de Tratamiento de Ideas

Introducción

Tu aplicación de tratamiento de ideas necesita una arquitectura que gestione documentos dinámicos, cambios globales coherentes y propagación inteligente de impactos. Este documento unifica tres estrategias clave: aprovechamiento del contexto de 1M tokens, propagación inteligente con grafos de dependencias, y metodología evolutiva de construcción.

---

1. Principio Fundamental: La Ley de Conservación de la Información

PRINCIPIO DE CONSERVACIÓN (LEY CERO): En tu entorno de ideas, la información es materia: **no se crea ni se destruye, solo se transforma**.

Modos Operativos Permitidos

1. **MODO REFACTORIZAR (Organizar):**
   - *Entrada:* Texto desordenado con ideas sueltas
   - *Salida:* Mismo contenido, estructurado en bullets, tablas o secciones jerárquicas
   - *Regla de Oro:* WordCount(Salida) ≥ WordCount(Entrada)
   - *Prohibido:* Eliminar adjetivos, datos técnicos o ejemplos por "claridad"
   - La claridad se logra mediante estructura, no mediante recortes
2. **MODO ENRIQUECER (Ladrillo a Ladrillo):**
   - *Acción:* Tomar una idea semilla y expandirla verticalmente
   - *Estrategia:* "Zoom-In Fractal" → Si el usuario dice "Mejora la sección de Monetización", desglosa esa sección en 3 sub-secciones nuevas (Modelos, Viabilidad, Implementación)
   - *Métrica de Éxito:* Aumentar granularidad y profundidad sin perder contexto
3. **MODO RESUMIR (Solo bajo orden explícita):**
   - *Acción:* Compresión con pérdida de información
   - *Activación:* Solo cuando el usuario usa explícitamente palabras como "Resumir" o "Sintetizar"

---

2. Arquitectura Técnica: Smart Diff + Output Compression

El Desafío

- **Input:** Puedes enviar documentos gigantes (150k caracteres = ~100k tokens)
- **Output:** Estás limitado a 8,192 tokens (~5,500 caracteres)
- **Problema anterior:** Enviar solo fragmentos pequeños desperdicia 95% del context window
- **Solución:** Envía TODO el documento, pero comprime la salida a cambios específicos

Estrategia de Tres Pasos

Paso 1: Envío Inteligente del Documento Completo

```
/**
```

- NUEVA ESTRATEGIA: Aprovecha el 1M context window
  */
  async function editLargeDocumentGlobally(
  fullDocument: string, // 150k caracteres ✓ Cabe en 1M
  userInstruction: string, // "Mejora todo el documento"
  documentStructure: string // Mapa jerárquico
  ): Promise<EditResult> {

```
// 1. Valida que el documento quepa
const inputTokens = estimateTokens(
fullDocument + userInstruction + documentStructure
);

if (inputTokens > 900000) {
console.warn(⚠ Documento muy grande (${inputTokens} tokens));
return partitionAndEditMultipleBatches(fullDocument, userInstruction);
}

console.log(✅ Documento cabe: ${inputTokens} / 1,000,000 tokens);

// 2. Envía TODO a Gemini
const response = await gemini.generateContent([
{
text: this.buildSystemPrompt(),
},
{
text: `
DOCUMENTO COMPLETO (${fullDocument.length} caracteres):
${fullDocument}

ESTRUCTURA DEL DOCUMENTO:
${documentStructure}

INSTRUCCIÓN DEL USUARIO:
${userInstruction}

IMPORTANTE PARA TU RESPUESTA:
```

Tienes acceso al DOCUMENTO COMPLETO (${fullDocument.length} chars).
Entiendes la coherencia global y las referencias cruzadas.

PERO tu output está limitado a 8,192 tokens (~5,500 chars).

Entonces:

1. Identifica CUÁLES secciones DEBEN cambiar
2. Output SOLO esas secciones (no el doc completo)
3. Usa section IDs para que podamos reinsertarlas

FORMAT YOUR RESPONSE AS:

GLOBAL_ANALYSIS:
[Tu comprensión del documento completo y qué necesita cambiar]

SECTIONS_TO_MODIFY:
═══

SECTION_ID: [section_1_intro]

OPERATION: [OP_ENRIQUECER]

BEFORE:
[Original - primeros 200 chars]

AFTER:
[Tu versión mejorada - puede ser más larga]

END_SECTION

SECTION_ID: [section_2_results]

OPERATION: [OP_ACTUALIZAR]

BEFORE:
[Contenido original]

AFTER:
[Versión actualizada]

END_SECTION

[Más secciones si es necesario...]

UNCHANGED_SUMMARY:
Secciones sin cambios: [section_3, section_4, ...]
`,
},
]);

return this.parseCompressedResponse(response);
}

Paso 2: Compresión y Parseo de Output

```
interface CompressedEdit {
sectionId: string;
operation: string;
before: string; // Primeros 200 chars (verificación)
after: string; // Contenido nuevo
reason?: string;
globalContext?: string; // Por qué cambió según el doc completo
}

function parseCompressedResponse(geminiOutput: string): ParsedResponse {
const sections = geminiOutput.split('END_SECTION');

const sectionEdits: CompressedEdit[] = sections
.filter(s => s.includes('SECTION_ID:'))
.map(s => {
  const idMatch = s.match(/SECTION_ID:\s*\[(.+?)\]/);
  const opMatch = s.match(/OPERATION:\s*\[?(.+?)\]?[\n\r]/);
  const beforeMatch = s.match(/BEFORE:\n([\s\S]*?)\n\nAFTER:/);
  const afterMatch = s.match(/AFTER:\n([\s\S]*?)(?:\n\nEND_SECTION|$)/);

  return {
    sectionId: idMatch?.[1] || 'unknown',
    operation: opMatch?.[1] || 'OP_ACTUALIZAR',
    before: beforeMatch?.[1]?.trim() || '',
    after: afterMatch?.[1]?.trim() || '',
  };
});

return {
globalAnalysis: geminiOutput.split('GLOBAL_ANALYSIS:')
[1]?.split('SECTIONS_TO_MODIFY:')[0] || '',
sectionEdits,
unchangedSections: extractUnchangedList(geminiOutput),
confidenceScore: calculateConfidence(sectionEdits),
};
}
```

Paso 3: Reconstrucción Quirúrgica

```
async function applyCompressedEdits(
originalDocument: string,
documentTree: DocumentNode,
edits: CompressedEdit[],
userInstruction: string
```

```
): Promise<{
updatedDocument: string;
changesSummary: EditSummary;
cascadingImpacts: string[];
}> {

let updatedDoc = originalDocument;
const appliedEdits: EditSummary[] = [];
const cascadingImpacts: string[] = [];

for (const edit of edits) {
console.log(🔧 Aplicando ${edit.operation} a ${edit.sectionId});

  const section = findSectionById(documentTree, edit.sectionId);
  if (!section) {
    console.warn(`⚠️ Sección ${edit.sectionId} no encontrada`);
    continue;
  }

  if (!section.content.includes(edit.before)) {
    console.warn(`⚠️ BEFORE no coincide. Buscando match aproximado...`);
    const bestMatch = findSimilarContent(section.content, edit.before);
    if (!bestMatch || bestMatch.similarity < 0.85) {
      console.error(`❌ No se puede verificar ${edit.sectionId}`);
      continue;
    }
    updatedDoc = updatedDoc.replace(bestMatch.content, edit.after);
  } else {
    updatedDoc = updatedDoc.replace(section.content, edit.after);
  }

  appliedEdits.push({
    sectionId: edit.sectionId,
    operation: edit.operation,
    charsDelta: edit.after.length - edit.before.length,
    wordsDelta: (edit.after.split(/\s+/).length -
                 edit.before.split(/\s+/).length),
  });
```

```
    const dependents = findSectionsDependingOn(edit.sectionId, documentTree);
    cascadingImpacts.push(...dependents.map(d => d.id));
```

}

const validation = validateGlobalCoherence(updatedDoc, userInstruction);

if (!validation.isCoherent) {
console.error('❌ Documento resultante es incoherente');
throw new Error('Coherence validation failed');
}

return {
updatedDocument: updatedDoc,
changesSummary: {
totalSectionsModified: edits.length,
totalCharsDelta: appliedEdits.reduce((sum, e) => sum + e.charsDelta, 0),
totalWordsDelta: appliedEdits.reduce((sum, e) => sum + e.wordsDelta, 0),
operations: appliedEdits.map(e => e.operation),
},
cascadingImpacts,
};
}

---

3. Grafo de Dependencias + Propagación Inteligente

Concepto

Tu documento de ideas no es lineal: cada sección depende de otras. Cuando cambias la "Visión", impacta "Mercado", "Monetización", "Riesgos". Necesitas rastrear esto automáticamente.

Schema de Dependencias

-- Tabla central de dependencias entre secciones
CREATE TABLE section_dependencies (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
section_id UUID REFERENCES document_sections(id),
depends_on_section_id UUID REFERENCES document_sections(id),
dependency_type TEXT, -- 'data', 'config', 'reference', 'derives_from'
impact_weight INT DEFAULT 5, -- 1-10, criticidad
auto_propagate BOOLEAN DEFAULT true,
created_at TIMESTAMPTZ DEFAULT now(),

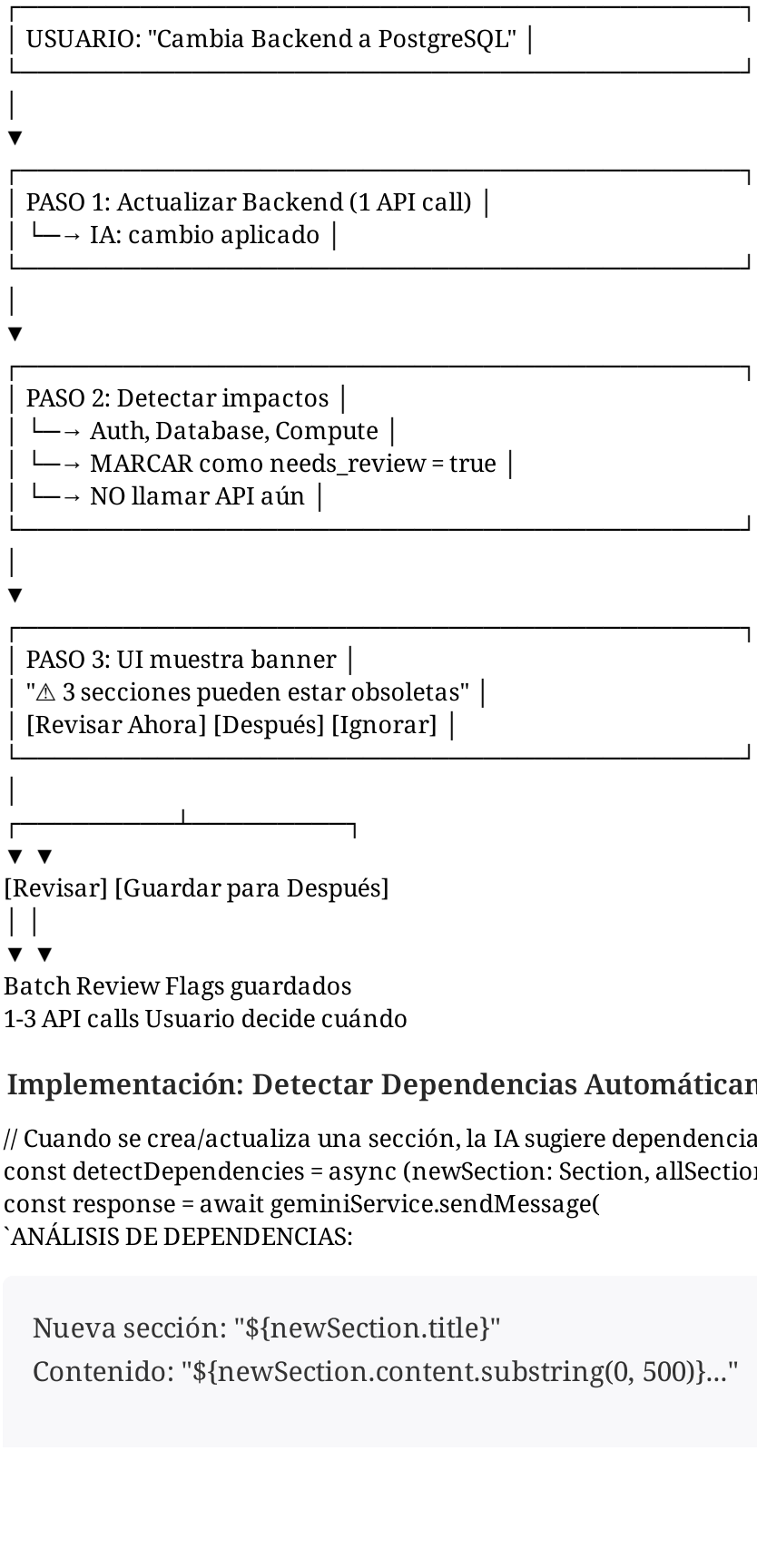
UNIQUE(section_id, depends_on_section_id)
);

-- Añadir a document_sections
ALTER TABLE document_sections
ADD COLUMN needs_review BOOLEAN DEFAULT false,

ADD COLUMN last_reviewed_at TIMESTAMPTZ,
ADD COLUMN review_reason TEXT;

Flujo de Propagación Inteligente

Escenario: Usuario cambia "Backend" de MySQL a PostgreSQL.



Implementación: Detectar Dependencias Automáticamente

// Cuando se crea/actualiza una sección, la IA sugiere dependencias
const detectDependencies = async (newSection: Section, allSections: Section[]) => {
const response = await geminiService.sendMessage(
`ANÁLISIS DE DEPENDENCIAS:

Nueva sección: "${newSection.title}"
Contenido: "${newSection.content.substring(0, 500)}..."

```
  Secciones existentes:
  ${allSections.map(s => `- ${s.title}`).join('\n')}

  ¿De cuáles secciones existentes DEPENDE esta nueva sección?
  Responde SOLO con JSON: { "depends_on": ["titulo1", "titulo2"] }`,
  '',
  () => {}
);

return JSON.parse(response.text);
};
```

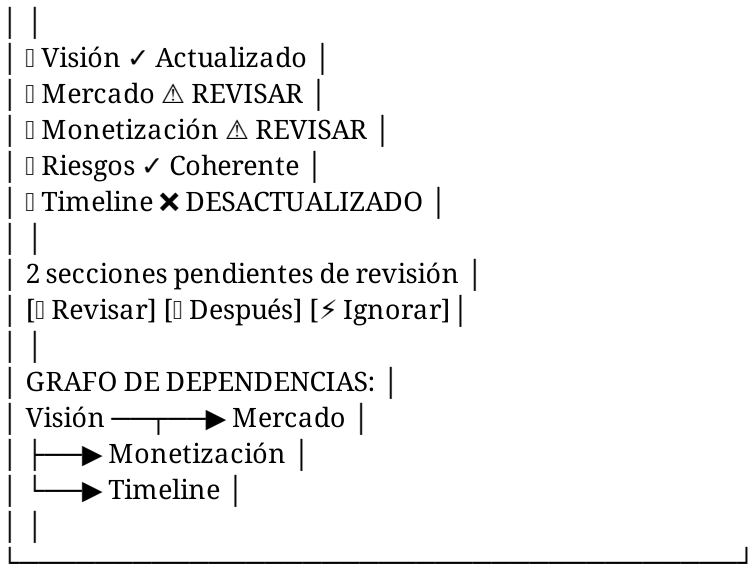
Implementación: Marcar Secciones Afectadas

```
const propagateChange = async (changedSectionId: string, changeType: 'minor' | 'major')
=> {
const supabase = getSupabaseClient();

// Buscar todas las secciones que dependen de la cambiada
const { data: dependents } = await supabase
.from('section_dependencies')
.select('section_id, impact_weight')
.eq('depends_on_section_id', changedSectionId);

if (!dependents || dependents.length === 0) return;

// Marcar para revisión según peso
const sectionsToMark = dependents
.filter(d => changeType === 'major' || d.impact_weight > 7)
.map(d => d.section_id);

await supabase
.from('document_sections')
.update({
needs_review: true,
review_reason: Cambio ${changeType} en sección dependiente
})
.in('id', sectionsToMark);

return sectionsToMark;
};
```

Panel de Salud del Documento

Tu UI debe mostrar el estado global:

```
┌─────────────────────────────────────────────────┐
│  SALUD DEL DOCUMENTO                            │
├─────────────────────────────────────────────────┤
```



---

4. Metodología Evolutiva: "Arquitecto Evolutivo de Sistemas"

Rol del Sistema

Tu app actúa como un **Arquitecto Evolutivo**, no como editor básico. Su objetivo es **CONSTRUIR y MANTENER un documento vivo y complejo**.

Ley de Densidad Creciente (Anti-Resumen)

- **Estrictamente prohibido** eliminar información a menos que sea redundante
- Al "mejorar" una sección, la métrica de éxito es aumentar granularidad:
  - ✖ *Malo:* "La idea es escalable"
  - ✓ *Bueno:* "La idea es escalable mediante microservicios con Kubernetes, CDN global en 4 regiones, y caché distribuido con Redis"

Metodología "Ladrillo a Ladrillo" (Bottom-Up)

Construye desde los cimientos:

1. **Define los Átomos** (Datos, tablas, definiciones concretas)
2. **Define los Componentes** (Párrafos, explicaciones, relaciones)
3. **Integra en la Estructura** (Capítulos, secciones, flujos)

Nunca decores el tejado si no hay paredes.

Gestión del Límite de Output (8k)

Tienes un límite físico de escritura. Si un cambio afecta múltiples partes:

Paso 1: Llama a updateSection para la Sección 1
↓ Sistema procesa y confirma
Paso 2: Llama a updateSection para la Sección 10
↓ Sistema procesa y confirma
Paso 3: En siguiente turno, Sección 15

Divide y vencerás. Actúa como un cursor que se mueve por el documento parcheando zonas.

Protocolo de Integridad Referencial

Si el usuario cambia una premisa fundamental (ej: "Ahora el usuario target son niños"):

1. Revisa la Tabla de Contenidos
2. Identifica TODAS las secciones que dependen de "Usuario Target"
3. Propón un plan en cascada: "Detecto que esto afecta a UX, Copys, Legal. Procedo a actualizar..."

---

5. Estructura de Archivos Recomendada

En lugar de un único documento gigante, organiza tu proyecto como **Repositorio de Código**:

```
/mi-proyecto-ideas/
├── 00_INDICE_MAESTRO.md ← Tabla de contenidos + mapa de dependencias
├── 01_VISION.md ← Visión y objetivos
├── 02_MERCADO.md ← Análisis de mercado
├── 03_MONETIZACION.md ← Modelos de ingresos
├── 04_TECNICA/
│ ├── 04.1_ARQUITECTURA.md
│ ├── 04.2_RIESGOS_TECN.md
│ └── 04.3_TIMELINE.md
├── 05_RIESGOS.md ← Análisis de riesgos
└── 99_CHANGELOG.md ← Historial de versiones
```

Ventaja: La IA lee TODO (contexto global de 1M tokens) pero solo edita archivos pequeños (evitando límite de 8k).

---

6. Resumen: Cuántas API Calls Necesitas

Escenario	Sin Sistema	Con Sistema
Cambio menor en 1 sección	1 + revisión manual	1 call
Cambio mayor que afecta 5 secciones	6 calls inmediatas	1 call + 2 batch después
Usuario ignora revisiones	Documento inconsistente	Flags guardados, revisa cuando quiera
Cambio en raíz del árbol	10+ calls	1 + batch de 3-4 calls

---

7. Checklist de Implementación

✅ **Fase 1: Backend**

- [ ] Nueva Edge Function: `lux-edit-global` (envía doc completo)
- [ ] Parser de output comprimido: `parseCompressedOutput()`
- [ ] Aplicador de cambios: `applyCompressedEdits()`

✅ **Fase 2: Grafo de Dependencias**

- [ ] Tabla `section_dependencies` en Supabase
- [ ] Función de auto-detección: `detectDependencies()`
- [ ] Función de propagación: `propagateChange()`

✅ **Fase 3: UI**

- [ ] Botón "EDITAR GLOBALMENTE" en editor
- [ ] Banner de propagación con estado de secciones
- [ ] Panel de Salud del Documento

✅ **Fase 4: Refinamiento**

- [ ] Lógica de "revisión batch" (máximo 3 secciones por call)
- [ ] Validación de coherencia global
- [ ] Métricas de cambio (deltas de caracteres y palabras)

---

8. Ejemplo Práctico Completo

Usuario: "Mejora la coherencia terminológica. Veo que mezclamos 'inteligencia artificial' con 'IA' con 'machine learning'"

Flujo:

1. **ENVÍO:**
   - Documento: 100,000 caracteres (completo) ✓
   - Estructura: 2,000 caracteres
   - Instrucción: 500 caracteres
   - **Total: ~67,000 tokens / 1,000,000** (bien dentro del límite)
2. **GEMINI ANALIZA (viendo TODO):**
   - Ve que "inteligencia artificial" aparece 45 veces
   - Ve que "IA" aparece 23 veces
   - Ve que "machine learning" aparece 12 veces
   - Entiende contexto: Sección 1 (académica) usa "inteligencia artificial"
   - Sección 2 (técnica) usa "IA"
   - Sección 3 (divulgación) mezcla todo
3. **GEMINI DECIDE (globalmente):**
   - Sección académica: estandariza a "inteligencia artificial"
   - Sección técnica: estandariza a "IA"
   - Sección divulgación: primera "inteligencia artificial", luego "IA"
4. **GEMINI OUTPUT (comprimido a 8k):**
   GLOBAL_ANALYSIS:
   El doc mezcla 3 términos. Tras análisis contextual:

- Cap 1: "inteligencia artificial" = 15 cambios
- Cap 2: "IA" = 8 cambios
- Cap 3: primera "inteligencia artificial", luego "IA"

SECTIONS_TO_MODIFY:

SECTION_ID: [cap_1_intro]

OPERATION: OP_ACTUALIZAR
BEFORE:
"Machine learning es una rama de IA. El ML permite..."
AFTER:
"La inteligencia artificial es una rama del conocimiento...
La inteligencia artificial permite..."
END_SECTION
[... más secciones ...]

5. **TU APP APLICA:**
   - Parsea 18 cambios de 8k tokens
   - Los aplica quirúrgicamente al documento de 100k
   - Valida coherencia global
   - Guarda versión v2.0

Resultado: Documento de 100k caracteres, perfecto y coherente, editado globalmente en 1 API call.

---

9. Próximos Pasos

1. **Implementa Smart Diff primero** → Máximo impacto, mínimo desarrollo
2. **Añade tabla de dependencias** → Visibilidad de impactos
3. **Panel de Salud del Documento** → Control visual del usuario
4. **Integra propagación inteligente** → Automatización de revisiones

La clave: **Gemini VE TODO, pero OUTPUT COMPRIMIDO**. Esto te permite cambios globalmente coherentes con mínimas API calls.