

**DMC**



# **DOCUMENTO MAESTRO DE CONTEXTO (DMC)**

**LeadBoostAI – Versión Conceptual 1.0**

**Modo Híbrido (CTO + Arquitecto Enterprise)**

**Autor: Alfred**

**Destinatario: Alejandro (Fundador / Director)**

# CAPITULO 1



# CAPÍTULO 1 — FUNDAMENTO DEL SISTEMA

## 1. Propósito Supremo del Producto

LeadBoostAI no es un SaaS.

No es una herramienta.

No es un CRM.

No es un analizador de campañas.

**LeadBoostAI es un Sistema Operativo Empresarial Autónomo**, capaz de:

- comprender un negocio,
- analizar datos internos y externos,
- detectar oportunidades,
- planear acciones estratégicas,
- ejecutarlas en plataformas reales (Ads, CRM, Email, Workflows),
- medir impacto,
- aprender de los resultados,
- optimizar sin intervención humana.

El objetivo no es “automatizar marketing”.

El objetivo es:

**Construir un sistema que piense, actúe y mejore como un operador humano experto —pero con velocidad de máquina.**

---

## 2. Filosofía del Sistema (Capa CTO)

LeadBoostAI opera bajo cinco principios:

### 2.1. Decisión autónoma

El sistema debe ser capaz de tomar decisiones estratégicas sin supervisión directa.

No ejecuta tareas aisladas:  
**opera ciclos completos de negocio.**

## 2.2. Explicabilidad interna

Cada acción debe tener:

- causa,
- contexto,
- justificación,
- evidencia,
- trazabilidad.

Nada ocurre sin explicación.

## 2.3. Integración profunda con plataformas externas

El sistema debe ser capaz de actuar en:

- Google Ads
- Meta Ads
- Email
- CRM
- Automatización comercial
- Workflows
- Integraciones adicionales (Actuator+)

## 2.4. Aprendizaje continuo

LeadBoostAI aprende de:

- sus aciertos,
- sus errores,

- el mercado,
- los leads,
- el desempeño comercial,
- el comportamiento humano.

## 2.5. Operatividad empresarial

Todo debe poder escalar a:

- múltiples equipos
  - múltiples clientes
  - múltiples industrias
  - múltiples canales
  - múltiples fuentes de verdad
- 

## 3. Macro-Arquitectura del Sistema (Capa Arquitecto Enterprise)

El sistema se compone de **tres anillos funcionales**:

---

### ● ANILLO 1 — Capa Cognitiva (Inteligencia)

Microservicios que piensan, evalúan, analizan y proyectan.

Incluye:

- **microservice\_analyst** → percepción, análisis contextual
- **microservice\_optimizer** → planeación, scoring, decisión
- **microservice\_memory** → almacenamiento vectorial + contexto semántico
- **microservice\_enterprise** → modelado del negocio y reglas globales

- **microservice\_scout** → descubrimiento, scraping, adquisición de señales

Estos cinco componentes forman **la mente del sistema**.

---

## ● **ANILLO 2 — Capa de Orquestación (Dirección Ejecutiva)**

Diseñada para asegurar que el sistema opere en ciclos completos:

Incluye:

- **backend core** (API central)
- **SAGAs internas** (workflow determinista)
- **Mensajería resiliente**
- **Puentes de comunicación inter-servicio**
- **Interpretabilidad**

Aquí viven:

- reglas de gobernanza,
  - contratos entre dominios,
  - invariantes globales.
- 

## ● **ANILLO 3 — Capa de Ejecución (Actuación Operativa)**

Responsable de transformar decisiones en acciones reales:

- **microservice\_actuator**
- **microservice\_actuator\_plus**
- **Integraciones con plataformas externas**

Este anillo interactúa con APIs externas, automatiza pasos humanos y ejecuta instrucciones de manera segura y trazable.

---

## 4. Dominios del Sistema (Diseño por Separación Vertical)

Cada microservicio pertenece a un dominio funcional:

### Dominio Analyst

Resolver: “*¿Qué está pasando?*”

Responsable de percepción, análisis, correlación, síntesis.

### Dominio Memory

Resolver: “*¿Qué significado tiene esto?*”

Base vectorial para razonamiento contextual.

### Dominio Optimizer

Resolver: “*¿Qué debemos hacer?*”

Motor de planeación estratégica.

### Dominio Enterprise

Resolver: “*¿A qué reglas responde el negocio?*”

Centro de políticas, permisos y modelos económicos.

### Dominio Scout

Resolver: “*¿Qué señales externas importan?*”

Recolecta información del mercado, web, leads y fuentes externas.

### Dominio Actuator

Resolver: “*¿Cómo lo ejecutamos?*”

Capa muscular del sistema.

### Dominio Backend

Resolver: “*¿Cómo nos comunicamos con humanos y sistemas?*”

API, UI, autenticación, IAM, servicios de orquestación.

---

## 5. Identidad del Sistema (Qué Hace a LeadBoostAI Único)

## **5.1. Cognición y Acción integradas**

No es solo analítico (como BI).

No es solo operacional (como Zapier).

No es solo predictivo (como ML aislado).

LeadBoostAI combina:

- Análisis,
- Planeación,
- Ejecución,
- Aprendizaje.

## **5.2. Mentalidad de Operador Senior**

El sistema opera como un humano experto:

- interpreta señales
- detecta oportunidades
- selecciona acciones
- evalúa resultados
- ajusta estrategias
- aprende

## **5.3. Enfoque Enterprise real**

El diseño soporta:

- múltiples clientes
- múltiples unidades de negocio
- múltiples flujos simultáneos
- auditoría extrema
- cumplimiento

- resiliencia ante fallos
  - modularidad para escalar
- 



## CAPÍTULO 1 completado.

Este capítulo es la **columna vertebral conceptual**.

Define:

- qué es,
- por qué existe,
- cómo piensa,
- qué dominios lo conforman,
- cómo se concibe desde alto nivel.

# CAPITULO 2

# CAPÍTULO 2 — MAPA ARQUITECTÓNICO COMPLETO DEL SISTEMA

LeadBoostAI – DMC v1.0

Capítulo central del sistema

---

## 2.1. Arquitectura General (Topología de Alto Nivel)

LeadBoostAI se organiza en **tres capas operativas**:

---

### Capa Cognitiva (Inteligencia)

Microservicios que piensan, entienden, infieren, predicen y planean.

Incluye:

- `microservice_analyst`
- `microservice_memory`
- `microservice_optimizer`
- `microservice_scout`
- `microservice_enterprise`

Esta capa responde a:

- ¿Qué está pasando?

- ¿Qué significa?
  - ¿Qué debemos hacer?
  - ¿Qué impacto tiene en el negocio?
  - ¿Qué oportunidades existen?
- 



## Capa de Orquestación (Dirección Ejecutiva)

Garantiza que el sistema opere de forma determinista y confiable:

Incluye:

- `backend/` (API core)
- `src/` (frontend + SAGA UI + messaging UI)
- mensajería resiliente (Kafka o equivalentes conceptuales)
- corredores de eventos
- Workflows SAGA (en microservicios)
- Actores de larga duración
- Mecanismos de correlación
- Garantías de idempotencia

Esta capa garantiza:

- orden
- consistencia
- trazabilidad
- estabilidad en caso de fallos
- ejecución correcta de SAGA
- integración con humanos



## Capa de Ejecución (Actuación Operativa)

Transforma decisiones en acciones reales:

Incluye:

- `microservice_actuator`
- `microservice_actuator_plus`
- conectores a plataformas externas
- drivers de acción
- validación de resultados

Esta capa responde a:

- “Hazlo.”
  - “Ejecuta esta acción en Google Ads.”
  - “Lanza este email.”
  - “Modifica el presupuesto.”
  - “Dispara este evento.”
- 



## 2.2. Mapa de Microservicios (Vista Detallada)

A continuación tienes la definición **oficial** de cada microservicio y su rol dentro del sistema.

Esta es la guía que las IA usarán para comprender el sistema completo.

---

# MICROSERVICE: ANALYST

*Dominio: Percepción y Comprensión del Entorno*

## **Responsabilidades:**

- Recibir señales crudas (datos, leads, campañas, comportamiento).
- Procesar información inicial.
- Clasificar, normalizar y contextualizar señales.
- Generar un *State Vector* del negocio.
- Detectar anomalías, patrones o eventos significativos.

## **Principio central:**

“Entiende antes de decidir.”

---

# MICROSERVICE: MEMORY

*Dominio: Conocimiento, Contexto Persistente y Vectorización*

## **Rol:**

La base cognitiva del sistema.

## **Responsabilidades:**

- Almacenar embeddings, contexto conversacional y estado semántico.
- Proveer búsqueda vectorial.
- Mantener memoria de largo plazo.
- Permitir aprendizaje continuo.

## **Principio central:**

“Nada se pierde. Todo se recuerda y se relaciona.”



## MICROSERVICE: OPTIMIZER

**Dominio:** *Planeación y Toma de Decisiones*

**Responsabilidades:**

- Generar planes estratégicos.
- Evaluar múltiples opciones.
- Realizar scoring de alternativas.
- Emitir decisiones formales.
- Priorizar acciones recomendadas.

**Principio central:**

“Decide inteligentemente, no automáticamente.”

---



## MICROSERVICE: ENTERPRISE

**Dominio:** *Gobernanza, Reglas del Negocio y Políticas Globales*

**Responsabilidades:**

- Representar la lógica de negocio de alto nivel.
- Mantener reglas, límites, permisos, políticas.
- Orquestar modelos, simulaciones y escenarios.
- Regular el comportamiento del sistema.
- Validar que las decisiones respeten parámetros empresariales.

**Principio central:**

**“Lo que decide el sistema debe respetar al negocio.”**

---



## MICROSERVICE: SCOUT

**Dominio: Búsqueda y Adquisición de Información Externa**

### Responsabilidades:

- Scraping
- Crawlers
- Recolección externa
- Descubrimiento de oportunidades
- Monitoreo de mercado
- Identificación de señales valiosas

### Principio central:

**“Encuentra oportunidades que el humano no ve.”**

---



## MICROSERVICE: ACTUATOR

**Dominio: Ejecución Operativa Real**

### Responsabilidades:

- Traducir órdenes del Optimizer / Enterprise en acciones específicas.
- Conectarse con APIs externas.
- Implementar comandos operativos (Ads, CRM, Email...).
- Garantizar éxito, reintentos e idempotencia.

- Confirmar ejecución.

### **Principio central:**

**“Acción sin riesgo, ejecución sin ambigüedad.”**

---



## **MICROSERVICE: ACTUATOR PLUS**

### **Dominio: Ejecución Avanzada y Automatización Profunda**

Extiende Actuator para operaciones más complejas:

- Workflows multi-paso
- Cambios de estrategia en tiempo real
- Ajustes operativos dinámicos
- Integraciones personalizadas

### **Principio central:**

**“Automatización completa del negocio.”**

---



## **MICROSERVICE: BACKEND**

### **Dominio: API Central, Seguridad, IAM y Orquestación Interna**

#### **Responsabilidades:**

- IAM (Identidad y Acceso)
- Gateways para frontend
- API REST/GraphQL
- Validación y autenticación

- Comunicaciones internas
- Webhooks

### **Principio central:**

**“El backend es la capa de orden y seguridad.”**

---



## **FRONTEND (src/)**

***Dominio: Interfaz Humana y Control Visual***

Incluye:

- páginas
- componentes
- dashboards
- hooks
- messaging UI
- sagas frontend para sincronización

### **Principio:**

**“Visualiza, controla y explica.”**

---



## **2.3. Flujo Cognitivo del Sistema (Mapa de Inteligencia)**

LeadBoostAI opera en ciclos:

1. **Percibir** → Analyst

2. **Interpretar** → Memory
3. **Planear** → Optimizer
4. **Validar reglas** → Enterprise
5. **Ejecutar** → Actuator / Actuator+
6. **Observar resultados** → Analyst
7. **Aprender** → Memory
8. **Reajustar estrategia** → Optimizer

Este ciclo ocurre **millones de veces por cliente**, de manera:

- autónoma
  - trazable
  - segura
  - auditabile
- 



## 2.4. Flujo de Datos Interno (Arquitectura Técnica)

### Entrada:

Eventos del negocio, señales, datos externos, comportamiento.

### Procesamiento:

Analyst → Memory → Optimizer → Enterprise.

### Salida:

Acciones operativas vía Actuator.

### Retroalimentación:

Actuator → Analyst → Memory.

---

## 2.5. Comunicación Entre Microservicios (Contratos)

Se hace mediante:

- eventos
- colas
- mensajería resiliente
- comandos explícitos
- respuestas idempotentes
- workflows SAGA

Nada ocurre por acoplamiento directo.

---



## CAPÍTULO 2 COMPLETADO

Este es el **mapa completo del sistema**.

Con esto, cualquier IA puede entender:

- qué hace cada microservicio,
- cómo encaja,
- cómo fluye la información,
- qué reglas gobiernan el sistema,
- cómo se orquesta el pensamiento completo.

# CAPITULO 3

# CAPÍTULO 3 — ORQUESTACIÓN, SAGA, MENSAJERÍA Y GARANTÍAS DEL SISTEMA

LeadBoostAI – DMC v1.0

---



## 3.1. Principio rector de esta capa

**“El sistema nunca debe perder un evento, ejecutar dos veces la misma acción, ni avanzar un flujo sin evidencia.”**

Esto significa:

- cero pérdida de mensajes
- cero duplicados peligrosos
- cero avance inconsistente
- cero estados corruptos
- cero acciones sin justificación
- cero ejecuciones fantasma

La capa de orquestación es la **columna vertebral** que hace que LeadBoostAI sea confiable para operaciones reales.

---



## 3.2. Modelo de Mensajería del Sistema

LeadBoostAI usa un modelo inspirado en Kafka + SAGA (aunque el backend actual no tenga Kafka formalizado, la arquitectura conceptual Sí está diseñada así).

**El sistema opera con 3 tipos de mensajes:**

## 1. Commands

*“Haz esto.”*

Ejemplo:

- enviar\_email
- crear\_campaña
- analizar\_lead
- iniciar\_simulación

Los commands **siempre generan side effects** → Actuator o capa externa significa ejecución real.

---

## 2. Events

*“Esto ocurrió.”*

Ejemplo:

- lead\_calificado
- análisis\_completado
- campaña\_creada
- ejecución\_exitosa

Evento = hecho consumido por múltiples microservicios.

---

## 3. State Updates

*“Actualiza tu modelo mental.”*

Ejemplo:

- memory\_update
- cambio\_en\_contexto
- reevaluación

Esto alimenta al microservice\_memory, optimizer, analyst.

---



## 3.3. El Ciclo de Control SAGA (versión empresarial)

Una SAGA describe **un flujo de múltiples pasos**, donde cada paso:

- tiene una acción,
- un evento de confirmación,
- y una compensación si algo falla.

La estructura general en LeadBoostAI:

STEP 1 – Command: iniciar\_operación  
STEP 1 – Event esperado: operación\_inicializada

STEP 2 – Command: ejecutar\_analista  
STEP 2 – Event esperado: analisis\_completado

STEP 3 – Command: generar\_estrategia  
STEP 3 – Event esperado: estrategia\_generada

STEP 4 – Command: ejecutar\_accion  
STEP 4 – Event esperado: accion\_ejecutada

...

Si cualquier paso falla:

- **COMPENSATE**: revertir o notificar estado de fallo
  - **HALT**: detener flujo
  - **AUDIT**: registrar causa exacta
  - **RETRY**: según reglas del sistema
-



## 3.4. Invariantes que gobiernan TODA la orquestación

Estos son **no negociables**, y Gemini, Codex y Copilot deben operar siempre bajo ellos.

### 3.4.1. Un Command NO puede ejecutarse dos veces.

Garantía:

- idempotencia por key (operation\_id)
  - almacenamiento de estados previos
  - detección de duplicados
- 

### 3.4.2. Un Event NO puede perderse.

Garantía:

- almacenamiento durable antes de confirmar procesamiento
  - control de offset avanzado
  - no commit hasta side-effect completado
- 

### 3.4.3. Una SAGA NO puede avanzar sin su evento correspondiente.

Esto es crítico:

Un flujo NUNCA puede pasar al paso siguiente sin recibir el evento esperado.

Este fue el error que detectaste hace días en Gemini.

---

### 3.4.4. Las compensaciones deben dispararse SIEMPRE ante fallos.

Modelo:

```
command -> event (ok) → advance  
command -> no event (timeout) → compensar  
command -> event fallo → compensar  
command -> error interno → compensar
```

---

### 3.4.5. La mensajería nunca debe depender del frontend.

El sistema funciona ANTE esto:

- si se cae la UI
- si el usuario cierra pestañas
- si hay latencia
- si se pierde conexión

El backend DEBE mantener coherencia solo.

---

## 3.5. Arquitectura de Orquestación (vista componente)

La capa de orquestación se distribuye así:

---

### 3.5.1. Backend → Gateway + Orquestador

- recibe requests
- inicia flujos
- valida IAM
- dispara comandos iniciales

- registra la SAGA
- 

### **3.5.2. Microservicios → Ejecutores de pasos**

Cada microservicio ejecuta uno o más pasos del flujo.

Ejemplo:

- Analyst → genera insights
  - Memory → actualiza estado lógico
  - Optimizer → planifica
  - Actuator → ejecuta acción final
- 

### **3.5.3. Mensajería → Canal de Comunicación**

Eventos y comandos se transmiten por:

- colas internas
  - hooks
  - workers
  - loops de procesamiento
  - estados de SAGA
- 

### **3.5.4. SAGA Store → Memoria del Flujo**

Almacena:

- estado actual
- paso actual

- payload
- event esperado
- estado final
- logs

Es crucial para resiliencia.

---



## 3.6. Modelo de Resiliencia ante Fallos

LeadBoostAI debe soportar fallos sin perder consistencia:

### Si un microservicio falla:

- retry automático
- compensación según regla
- estado preservado

### Si un evento no llega:

- timeout inteligente
- compensación
- log detallado
- no avanzar nunca

### Si Actuator falla en una acción externa:

- reintento con backoff
- si supera política → compensación
- notificación al Enterprise

### Si Memory falla:

- fallback a memoria previa
  - degradación funcional sin perder integridad
-



## 3.7. Garantías de Seguridad y Gobernanza de Flujos

Toda acción:

- tiene un actor
- tiene un propósito
- tiene impacto económico real

Por eso:

### 3.7.1. IAM filtra quién puede iniciar SAGA.

Nadie sin autorización.

### 3.7.2. Todo microservicio firma sus eventos.

### 3.7.3. Ninguna acción externa se ejecuta sin validación de Enterprise.

---



## 3.8. Mecanismo de Correlación y Contexto

Cada flujo tiene:

- `operation_id`
- `context_id`
- `saga_id`

Todos los eventos deben incluirlos.

---



## 3.9. Orquestación Visual (Frontend)

En tu carpeta `src/sagas`, la UI muestra:

- estados
- progreso de SAGA
- eventos recibidos
- pasos ejecutados

Pero:

**Frontend JAMÁS controla el estado real.  
Solo lo visualiza.**

---



## CAPÍTULO 3 COMPLETADO

Este capítulo define:

- cómo se ejecutan flujos reales
- cómo se garantiza resiliencia
- cómo se evitan errores catastróficos
- cómo se articula la SAGA completa
- cómo se piensa y actúa ante fallos
- cómo se comunican microservicios en un sistema vivo

Esto es lo que permite que LeadBoostAI sea confiable *incluso cuando toma decisiones y ejecuta acciones comerciales con impacto económico real.*

# CAPITULO 4

# CAPÍTULO 4 — INVARIANTES DEL SISTEMA

LeadBoostAI – DMC v1.0

Definición de reglas absolutas

---



## 4.1. Naturaleza de los invariantes

Un invariante es:

**“Una condición que siempre debe mantenerse verdadera en todo el sistema, en cualquier momento, sin importar el estado.”**

Si un invariante se rompe, **se detiene la operación**, se compensa, se loggea y se notifica.

Los invariantes sirven para:

- gobernar SAGA
  - definir garantías de consistencia
  - proteger flujos
  - asegurar idempotencia
  - restringir acciones peligrosas
  - validar estados críticos
- 

## 4.2. Invariantes Globales (afectan a todo el sistema)

**4.2.1. Ninguna acción externa puede ejecutarse sin autorización del dominio Enterprise.**

Razón:

Evita decisiones sin contexto financiero o estratégico.

---

#### **4.2.2. Ningún microservicio puede avanzar un paso SAGA sin el evento correspondiente.**

Razón:

Garantiza determinismo.

---

#### **4.2.3. Todo comando debe tener un `operation_id` único por flujo.**

Razón:

Previene duplicados.

---

#### **4.2.4. Todo evento debe contener `saga_id` y `context_id`.**

Razón:

Permite correlación y auditoría.

---

#### **4.2.5. Ningún flujo puede iniciarse sin ser registrado previamente en SAGA Store.**

Razón:

Evita “flujos fantasma”.

---

#### **4.2.6. Ningún recurso puede mutar estado sin un evento que justifique el cambio.**

Razón:

Protección ante corrupción de datos.

---

#### **4.2.7. El sistema nunca puede ejecutar un comando sin verificar idempotencia.**

Razón:

Evita ejecuciones dobles no deseadas.

---

## 4.3. Invariantes de Microservicios

Cada microservicio tiene reglas internas para evitar errores lógicos.

---

### 4.3.1. Analyst

- Ninguna inferencia puede emitirse sin datos suficientes.
  - Ningún insight puede ser contradictorio con el estado actual.
  - Toda señal debe ser normalizada antes de ser propagada.
- 

### 4.3.2. Memory

- No puede borrar contexto sin respaldo.
  - No puede sobrescribir memoria previa sin relación semántica.
  - Ningún embedding puede guardarse si no pasa validación dimensional.
- 

### 4.3.3. Optimizer

- No puede tomar decisiones sin análisis previo.
  - No puede emitir un plan sin tener alternativas evaluadas.
  - Toda decisión debe tener un score justificable.
- 

### 4.3.4. Enterprise

- No puede permitir acciones fuera de políticas.
  - No puede aceptar decisiones inconsistentes con el modelo económico.
  - Toda regla debe ser explícita, no implícita.
- 

#### **4.3.5. Scout**

- No puede duplicar scraping de la misma fuente dentro del mismo contexto.
  - No puede generar señales falsas o incompletas.
  - No puede saturar endpoints de terceros.
- 

#### **4.3.6. Actuator / Actuator+**

- No pueden repetir una acción externa si ya fue confirmada.
  - No pueden ejecutar acciones sin validar límites de riesgo.
  - Deben reportar resultado inmediatamente.
- 



### **4.4. Invariantes de Orquestación**

Estos son críticos para evitar fallos catastróficos:

---

#### **4.4.1. Ninguna SAGA puede tener dos pasos activos simultáneamente.**

---

#### **\*\*4.4.2. Una SAGA no puede finalizar sin:**

- acción completada

- compensación ejecutada
- o rollback total

finalización = success OR compensated OR rollback

---

#### **4.4.3. Todos los timeouts deben estar definidos.**

Nunca un flujo debe esperar indefinidamente.

---

#### **4.4.4. El estado de una SAGA debe ser determinista.**

Dos corridas con las mismas entradas => mismo resultado.

---

#### **4.4.5. Si un paso falla, se ejecuta compensación inmediatamente.**

---



## **4.5. Invariantes de Seguridad**

Tu sistema manipula datos reales y ejecuta acciones con impacto económico.

Por eso:

---

#### **4.5.1. Ningún microservicio debe aceptar input sin sanitización.**

---

#### **4.5.2. Ningún comando de Actuator puede ejecutarse sin IAM.**

---

#### **4.5.3. Ningún token externo debe guardarse en texto plano.**

---

#### **4.5.4. Ninguna acción crítica puede ocurrir sin logging.**

---

## 4.6. Invariantes de Aprendizaje

Para mantener coherencia del sistema cognitivo:

---

**4.6.1. Memory nunca puede aprender de un estado parcial.**

---

**4.6.2. Optimizer no puede entrenar modelos sobre datos inconsistentes.**

---

**4.6.3. Analyst no puede generar insights basados en ruido o datos corruptos.**

---

## 4.7. Invariantes de Frontend (UI / UX)

Aunque la UI no controla el sistema:

- Debe reflejar estados reales.
  - Nunca puede mostrar pasos no ejecutados.
  - Nunca puede iniciar flujos no permitidos.
  - Nunca puede ignorar reglas de IAM.
- 



## CAPÍTULO 4 – Conclusión

Los invariantes son la **Constitución del Sistema**.

Sin ellos:

- fallan flujos,
- se duplican acciones,
- se pierde dinero,

- se rompe la consistencia,
- el sistema deja de ser confiable.

Con ellos:

- LeadBoostAI puede operar como un sistema autónomo enterprise.

Este capítulo será fundamental cuando:

- Gemini diseña microservicios nuevos
- Copilot reciba instrucciones
- Hagamos auditorías lógicas
- Implementemos Fase 3 (IAM + Seguridad + Políticas + Gobernanza)

# CAPITULO 5

# CAPÍTULO 5 — ARQUITECTURA DE DOMINIOS Y CONTRATOS INTER-SERVICIO

LeadBoostAI – DMC v1.0

Formato C (Arquitectura Enterprise Formal)

---



## 5.1. Propósito del Capítulo

Definir, formalmente, la arquitectura lógica que gobierna las interacciones entre todos los microservicios del sistema, incluyendo:

- **Responsabilidades exactas**
- **Contratos explícitos**
- **Relaciones permitidas**
- **Límites duros (no negociables)**
- **Matrices RACI entre dominios**
- **Formatos de mensajes**
- **Anti-patrones estrictamente prohibidos**

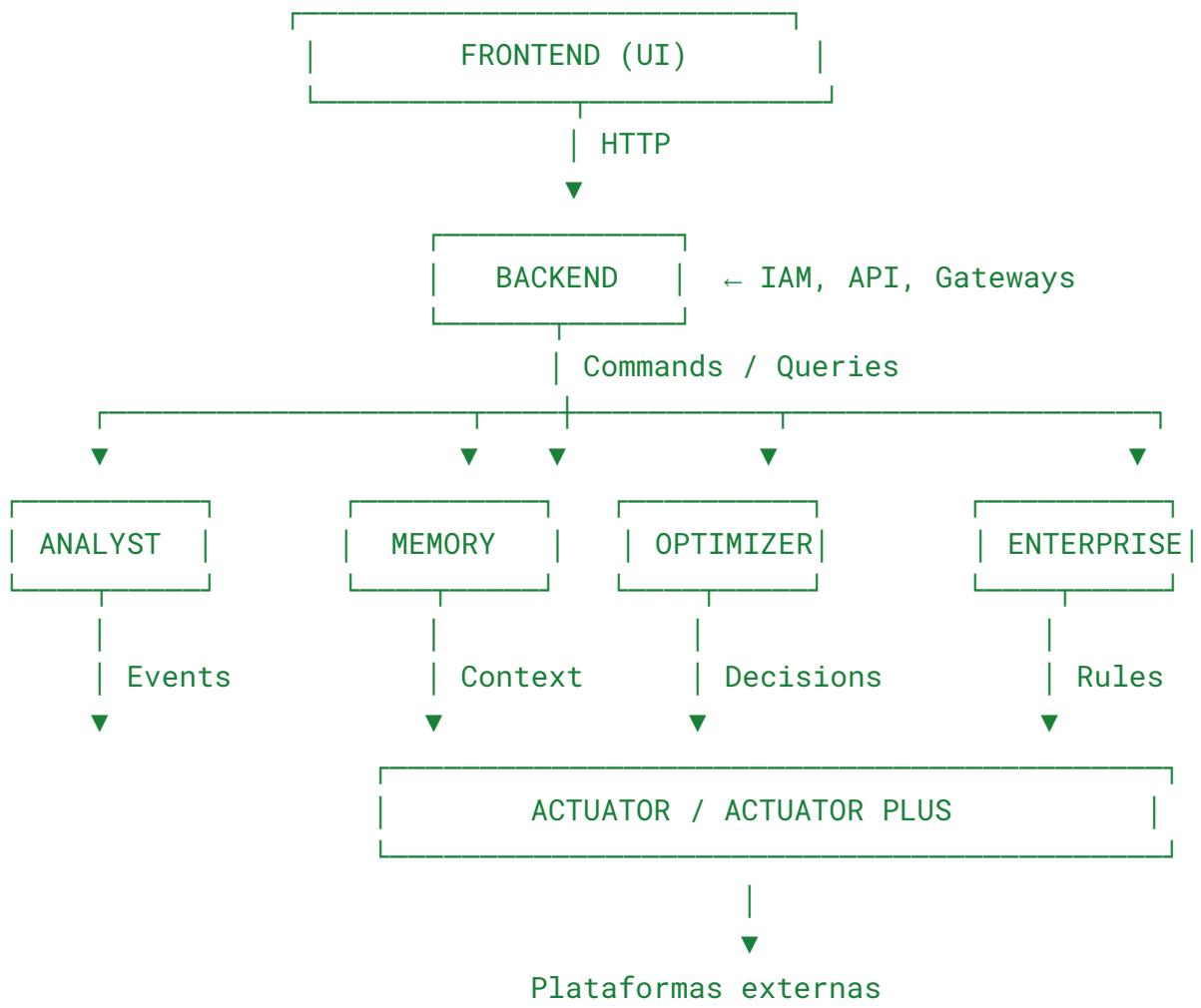
El objetivo es que NINGÚN servicio haga lo que otro debería hacer y que TODAS las IA comprendan claramente los borders del sistema.

---



## 5.2. Mapa Conceptual de Dominios (Topología Formal)

Texto-diagrama:



## 📌 5.3. Matriz RACI Inter-Dominios (Versión Enterprise)

**R = Responsible**

**A = Accountable**

**C = Consulted**

**I = Informed**

Dominio	Análisis	Memoria	Decisión	Reglas	Ejecución	UI	Backend
Analyst	R,A	C	I	I	I	I	C

<b>Memory</b>	C	R,A	C	I	I	I	C
<b>Optimizer</b>	C	C	R,A	C	I	I	C
<b>Enterprise</b>	I	I	C	R,A	C	I	C
<b>Actuator</b>	I	I	I	C	R,A	I	C
<b>Backend</b>	I	I	I	I	I	C	R,A
<b>Frontend</b>	I	I	I	I	I	R,A	C

Esta tabla es crucial para:

- evitar “microservicios que deciden cosas”,
  - evitar duplicidades,
  - saber EXACTAMENTE quién manda en cada etapa del flujo.
- 

## 5.4. Contratos formales (SPEC) de cada dominio

Cada dominio tiene:

- **Mandato (qué hace)**
  - **Superficie pública (qué expone)**
  - **Entradas permitidas (inputs)**
  - **Salidas garantizadas (outputs)**
  - **Interacciones autorizadas**
  - **Limitaciones explícitas**
  - **Anti-patrones prohibidos**
-



## 5.4.1. Dominio ANALYST — SPEC FORMAL

### Mandato

Transformar señales crudas en información estructurada y comprensible.

### Superficie pública

Endpoints/eventos:

- `analyze.lead`
- `analyze.campaign`
- `analyze.performance`
- `event.insights.ready`

### Entradas (inputs)

Tipo	Nombre	Descripción
Event	<code>lead.created</code>	Nuevo lead ingresado
Event	<code>campaign.update</code>	Cambio en métricas d
Command	<code>analyze.subject</code>	Solicitud explícita de análisis

### Salidas (outputs)

- `insights.generated`
- `patterns.detected`
- `anomaly.detected`

### Interacciones autorizadas

Destino	Motivo
---------	--------

Memory      recuperar contexto

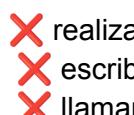
Optimizer    entregar insights  
procesados

Enterprise    alertas de riesgo

## Limitaciones

- No genera decisiones.
- No ejecuta acciones.
- No modifica reglas.

## Anti-patrones prohibidos



- realizar planeación
- escribir en bases de datos externas
- llamar a Actuator



## 5.4.2. Dominio MEMORY — SPEC FORMAL

### Mandato

Mantener el contexto semántico histórico del sistema.

### Superficie pública

- `memory.store`
- `memory.retrieve`
- `memory.query.similar`

### Entradas

Embeddings, snapshots, texto, metadata.

## Salidas

Contexto relacionado, similitudes, inferencias.

## Interacciones autorizadas

Destino	Motivo
Analyst	enriquecer análisis
Optimizer	contexto para decisiones
Enterprise	estado global histórico

## Limitaciones

- No toma decisiones.
- No ejecuta estrategias.
- No muta reglas globales.

## Anti-patrones prohibidos

- ✗ almacenar datos no validados
  - ✗ vectorizar basura o ruido
  - ✗ comunicarse con Actuator
- 

## 5.4.3. Dominio OPTIMIZER — SPEC FORMAL

### Mandato

Convertir insights en estrategias y planes de acción.

### Superficie pública

- `optimizer.plan`
- `optimizer.propose.strategy`

- `optimizer.simulate`

## Entradas

- Insights
- Contexto
- Reglas del negocio

## Salidas

- `strategy.proposed`
- `actionplan.generated`

## Interacciones autorizadas

Destino	Motivo
Enterprise	validar estrategias
Analyst	pedir análisis adicional
Backend	enviar planes formalizados (no acciones directas)

## Limitaciones

- No ejecuta acciones.
- No salta reglas.
- No comunica a plataformas externas.

## Anti-patrones prohibidos

- ✗ “decidir y ejecutar” en el mismo paso
  - ✗ escribir en bases de ejecución
  - ✗ ignorar políticas de Enterprise
-



## 5.4.4. Dominio ENTERPRISE — SPEC FORMAL

### Mandato

Ser el árbitro, juez y regulador del sistema.

### Superficie pública

- `enterprise.validate.strategy`
- `enterprise.approve`
- `enterprise.reject`
- `enterprise.adjust`

### Entradas

- Estrategias propuestas
- Políticas del negocio
- Restricciones

### Salidas

- `strategy.approved`
- `strategy.rejected`
- `risk.flagged`

### Interacciones autorizadas

Destino	Motivo
---------	--------

Optimizer devolver decisiones

Actuator autorizar ejecución

Backend      exponer estado  
                empresarial

## Limitaciones

- No ejecuta acciones.
- No analiza datos.
- No genera embeddings.

## Anti-patrones prohibidos

- ✗ sobrescribir reglas sin trazabilidad
  - ✗ aprobar estrategias contradictorias
  - ✗ ignorar anomalías reportadas por Analyst
- 

## 5.4.5. Dominio SCOUT — SPEC FORMAL

### Mandato

Descubrir nuevas señales externas.

### Interacciones autorizadas

- Analyst
- Memory

### Anti-patrones

- ✗ hablar con Optimizer
  - ✗ hablar con Actuator
-



## 5.4.6. Dominio ACTUATOR — SPEC FORMAL

### Mandato

Materializar decisiones en plataformas reales.

### Entradas

- `execution.command`
- `strategy.approved`

### Salidas

- `execution.success`
- `execution.failure`

### Limitaciones

- No decide.
- No analiza.
- No planea.

### Anti-patrones

- ✗ ejecutar sin autorización
  - ✗ ignorar fallos
  - ✗ modificar reglas
- 

## 📌 5.5. Contratos Inter-Servicio (matriz formal)

Tabla reducida (la completa se guarda para el Apéndice del DMC):

Origen → Destino	Permitido	Tipo	Descripción
Analyst → Memory	✓	Query	Solicita contexto
Analyst → Optimizer	✓	Event	Enviar insights
Optimizer → Enterprise	✓	Proposal	Validar estrategia
Enterprise → Actuator	✓	Approval	Autorizar ejecución
Actuator → Analyst	✓	Event	Reportar resultados
Frontend → Backend	✓	HTTP	Comandos UI
Backend → microservicios	✓	Command	Orquestación

Ningún otro flujo está permitido.

---

## ⭐ 5.6. Anti-patrones prohibidos (lista dura)

Los siguientes comportamientos son **violaciones graves a la arquitectura**:

- **acoplamientos directos entre microservicios**
- **Optimizer ejecutando acciones**
- **Actuator generando estrategias**
- **Frontend tomando decisiones de negocio**
- **Memory respondiendo con datos corruptos**
- **Enterprise aprobando acciones sin contexto**
- **Analyst siendo usado como "cajón de sastre"**
- **Backend guardando lógica empresarial**

Cada uno de ellos destruye la escalabilidad del sistema.

---



## 5.7. Glosario formal de tipos de mensajes

### Commands

Son órdenes. Siempre generan side effects.

### Events

Hechos consumibles por múltiples dominios.

### Queries

Solicitudes de datos sin efectos secundarios.

### Approvals

Autorizaciones formales desde Enterprise.

---



## 5.8. Conclusión del Capítulo 5

Ahora el sistema tiene:

- contratos formales
- responsabilidades exactas
- límites duros
- matrices RACI
- interacciones autorizadas
- glosario formal
- topología lógica enterprise

Este capítulo es lo que usará Gemini para blueprint, Codex para indexación y Copilot para implementación sin errores.

# CAPITULO 6



# CAPÍTULO 6 — ESTRUCTURA DEL REPOSITORIO Y MAPA DE ARCHIVOS CLAVE

LeadBoostAI — DMC v1.0

Formato enterprise (C)

---



## 6.1. Propósito del capítulo

Este capítulo define:

- la estructura lógica del repositorio,
- el significado de cada carpeta,
- cómo se relaciona esa estructura con los dominios del sistema,
- qué archivos son “core”,
- qué está permitido en cada nivel,
- qué está prohibido (anti-patrones),
- cómo deben leer esta estructura las IA.

Sirve para eliminar:

- ambigüedad,
- desorden,
- duplicidad de responsabilidades,
- ubicación incorrecta de lógica,
- contaminación entre componentes.



## 6.2. Mapa General del Repositorio (Vista Bird-Eye)

El repositorio de LeadBoostAI (rama `lite-contexto-ia`) está organizado así:

```
/  
└── backend/  
└── microservice_actuator/  
└── microservice_actuator_plus/  
└── microservice_analyst/  
└── microservice_enterprise/  
└── microservice_memory/  
└── microservice_optimizer/  
└── microservice_scout/  
└── src/  
└── scripts/  
└── config/  
└── docs/  
└── blue_prints/
```

Esta estructura es **alta calidad enterprise**:  
separación por dominios, capas claras, estructura escalable.

---



## 6.3. Estructura de Alto Nivel por Capas

El repositorio se divide en tres capas fundamentales:

Capa 1 – UI y Frontend	→ <code>src/</code>
Capa 2 – Orquestación/API	→ <code>backend/</code>
Capa 3 – Dominios Cognitivos	→ <code>microservice_*</code>
Capa 4 – Documentación	→ <code>docs/</code> , <code>blue_prints/</code>

Cada una cumple un rol específico.



## 6.4. Capa UI / Frontend (src/)

### Propósito

Interfaz visual que interactúa exclusivamente con `backend/`.

### Subcarpetas clave

Carpeta	Propósito
<code>component</code>	Componentes UI reutilizables
<code>s/</code>	
<code>pages/</code>	Vistas completas del sistema
<code>hooks/</code>	Lógica de interacción con backend
<code>messaging</code>	Mensajería visual, estado de SAGA, loadings
<code>/</code>	
<code>sagas/</code>	Implementación UI de flujos (solo visual/UX)
<code>styles/</code>	Estilos globales o temáticos
<code>utils/</code>	Funciones auxiliares UI

### Reglas duras

- ✗ Nada de lógica empresarial.
  - ✗ Nada de ejecución de acciones reales.
  - ✗ Nada de modelado de dominio.
  - ✓ Únicamente UI y lógica visual.
- 



## 6.5. Capa de Orquestación / API (`backend/`)

Esta capa es el “punto de entrada oficial” al sistema.

### Subcarpetas clave

Carpeta	Propósito
config/	configuración del backend
core/	lógica de infraestructura y orquestación
routes/	endpoints expuestos al frontend
microservice_	gateway entre UI y microservicios
bff/	
src/	subdominios internos del backend (controladores, repositorios, core)

### Reglas duras

- ✓ Autenticación / IAM
- ✓ Validación de inputs
- ✓ Orquestación inicial de SAGA
- ✗ Sin lógica empresarial (eso es Enterprise)
- ✗ Sin procesamiento intensivo (eso es Analyst/Optimizer)
- ✗ Sin acciones externas (Actuator)

El backend coordina, no piensa.

---

## 6.6. Capa Cognitiva / Dominios Internos (`microservice_*`)

Cada carpeta `microservice_X/` representa un dominio completo del sistema.

A continuación se define su estructura interna obligatoria.

---

## 6.6.1. Estructura estándar de un microservicio

Todos los microservicios deben seguir este patrón:

```
microservice_X/
└── api/ or handlers/      → entradas del servicio
└── core/                  → lógica principal del dominio
└── models/                 → tipos, entidades y estructuras
└── services/ or interfaces/ → componentes auxiliares
└── __pycache__/            → ignorado siempre
```

### Reglas duras

- ✓ `core/` contiene el cerebro del dominio
  - ✓ `api/ o handlers/` define cómo se expone
  - ✓ `models/` contiene entidades puras (sin lógica pesada)
  - ✗ nada en un microservicio debe saber detalles internos de otro
  - ✗ no compartir código entre dominios sin un módulo dedicado
- 



## 6.6.2. Microservice — ACTUATOR

Ubicación: `microservice_actuator/`

### Subcarpetas relevantes:

- `handlers/platforms` → conectores a plataformas externas
- `interfaces` → contratos para ejecuciones
- `models` → definiciones de acciones
- `core/critics` → validación y análisis previo a ejecución

### **Anti-patrones prohibidos:**

-  lógica empresarial
  -  decisiones estratégicas
  -  llamadas directas a Memory/Optimizer
- 



## **6.6.3. Microservice — ACTUATOR PLUS**

Igual que Actuator pero para flujos avanzados/multi-paso.

---



## **6.6.4. Microservice — ANALYST**

Ubicación: `microservice_analyst/`

### **Enfoque**

- análisis
- síntesis
- preprocesamiento

### **Carpetas clave:**

- `api/routes`
  - `core/`
  - `models/`
  - `services/`
-



## 6.6.5. Microservice — MEMORY

Ubicación: `microservice_memory/`

**Carpetas clave:**

- `chroma_db/` → almacenamiento vectorial real (eliminado en lite branch)
  - `core/` → indexado vectorial, recuperación
  - `models/`
  - `services/`
- 



## 6.6.6. Microservice — OPTIMIZER

Ubicación: `microservice_optimizer/`

**Carpetas clave:**

- `logic/`
  - `api/`
  - `core/`
  - `models/`
  - `tests/`
- 



## 6.6.7. Microservice — ENTERPRISE

Ubicación: `microservice_enterprise/`

**Carpetas clave:**

- `api/`
  - `core/` → restricciones, políticas, modelos económicos
  - `models/`
  - `scenarios/`
- 



## 6.6.8. Microservice — SCOUT

Ubicación: `microservice_scout/`

### Carpetas clave:

- `core/network/`
  - `tests/`
  - `models/`
- 



## 6.7. Capa Documental (docs/, blue\_prints/)

```
docs/
└── advanced_reports/
└── migrations/
└── assets/
└── db_migrations/
└── ...
```

### Propósito

- documentación del sistema

- reportes de fases anteriores
- migraciones SQL
- assets históricos

## Reglas duras

- ✗ nada en docs debe afectar código
  - ✓ es referencia arquitectónica
  - ✓ sirve para IA (Codex/Gemini)
- 

## 6.8. Mapa de archivos críticos del repositorio

Esta tabla marca los archivos que “definen” al sistema:

Archivo/Carpeta	Rol
backend/src/core	corazón del backend
microservice_optimizer/logic	motor de decisión
microservice_memory/core	memoria contextual
microservice_analyst/core	percepción e interpretación
microservice_enterprise/core	reglas del negocio
microservice_actuator/handlers	conexión a plataformas externas
src/messaging/scripts/	interfaz visual de flujos herramientas de despliegue/mantenimiento

---

## 6.9. Anti-patrones estructurales prohibidos

Estas violaciones destruyen proyectos enterprise:

- ✗ Lógica empresarial en frontend
- ✗ Lógica de decisión en backend
- ✗ Actuator modificando reglas
- ✗ Optimizer ejecutando acciones
- ✗ Memory tomando decisiones
- ✗ Mezclar carpetas por tecnología en vez de por dominio
- ✗ Microservicios compartiendo modelos o código directamente
- ✗ Archivos “god module” con demasiadas responsabilidades
- ✗ Documentación crítica fuera de `docs/`

Estas reglas protegen tu arquitectura.

---

## 6.10. Cómo debe interpretar esta estructura una IA

### Gemini

- usa este capítulo para comprender el repositorio en contexto empresarial.
- sabe dónde leer, agrupar, sintetizar y auditar código.

### Codex

- usa esto como “mapa semántico” para documentar y generar índices.

## Copilot

→ sabe dónde generar código y dónde NO insertar lógica por error.

---



## 6.11. Conclusión del Capítulo 6

Ahora tu sistema tiene:

- **estructura formalizada,**
- **un mapa de carpetas con roles claros,**
- **un modelo enterprise estable,**
- **puntos críticos identificados,**
- **anti-patrones prohibidos,**
- **uso correcto para IA.**

Este capítulo convierte tu repositorio en un entorno **legible por máquinas y por humanos a nivel senior**.

# CAPITULO 7



# CAPÍTULO 7 — ESTÁNDARES OPERATIVOS PARA IA (Gemini, Codex, Copilot, ChatGPT)

LeadBoostAI — DMC v1.0

Formato Enterprise (C)

---



## 7.1. Propósito del Capítulo

Definir:

- **roles formales** de cada IA
- **responsabilidades exactas**
- **límites estrictos** (qué NO pueden hacer)
- **cómo deben leer este DMC**
- **cómo deben colaborar entre sí**
- **el pipeline IA → RFC → Implementación → Auditoría**
- **la disciplina que mantiene la coherencia del proyecto**

Este capítulo convierte a tus IA en:

- **ingenieros senior,**
  - **arquitectos,**
  - **auditores,**
  - **operadores disciplinados.**
-



## 7.2. Los Cuatro Roles del Sistema IA

La arquitectura de trabajo IA se compone de **cuatro unidades**:

1. **ChatGPT (Alfred) – CTO/Director Estratégico**
2. **Gemini 3 Pro – Arquitecto Técnico**
3. **Copilot – Implementador y refactor automático**
4. **Codex – Indexador profundo y generador de documentación técnica**

Cada uno tiene funciones **no intercambiables**.

---

## 7.3. Rol Formal: CHATGPT (“ALFRED”) — CTO DEL SISTEMA

### Mandato

Dirigir la visión técnica y estratégica.

Tomar decisiones de alto nivel.

Definir fases.

Crear prompts maestros.

Orquestar a las demás IA.

### Responsabilidades principales

- Crear el pipeline de implementación.
- Escribir el DMC.
- Definir los invariantes.
- Crear la arquitectura conceptual de cada fase.
- Dar instrucciones precisas a Gemini y Copilot.
- Revisar consistencia de auditorías.
- Estandarizar diseño de prompts.

- Asegurar claridad total del proyecto.

## **Prohibido**

- Generar código operativo directo (eso es Copilot).
- Leer repositorios completos (eso es Codex).
- Hacer auditoría profunda de código (eso es Gemini).

## **Alfred garantiza:**

**claridad, dirección, orden y visión enterprise.**

---

## **7.4. Rol Formal: GEMINI 3 PRO — Arquitecto Técnico con Acceso al Repo**

### **Mandato**

Leer el repositorio REAL y producir:

- blueprint (RFC) completo de cada fase
- auditorías profundas
- especificaciones técnicas de implementación
- validación de invariantes arquitectónicos
- detección de riesgos
- sugerencias de diseño alineadas al DMC

### **Responsabilidades**

- Recibir prompts maestros de Alfred.
- Leer carpetas, archivos y repositorios.

- Diseñar arquitectura concreta.
- Identificar fallos catastróficos.
- Detectar violaciones a los invariantes.
- Evaluar si Copilot implementó correctamente.
- Generar reportes técnicos.

## **Prohibido**

- Modificar código directamente.
- Tomar decisiones estratégicas (eso es Alfred).
- Escribir implementación (eso es Copilot).
- Ignorar el DMC.

## **Gemini debe operar siempre bajo:**

- Capítulo 5 (contratos inter-servicio)
  - Capítulo 4 (invariantes)
  - Capítulo 3 (orquestación/SAGA)
  - Capítulo 6 (estructura del repo)
- 

## **7.5. Rol Formal: COPILOT — Implementador y Refactor**

### **Mandato**

Escribir código EXACTAMENTE según:

- el DMC

- el RFC generado por Gemini
- el prompt maestro de Alfred

## **Responsabilidades**

- Implementar funciones, handlers, servicios, modelos.
- Integrar microservicios.
- Refactorizar para cumplir con el blueprint.
- Aplicar auditorías de Gemini y Alfred.
- Asegurar consistencia con contratos inter-servicio.

## **Prohibido**

- Diseñar arquitectura por sí mismo.
- Cambiar decisiones del RFC.
- Inventar endpoints o modelos no existentes.
- Mover lógica a dominios equivocados.
- Saltarse invariantes.

**Copilot ejecuta, no decide.**

---

## **7.6. Rol Formal: CODEX — Indexador Profundo y Documentador**

### **Mandato**

Ser “los ojos internos del repositorio”.

### **Responsabilidades**

- Leer todo el repo sin límites de tamaño.
- Generar documentación técnica:
  - index de archivos
  - resumen por carpeta
  - mapa de dependencias
  - explicación de estructuras internas
- Producir un **Documento de Contexto del Código (DCC)** para Alfred y Gemini.
- Ayudar a detectar archivos problemáticos.
- Facilitar refactors grandes.

## Prohibido

- Crear arquitectura (eso es Alfred + Gemini).
- Escribir código (eso es Copilot).
- Decidir dominios.

Codex observa, describe, documenta.

---

## 7.7. Pipeline IA Ideal (flujo de trabajo completo)

Este es el pipeline que debe seguirse SIEMPRE:

1. Alfred (ChatGPT) define fase → crea prompt maestro
2. Gemini recibe el prompt maestro + repo → genera RFC / blueprint
3. Alfred revisa y aprueba el RFC → puede ajustarlo
4. Copilot implementa el RFC EXACTAMENTE
5. Gemini audita el código implementado
6. Alfred realiza auditoría conceptual (nivel CTO)
7. Copilot corrige según auditorías

8. Pruebas de ejecución real
9. Copilot genera reporte técnico de cierre
10. Gemini y Alfred reciben el reporte para mantener contexto

Esto crea un flujo **determinista**, profesional y escalable.

---

## 7.8. Reglas de Comunicación entre IAs

### 1. Solo Alfred se comunica con todos.

Tú eres Bruce Wayne.  
Alfred es tu mano derecha.

### 2. Gemini NUNCA debe hablar “por su cuenta”.

Todo lo que produce depende de:

- prompt maestro de Alfred
- capítulo correspondiente del DMC

### 3. Copilot espera instrucciones.

Nunca improvisa arquitectura.

### 4. Codex documenta, no decide.

### 5. Ninguna IA puede ignorar el DMC.

El DMC es la **constitución**.

---

## 7.9. Anti-patrones prohibidos en la colaboración IA

 Que Copilot diseñe arquitectura

Provoca caos estructural.

### Que Gemini implemente código

Rompe consistencia.

### Que Alfred dé instrucciones ambiguas

Crea fallos catastróficos.

### Que Gemini ignore invariantes

Rompe SAGA, produce duplicados o pérdida de mensajes.

### Que Copilot agregue funciones no especificadas

Acoplamiento no deseado.

### Que Codex produzca documentación sin contexto conceptual

Genera ruido técnico.

---

## 7.10. Cómo debe leer este DMC cada IA

### Gemini

- Lo usa como **guía arquitectónica absoluta**
- Valida si el repo respeta los dominios e invariantes
- Construye RFCs basados en él

### Copilot

- Lo usa para **ubicarse**
- Saber dónde generar código
- Entender límites y responsabilidades

### Codex

- Lo usa para **documentación coherente**
- Para generar un índice alineado al DMC

### Alfred

- Es el guardián del documento
  - Asegura que todos sigan las reglas
- 



## 7.11. Conclusión del Capítulo 7

Este capítulo establece:

- disciplina
- orden
- roles claros
- un pipeline profesional
- límites entre IA
- uso correcto del DMC

Es el capítulo que convertirá tus IA en un **equipo verdaderamente enterprise**, operando como:

- CTO
- Arquitecto
- Ingeniero
- Auditor
- Documentador

Todos alineados a una sola visión: **LeadBoostAI**.

# CAPITULO 8

# CAPÍTULO 8 — BLUEPRINT DE COMUNICACIÓN INTERNA DEL SISTEMA

LeadBoostAI – DMC v1.0

Formato Enterprise (C)

---



## 8.1. Propósito del capítulo

Este capítulo establece:

- ✓ El protocolo oficial de comunicación
- ✓ Las reglas formales para commands y events
- ✓ Los formatos estructurales de mensajes
- ✓ La taxonomía oficial del sistema
- ✓ Correlación mediante IDs
- ✓ Reglas absolutas para evitar duplicados y pérdida
- ✓ Cómo circula información entre dominios

Aquí definimos el **contrato interno más importante** de toda la plataforma.

---



## 8.2. Modelo de Comunicación General

LeadBoostAI opera bajo un modelo **event-driven**, con componentes loosely coupled, siguiendo principios de:

- SAGA

- mensajería resiliente
- commands + events
- correlación determinista
- idempotencia

Texto-diagrama:

Command → procesamiento → Event → consumidores → siguiente Command →  
...

---

## ◆ 8.3. Tipos Formales de Mensajes (Taxonomía Oficial)

Hay cuatro tipos principales:

1. Commands
2. Events
3. Queries
4. Approvals

Cada uno tiene un propósito único.

---

### 8.3.1. Command (orden explícita)

Formato:

```
{  
  "type": "command",  
  "name": "execute_action",  
  "operation_id": "...",  
  "saga_id": "...",  
  "payload": { ... },
```

```
"timestamp": "..."  
}
```

Propósito:

- mover el sistema
- desencadenar side-effects
- iniciar pasos de SAGA

Reglas:

- no deben perderse
- no pueden duplicarse
- deben validarse antes de ejecutarse

Prohibido:

- encadenar commands dentro de commands sin eventos intermedios
- 

### 8.3.2. Event (hecho ocurrido)

Formato:

```
{  
  "type": "event",  
  "name": "analysis.completed",  
  "operation_id": "...",  
  "saga_id": "...",  
  "payload": { ... },  
  "timestamp": "..."  
}
```

Propósito:

- notificar que algo ocurrió

- informar a múltiples microservicios
- avanzar SAGA

Reglas:

- deben persistirse antes de su consumo
  - no se ignoran
  - no llevan efectos secundarios
  - pueden tener múltiples consumidores
- 

### 8.3.3. Query (petición sin side effect)

Formato:

```
{  
  "type": "query",  
  "name": "memory.retrieve_context",  
  "payload": { ... }  
}
```

Propósito:

- recuperar datos
- buscar contexto
- obtener información sin mutar nada

Reglas:

- no generan commands
  - no avanzan flujos
  - no influyen en SAGA
-

### 8.3.4. Approval (autorización formal)

Formato:

```
{  
  "type": "approval",  
  "name": "strategy.approved",  
  "strategy_id": "...",  
  "constraints": { ... }  
}
```

Propósito:

- indicar que Enterprise aprueba una acción
- habilitar pasos en Actuator

Reglas:

- sin approval no hay ejecución
  - es un mensaje determinante
- 

## 8.4. Modelo de Correlación **(operation\_id, saga\_id, context\_id)**

Estos tres ID son el **ADN del sistema**.

---

### 8.4.1. **operation\_id**

Identifica UNA ejecución específica de un flujo.

Reglas:

- único
- obligatorio

- base de idempotencia
- 

#### 8.4.2. `saga_id`

Identifica el flujo completo multi-step.

Reglas:

- un saga\_id gobierna todo el ciclo
  - los microservicios usan saga\_id para saber qué esperar
- 

#### 8.4.3. `context_id`

Identifica el contexto empresarial (cliente, campaña, lead...).

Reglas:

- usado por Memory, Analyst, Optimizer
  - nunca debe faltar en análisis
- 

## 8.5. Secuencia estándar de comunicación entre microservicios

A continuación se detalla el blueprint formal de comunicación.

---

### ◆ 8.5.1. De Analyst → Memory

```
query: memory.retrieve_context  
event: insights.generated
```

Motivo: enriquecer análisis.

---

- ◆ **8.5.2. De Analyst → Optimizer**

event: insights.ready

Motivo: ciclo de decisión.

---

- ◆ **8.5.3. De Optimizer → Enterprise**

proposal: strategy.proposed

Motivo: validar reglas del negocio.

---

- ◆ **8.5.4. De Enterprise → Optimizer**

approval: strategy.approved

reject: strategy.rejected

adjust: strategy.adjusted

---

- ◆ **8.5.5. De Enterprise → Actuator**

approval: execute.action

Sin esto, Actuator no ejecuta nada.

---

- ◆ **8.5.6. De Actuator → Analyst**

event: execution.completed

event: execution.failed

Motivo: retroalimentación y ciclo cognitivo.

---

## ⭐ 8.6. Garantías de Entrega (Delivery Guarantees)

LeadBoostAI usa un modelo conceptual equivalente a:

### ✓ At-least-once delivery

Mensajes no se pierden.

### ✓ Idempotencia obligatoria

Duplicados no generan acciones duplicadas.

### ✓ No avanzar SAGA sin evento

Previene corrupción del flujo.

### ✓ Persistencia antes de acknowledgment

Evita pérdida silenciosa.

---



## 8.7. Anti-patrones prohibidos

### ✗ Commands que disparan commands sin eventos intermedios

Produce loops o duplicidad.

### ✗ Events que modifican estado interno

Los eventos notifican, no mutan.

### ✗ Queries que influyen en SAGA

Queries no deben afectar flujos.

### ✗ Approvals generados fuera de Enterprise

Rompe gobernanza.

### ✗ Microservicios hablando entre sí sin contrato previo

Debe existir flujo formal en Cap. 5.

---

## 8.8. Modelo de Seguridad sobre Mensajería

- todos los mensajes llevan `origin_service`
  - no se aceptan mensajes anónimos
  - aprobaciones llevan firma digital interna
  - mensajes críticos llevan `integrity_hash`
- 

## 8.9. Diagrama Formal de Flujo de Mensajes

Texto-diagrama:

```
Analyst
  └ event.insights.ready
    └→ Optimizer
      └ proposal.strategy
        └→ Enterprise
          └→ approval
            └→ Actuator
          └ reject (fin)

Actuator
  └ event.execution.completed
    └→ Analyst + Memory
```

---

## 8.10. Tipos oficiales de mensajes del sistema (Catálogo)

<b>Tipo</b>	<b>Ejemplo</b>	<b>Origen</b>	<b>Destino</b>
Command	execute.campaign	Backend/Optimizer	Actuator
Event	analysis.complete	Analyst d	Optimizer, Memory
Query	memory.retrieve	Analyst/Optimizer	Memory
Approval	strategy.approved	Enterprise	Actuator

---



## 8.11. Cómo deben usar esto las IA

### Gemini 3 Pro

Debe validar antes de cada fase que:

- commands están bien emitidos
- events están bien correlacionados
- approvals existen donde deben existir
- no hay flujos ilegales

### Copilot

Debe generar código que respete:

- el contrato formal de mensajes
- los tipos definidos
- el blueprint SAGA

### Codex

Debe documentar:

- dónde se manejan cada tipo de mensaje
- dónde se publican

- dónde se consumen

## Alfred

Debe mantener integridad conceptual del sistema.

---



## 8.12. Conclusión del Capítulo 8

Este capítulo define el **protocolo oficial de comunicación** del sistema, equivalente al “lenguaje interno” de LeadBoostAI.

Con esto:

- los microservicios son interoperables
- el sistema es consistente
- la ejecución es auditada
- la arquitectura es determinista
- las IA trabajan con un blueprint claro

# CAPITULO 9

# CAPÍTULO 9 — MODELO DE SEGURIDAD, IAM Y GOBERNANZA

LeadBoostAI — DMC v1.0

Formato Enterprise (C)

---



## 9.1. Propósito del capítulo

Definir el marco:

- de seguridad
- de acceso
- de gobernanza interna
- de firma de mensajes
- de auditoría
- de ejecución de acciones con impacto real

Este capítulo asegura que LeadBoostAI sea:

- ✓ Seguro
- ✓ Gobernable
- ✓ Auditado
- ✓ Predecible
- ✓ Trazable
- ✓ No manipulable
- ✓ No explotable

✓ No peligroso

---



## 9.2. Principios Rectores de Seguridad

Estos son los principios a los que TODO el sistema debe adherirse:

### 9.2.1. Principio de Privilegios Mínimos

Ningún componente debe tener más permisos de los necesarios para su función.

---

### 9.2.2. Principio de Separación de Responsabilidades

Los dominios no deben mezclarse ni asumir roles de otros microservicios.

---

### 9.2.3. Principio de Gobernanza Empresarial

Las decisiones automáticas siempre deben respetar:

- límites,
  - reglas,
  - políticas,
  - modelos económicos definidos por Enterprise.
- 

### 9.2.4. Principio de Auditabilidad Total

Toda acción debe dejar un rastro verificable.

---

### 9.2.5. Principio de Seguridad Determinista

El sistema NO debe tomar decisiones fuera del marco definido en el DMC.

---

## 9.3. Modelo de IAM (Identity and Access Management)

IAM gobierna:

- quién inicia flujos,
  - quién ejecuta acciones,
  - qué puede ver cada usuario,
  - qué puede escribir cada microservicio,
  - qué está estrictamente prohibido.
- 

### 9.3.1. Tipos de entidades en IAM

#### A) Usuarios humanos (UI)

Roles:

- Admin
- Manager
- Viewer

#### B) Microservicios (no humanos)

Cada uno es un “actor determinista” con permisos limitados.

#### C) Sistemas externos conectados

Plataformas Ads, CRM, APIs externas.

---

## 9.4. Matriz de Permisos para Microservicios

## Ley absoluta:

Ningún microservicio tiene permisos globales.

Dominio	Leer	Escribir	Modificar	Ejecutar Acción Externa
Analyst	✓	✗	✗	✗
Memory	✓	✓	✗	✗
Optimizer	✓	✗	✗	✗
Enterprise	✓	✓	✓	✗
Actuator	✓	✗	✗	✓
Backend	✓	✗	✗	✗
Frontend	✓	✗	✗	✗

## Interpretación:

- Solo **Actuator** ejecuta acciones externas.
- Solo **Enterprise** modifica reglas y políticas del sistema.
- Solo **Memory** escribe memoria contextual.



## 9.5. Límites Duros (NO negociables)

9.5.1. Sin autorización de Enterprise, NO hay ejecución.

9.5.2. Si un comando no tiene firma válida, se rechaza automáticamente.

9.5.3. Si un event no tiene correlación (`operation_id, saga_id`), se archiva y no se procesa.

9.5.4. El frontend JAMÁS puede ejecutar acciones reales directamente.

9.5.5. Actuator no puede generar estrategias.

9.5.6. Optimizer no puede saltarse políticas.

**9.5.7. Analyst no puede modificar estado del sistema.**

**9.5.8. Memory no puede almacenar información sin validación.**

---

## **9.6. Firma Digital Interna para Mensajes Críticos**

Cada mensaje de ejecución (Actuator) debe incluir:

```
"signature": {  
    "approved_by": "enterprise",  
    "timestamp": "...",  
    "policy_hash": "...",  
    "integrity_hash": "..."  
}
```

Esto asegura:

- autenticidad
- trazabilidad
- cumplimiento de reglas
- resiliencia ante manipulación

---

## **9.7. Auditoría Interna (Log de Gobernanza)**

Todo lo siguiente DEBE ser auditado:

- estrategias propuestas
- estrategias aprobadas
- acciones ejecutadas

- anomalías detectadas
- restricciones violadas
- rechazos de Enterprise
- fallos en Actuator
- compensaciones de SAGA

La auditoría es accesible solo a:

- Enterprise
  - Backend (para reporting)
  - Admin humano
- 

## 9.8. Gobernanza de Decisiones Automatizadas

El sistema debe evitar:

- acciones peligrosas,
- gasto excesivo,
- loops de auto-ejecución,
- decisiones sin base analítica.

Por eso:

### **Regla 1: Toda decisión pasa por 3 validaciones**

1. análisis (Analyst)
2. decisión (Optimizer)
3. política (Enterprise)

**Regla 2: Optimizer no ejecuta.**

**Regla 3: Actuator no decide.**

**Regla 4: Enterprise puede detener cualquier flujo.**

---

## 💣 9.9. Anti-patrones estrictamente prohibidos

- ✖ Ejecución directa desde UI
- ✖ Atajos entre microservicios sin backend
- ✖ Aprobaciones generadas fuera de Enterprise
- ✖ Subida de decisiones sin análisis
- ✖ Saltos entre dominios (ej. Memory → Actuator)
- ✖ Hardcode de credenciales en cualquier servicio
- ✖ Actuator ejecutando acciones sin firma
- ✖ Modificación de memoria sin validación

Cada uno de estos es considerado un **fallo crítico de seguridad**.

---

## 🛡 9.10. Modelo de Defensa por Capas (Defense-in-Depth)

La seguridad opera en múltiples niveles:

1. **IAM** → acceso controlado
2. **Firma digital** → autenticidad de mensajes
3. **Enterprise** → gobernanza del negocio

4. **Actuator** → ejecución segura
  5. **Invariantes** → reglas inquebrantables
  6. **Auditoría** → trazabilidad completa
  7. **Mensajería** → integridad y consistencia
  8. **Repositorio** → estructura limpia
- 



## 9.11. Cómo deben usar este capítulo las IA

### Gemini

Debe validar que cada fase nueva respete:

- roles
- permisos
- firmas
- gobernanza
- límites de ejecución

### Copilot

Nunca debe generar código que:

- ejecute acciones sin política
- rompa contratos de gobernanza
- introduzca lógica peligrosa

### Codex

Debe documentar:

- dónde vive IAM
- dónde se aplican aprobaciones
- dónde se loggean auditorías

## **Alfred**

Supervisa que TODAS las reglas se cumplan.

---



## **9.12. Conclusión del Capítulo 9**

Con este capítulo:

- el sistema ya no puede actuar fuera de control
- los microservicios están protegidos por límites duros
- Actuator solo ejecuta si Enterprise lo aprueba
- los flujos son auditables
- las decisiones son trazables
- LeadBoostAI se vuelve seguro y confiable

Este es el nivel real de seguridad enterprise.

# CAPITULO 10

## CAPÍTULO 10 — PIPELINE COGNITIVO COMPLETO

LeadBoostAI – DMC v1.0

Percepción → Interpretación → Decisión → Acción → Retroalimentación

### 10.1. Propósito del capítulo

Definir el ciclo cognitivo completo que gobierna todas las operaciones inteligentes del sistema, es decir:

cómo entra la información,

cómo se analiza,

cómo se contextualiza,

cómo se decide,

cómo se ejecuta,

cómo se evalúa,

cómo se aprende,

cómo se reinicia el ciclo.

Este pipeline es el equivalente al “cerebro operativo” de LeadBoostAI.

### 10.2. Arquitectura Cognitiva — Vista General

Texto-diagrama:

DATA IN → ANALYST → MEMORY → OPTIMIZER → ENTERPRISE → ACTUATOR → FEEDBACK → ANALYST (ciclo)

Este ciclo representa la cadena de pensamiento del sistema.

Cada dominio tiene una función cognitiva específica:

Analyst = percepción

Memory = contexto y significado

Optimizer = planeación y decisión

Enterprise = validación y gobernanza

Actuator = ejecución

Analyst + Memory = retroalimentación

● 10.3. Etapa 1 — PERCEPCIÓN (Analyst)

Objetivo

Responder: “¿Qué está pasando ahora?”

Entradas típicas:

comportamiento del lead

métricas de campaña

resultados de anuncios

señales externas (Scout)

datos del negocio

interacciones humanas

Responsabilidades:

normalizar datos

detectar anomalías

identificar patrones

extraer señales relevantes

Salida:

insights.generated

● 10.4. Etapa 2 — INTERPRETACIÓN / CONTEXTUALIZACIÓN (Memory)

Objetivo

Responder: “¿Qué significa esto a la luz de lo que ya sabemos?”

Memory provee:

contexto histórico

similitudes

casos previos

memoria de negocios

embeddings semánticos

estados previos

Salida:

context.retrieved

Esta etapa evita decisiones basadas en datos aislados.

#### ◆ 10.5. Etapa 3 — PLANEACIÓN / DECISIÓN (Optimizer)

Objetivo

Responder: “¿Qué debemos hacer ahora?”

Entradas:

insights (Analyst)

contexto (Memory)

restricciones (Enterprise)

Proceso:

evaluar escenarios

simular consecuencias

generar múltiples alternativas

asignar score

seleccionar estrategia óptima

Salida:

strategy.proposed

actionplan.generated

Esta etapa es equivalente al razonamiento estratégico humano.

#### ▀▀ 10.6. Etapa 4 — VALIDACIÓN / GOBERNANZA (Enterprise)

Objetivo

Responder: “¿Esta decisión respeta las reglas del negocio?”

Enterprise valida:

límites de presupuesto

ética empresarial

cronogramas

políticas internas

umbrales de riesgo

integridad del sistema

permisos

coherencia con objetivos

Salida:

strategy.approved

strategy.rejected

strategy.adjusted

risk.flagged

Sin aprobación, no existe ejecución.

#### 10.7. Etapa 5 — EJECUCIÓN OPERATIVA (Actuator / Actuator Plus)

Objetivo

Responder: “¿Cómo transformamos esta estrategia en acciones reales?”

Tareas:

conectarse con Google Ads / Meta / CRM

enviar emails

modificar presupuestos

activar workflows

crear campañas

actualizar estados

Entrada:

strategy.approved

Salida:  
execution.started  
execution.completed  
execution.failed

Esta es la etapa donde ocurre el impacto real en negocio.

#### 10.8. Etapa 6 — FEEDBACK / RETROALIMENTACIÓN

Entradas:

eventos de ejecución  
métricas reales  
resultados observables

Estas señales fluyen de vuelta a Analyst para iniciar un nuevo ciclo:

execution.completed → Analyst → insights.generated → ...

Y parte del feedback fluye a:

Memory, para aprendizaje.  
Optimizer, para mejorar estrategias futuras.  
Enterprise, para ajustar reglas si es necesario.

#### 10.9. Modelo Cognitivo Formal (Blueprint)

Este modelo es un pipeline determinista, similar a un loop de control empresarial.

Diagrama textual:

STEP 1: perceive()  
STEP 2: interpret()  
STEP 3: decide()  
STEP 4: govern()  
STEP 5: execute()  
STEP 6: evaluate()  
STEP 7: learn()  
STEP 8: loop back

Cada paso:

es un dominio distinto,  
tiene responsabilidades únicas,  
está regulado por invariantes del Capítulo 4.

Esto hace imposible que un microservicio “robe funciones” de otro.

#### 10.10. Errores Cognitivos Prohibidos

El sistema NO debe:

-  decidir sin interpretar
-  ejecutar sin decidir
-  decidir sin validar
-  actuar sin aprobación
-  aprender de datos corruptos
-  ignorar feedback
-  ejecutar loops sin control
-  tomar acciones no solicitadas
-  mezclar roles entre dominios

Cada uno de estos es considerado corrupción cognitiva.

#### 10.11. Garantías Cognitivas del Sistema

Estas garantías protegen la integridad del pensamiento del sistema:

- ✓ Coherencia (no contradicción)
- ✓ Determinismo (misma entrada → mismo resultado)
- ✓ Auditabilidad
- ✓ Idempotencia cognitiva (evitar “pensar dos veces” lo mismo sin cambio)
- ✓ Gobernanza obligatoria
- ✓ No pérdida de conocimiento
- ✓ Aprendizaje continuo pero seguro

#### 10.12. Cómo deben usar este capítulo las IA

Gemini

- usa el pipeline cognitivo como marco para diseñar fases inteligentes
- asegura que todo ajuste respete el ciclo
- detecta errores de razonamiento en la arquitectura

Copilot

- implementa código en el dominio correcto del pipeline
- no mezcla etapas cognitivas

Codex

- documenta el flujo completo del sistema
- identifica dónde vive cada función real del pipeline

Alfred

- dirige decisiones estratégicas basadas en este modelo
- corrige desviaciones del pensamiento del sistema

#### 🏁 10.13. Conclusión del Capítulo 10

Este capítulo define:

cómo piensa LeadBoostAI

cómo decide

cómo actúa

cómo aprende

cómo se retroalimenta

cómo mantiene su coherencia

Es la base para:

IA avanzada,

automatización determinista,

ejecución segura,

optimización continua.

Con este pipeline, LeadBoostAI adquiere un modelo mental profesional.

# CAPITULO 11

 CAPÍTULO 11 — ESCALABILIDAD, RESILIENCIA Y DESPLIEGUE ENTERPRISE  
LeadBoostAI – DMC v1.0  
Formato Enterprise (C)  
 11.1. Propósito del capítulo

Definir las reglas, arquitecturas y protocolos que permitirán a LeadBoostAI:

escalar horizontalmente,

resistir fallos de componentes,

permanecer operativo 24/7,

desplegar versiones nuevas sin interrupción,

operar en múltiples entornos,

soportar picos de carga,

evitar corrupción en sistemas distribuidos,

mantener integridad del negocio ante errores,

ser auditado y monitoreado.

Esto convierte tu sistema en una plataforma digna de empresas grandes.

 11.2. Principios de Arquitectura Resiliente

LeadBoostAI opera bajo 6 principios:

11.2.1. Escalabilidad Horizontal

Cualquier servicio debe poder multiplicarse por N sin cambiar su código.

11.2.2. Stateless en microservicios y Stateful solo en storage

Los microservicios no mantienen estado local crítico.

11.2.3. Tolerancia a Fallos

El sistema debe sobrevivir:

caída de microservicios

caída de un nodo

fallos de red

fallos de API externa

fallos de ejecución en Actuator

#### 11.2.4. Degradación Elegante (Graceful Degradation)

Si algo falla, el sistema baja de capacidades, NO colapsa.

#### 11.2.5. Observabilidad Completa

Todo debe ser medible, visible, trazable.

#### 11.2.6. Zero-Trust Interno

Cada microservicio valida inputs aunque venga de otro microservicio.

### 🔥 11.3. Escalabilidad del Sistema

LeadBoostAI escala bajo el modelo horizontal-first, donde duplicar instancias de microservicios es el método principal para mejorar capacidad.

#### 11.3.1. Microservicios escalables

Los siguientes servicios pueden escalar ilimitadamente:

microservice\_analyst

microservice\_memory (si usa backend vectorial o cluster)

microservice\_optimizer

microservice\_scout

microservice\_actuator

microservice\_actuator\_plus

Reglas para escalar:

no usar estado local

no depender del orden absoluto de mensajes

usar operation\_id para idempotencia

usar colas/eventos para distribución de carga

 11.4. Resiliencia por Dominio

Cada microservicio tiene exigencias específicas:

#### 11.4.1. Analyst — resiliencia

reintenta análisis fallidos

no pierde mensajes

se recupera del fallo leyendo eventos previos

deja logs detallados para auditoría

#### 11.4.2. Memory — resiliencia

almacenamiento durable

respaldo periódico

fallback a versiones previas

reconstrucción de memoria semántica

#### 11.4.3. Optimizer — resiliencia

evita loops infinitos de planeación

garantiza coherencia en decisiones

reintenta simulaciones fallidas

#### 11.4.4. Enterprise — resiliencia

validación estricta siempre

no permite decisiones sin reglas

fallback cuando faltan datos críticos

nunca pierde políticas internas

#### 11.4.5. Actuator — resiliencia

reintentos con estrategia de backoff

evitar acciones duplicadas

proteger APIs externas

compensación cuando fallan ejecuciones

## 11.5. Gestión de Fallos (Fault Management)

LeadBoostAI debe manejar 4 tipos de errores:

### 11.5.1. Fallos Transitorios

Ej: timeout en API de Google Ads.

- reintento automático
- backoff exponencial
- log de intento

### 11.5.2. Fallos Persistentes

Ej: fallo en integraciones externas.

- marcar estrategia como fallida
- detener SAGA
- enviar a Enterprise para revisión

### 11.5.3. Fallos Catastróficos

Ej: pérdida de integridad en memoria.

- fallo crítico
- rollback
- alertas al administrador
- auditoría obligatoria

### 11.5.4. Fallos Operativos

Ej: error humano UI.

- UI muestra error
- backend registra
- no impacta flujo core

## 11.6. Observabilidad Enterprise

Observabilidad es indispensable para saber si el cerebro del sistema está funcionando.

LeadBoostAI debe exponer:

### 11.6.1. Logs estructurados (JSON)

Incluyen:

operation\_id

saga\_id

context\_id

timestamp

origin

payload

estado

#### 11.6.2. Métricas

latencia

throughput

fallos por servicio

consumo de cola

tamaño de mensajes

#### 11.6.3. Trazas distribuidas

Cada acción del pipeline debe ser trazable.

Ejemplo:

analysis → context → plan → approval → execution → feedback

#### 11.6.4. Health checks

Cada microservicio expone:

health/ready

health/live

health/full

Para saber:

si está vivo

si está listo

si puede aceptar carga

#### 11.7. Estrategia de Despliegue Enterprise

El despliegue de LeadBoostAI debe cumplir:

##### 11.7.1. Multi-Entorno Obligatorio

Mínimo 3 ambientes:

Development

Staging / Pre-Producción

Production

NUNCA desarrollar directamente en producción.

##### 11.7.2. Despliegue Inmutable

Cada release genera:

contenedor inmutable

etiqueta única (tag)

artefacto verificable

##### 11.7.3. Estrategia de Zero-Downtime

Métodos aceptados:

Blue/Green Deployment

Rolling Updates

Canary Releases

Actuator siempre debe ser actualizado de forma canary por ser crítico.

##### 11.7.4. CI/CD Obligatorio

Pipeline debe incluir:

pruebas automáticas

auditoría de seguridad

revisión de invariantes

despliegue controlado

#### 11.8. Hardening del Sistema

Hardening significa hacer el sistema más robusto, resistente y seguro.

Reglas:

sin credenciales en código

variables en vault seguro

deshabilitar endpoints no usados

logging sensible anonimizado

rate limiting en APIs externas

auditoría estricta en Actuator

#### 11.9. Anti-patrones prohibidos (nivel infraestructura)

Los siguientes comportamientos destruyen resiliencia:

 microservicios con estado local persistente

 no usar IDs para correlación

 despliegue manual sin CI/CD

 Actuator ejecutando sin autorización

 UI controlando lógica crítica

 backend actuando sin SAGA

 logs no estructurados

 ignorar fallos de mensajería

 no usar health checks

#### 11.10. Cómo deben usar este capítulo las IA

Gemini

→ valida resiliencia al diseñar fases

→ audita riesgos de escalabilidad

→ detecta fallos arquitectónicos graves

Copilot

- implementa código que soporte escalabilidad
- evita estados locales
- genera handlers robustos

Codex

- documenta despliegue, rutas, health checks

Alfred

- garantiza que nuevas fases respeten estos principios

#### 🏁 11.11. Conclusión del Capítulo 11

Ahora LeadBoostAI tiene:

- ✓ Esquema de escalabilidad enterprise
- ✓ Modelo de resiliencia industrial
- ✓ Estrategias de tolerancia a fallos
- ✓ Observabilidad completa
- ✓ Lineamientos de despliegue profesional
- ✓ Hardening real
- ✓ Arquitectura 100% lista para crecer

Este capítulo convierte tu software en un sistema que puede operar a nivel Fortune 100.

# CAPITULO 12



# CAPÍTULO 12 — APÉNDICE TÉCNICO Y GLOSARIO DEL SISTEMA

LeadBoostAI – DMC v1.0

Formato Enterprise (C)

---



## 12.1. Propósito del capítulo

Este capítulo contiene:

- el glosario oficial del sistema
- contratos extendidos
- definiciones formales
- plantillas RFC
- plantillas de auditoría
- estructura estándar de prompts maestros
- tablas de dominios
- reglas finales inquebrantables
- anexos técnicos para IA

Este capítulo asegura que el sistema pueda:

- escalar correctamente,
- mantenerse ordenado,
- integrarse con IA avanzadas,
- ser auditado con precisión,

- ser extendido sin romper invariantes,
  - mantener coherencia por años.
- 



## 12.2. Glosario Técnico Oficial (Términos Core)

### **SAGA**

Patrón de coordinación distribuida que divide procesos largos en pasos compensables.

### **Command**

Mensaje que ordena ejecutar una operación.

### **Event**

Mensaje que notifica que algo ocurrió.

### **Approval**

Autorización emitida por Enterprise para permitir ejecución.

### **Query**

Solicitud sin side-effects.

### **operation\_id**

Identificador único de una operación.

### **saga\_id**

Identificador único de toda una SAGA.

### **context\_id**

Identificador de contexto empresarial.

### **Actuator**

Dominio encargado de ejecutar acciones externas.

## **Enterprise**

Dominio que gobierna reglas, políticas y autorizaciones.

## **Optimizer**

Dominio que genera decisiones estratégicas.

## **Analyst**

Dominio que percibe y analiza información.

## **Memory**

Dominio que almacena y provee contexto.

## **Frontend/UI**

Capa visual para interacción humana.

## **Backend/BFF**

Capa que coordina flujos entre UI y microservicios.

---

## **12.3. Tabla de Dominios + Responsabilidades**

<b>Dominio</b>	<b>Responsabilidad</b>	<b>Prohibido</b>
Analyst	percepción, insights	decidir, ejecutar
Memory	contexto, embeddings	ejecutar, gobernar
Optimizer	planeación, recomendación	ejecutar, aprobar
Enterprise	gobernanza, políticas	actuar, analizar
Actuator	ejecución externa	decidir, validar
Backend	orquestación inicial	lógica de negocio
UI	interacción humana	lógica critica

---

## 12.4. Contratos Extendidos Entre Dominios

**Analyst → Optimizer**

`insights.ready`

**Memory → Optimizer**

`context.ready`

**Optimizer → Enterprise**

`strategy.proposed`

**Enterprise → Actuator**

`strategy.approved`

**Actuator → Analyst**

`execution.completed`

**Analyst → Memory**

`feedback.learned`

---

## 12.5. Formato Oficial de Mensaje (Universal Message Schema – UMS)

```
{  
  "type": "command | event | query | approval",  
  "name": "string",  
  "operation_id": "uuid",  
  "saga_id": "uuid",  
  "context_id": "uuid",  
  "origin": "service_name",  
  "timestamp": "iso8601",  
  "payload": {}  
}
```

```
"signature": {  
    "approved_by": "enterprise|system",  
    "policy_hash": "sha256",  
    "integrity_hash": "sha256"  
}  
}  
}
```

## Reglas:

- `signature` obligatorio solo para acciones externas.
  - `operation_id` es obligatorio en todos menos queries.
  - `payload` nunca lleva datos sensibles.
  - `origin` debe siempre declararse.
- 



## 12.6. Invariantes Finales (Versión Formal)

Los invariantes del Capítulo 4 se presentan aquí como reglas matemáticas.

---

### 12.6.1. Invariante 1 — Ninguna ejecución sin aprobación

$\forall a \in \text{Actions}: \text{execute}(a) \Rightarrow \text{approval.exists}(a)$

---

### 12.6.2. Invariante 2 — SAGA nunca avanza sin evento

$\forall s \in \text{Steps}: \text{progress}(s) \Rightarrow \text{previous\_event.exists}(s)$

---

### 12.6.3. Invariante 3 — Idempotencia estricta

$\forall op: \text{process}(op) = \text{process}(op) \quad (\text{if } op.\text{operation\_id} \text{ repeated})$

---

#### **12.6.4. Invariante 4 — Dominio no cruzado**

```
forall logic L: L in D_i => L not in D_j (i != j)
```

---

#### **12.6.5. Invariante 5 — Memoria nunca ejecuta**

```
forall actions: actions.origin != Memory
```

---

#### **12.6.6. Invariante 6 — Actuator nunca decide**

```
forall decisions: decisions.origin != Actuator
```

---

#### **12.6.7. Invariante 7 — Optimizer nunca ejecuta**

```
execute(action) => origin != Optimizer
```

---



## **12.7. Plantilla Oficial de RFC para Gemini**

Gemini debe producir RFCs siguiendo exactamente este formato:

---

#### **12.7.1. Estructura del Documento RFC**

RFC-[phase\_id]: [Nombre de la fase]

1. Objetivo
2. Cambios arquitectónicos
3. Cambios de dominio
4. Cambios de modelos
5. Cambios de contratos
6. Cambios de mensajería
7. Cambios en SAGA
8. Riesgos técnicos
9. Pasos de implementación
10. Criterios de aceptación

### **Reglas para Gemini:**

- no puede inventar dominios nuevos
  - no puede cambiar contratos sin motivo
  - no puede romper invariantes
  - debe incluir riesgos sí o sí
- 



## **12.8. Plantilla de Prompt Maestro para Copilot (Implementación)**

ACTÚA COMO IMPLEMENTADOR SENIOR.

SIGUE ESTE RFC SIN DESVIARTE.

NO CAMBIES ARQUITECTURA.

NO INVENTES NADA.

RESPETA DOMINIOS Y CONTRATOS.

USA EL CÓDIGO ACTUAL COMO BASE.

IMPLEMENTA EXACTAMENTE LO SOLICITADO.

AL FINAL, GENERA RESUMEN DE CAMBIOS.

---



## **12.9. Plantilla de Auditoría Técnica (ChatGPT o Gemini)**

ACTÚA COMO AUDITOR FORENSE DE SISTEMAS DISTRIBUIDOS.

REVISA ÚNICAMENTE LOS ARCHIVOS PROPORCIONADOS.

BUSCA:

- pérdida de mensajes
- corrupción de saga
- doble procesamiento
- break de invariantes
- lógica fuera de dominio
- errores de mensajería
- problemas de seguridad

- idempotencia rota
- race conditions críticas

NO REPORTES:

- estilo
- sugerencias
- optimizaciones menores

FORMATO DE REPORTE:

1. Archivo
  2. Sección
  3. Naturaleza del fallo
  4. Riesgo
  5. Corrección precisa
- 

## 12.10. Reglas de Estilo Arquitectónico

**Obligatorio:**

- dominios separados
- estructura limpia
- contratos formales
- mensajes tipados

**Prohibido:**

- archivos gigantes
  - lógica mezclada
  - módulos multipropósito
  - funciones sin tests
-



## 12.11. Tablas de Referencia Rápida

### Mapa Rápido del Pipeline Cognitivo

Paso	Dominio
percepción	Analyst
interpretación	Memory
decisión	Optimizer
gobernanza	Enterprise
ejecución	Actuator
feedback	Analyst/Memory

---



## 12.12. Anexo: Estructura de un Blueprint de Gemini

BLUEPRINT-[phase\_id]

1. Descripción general
  2. Nuevos archivos
  3. Archivos modificados
  4. Contratos actualizados
  5. Flujos afectado
  6. Diagrama de comunicación
  7. Lógica a implementar
  8. Validaciones necesarias
  9. Errores a evitar
- 



## 12.13. Cierre del DMC v1.0

Con este capítulo, el Documento Maestro de Contexto queda **completo**.

El sistema ahora tiene:

- visión
- arquitectura
- invariantes
- contratos
- pipeline cognitivo
- seguridad
- resiliencia
- gobernanza
- estructuras de RFC
- plantillas de auditoría
- glosario
- anexos
- reglas de operación IA

**Ya tienes una constitución enterprise para LeadBoostAI.**