Guía Maestra del Proyecto

0) Objetivo del sistema

Construir una plataforma "base" (Core) que:

- Funciona por sí sola con un **panel de análisis y riesgos**.
- Recibe **módulos instalables** (Nóminas, Legal, Contratos, Producción, Clientes, etc.).
- Centraliza y normaliza riesgos procedentes de módulos en un **motor de riesgos común**.
- Garantiza **auditoría inmutable** con **trazabilidad blockchain**.
- Mantiene **seguridad extrema** (Zero-Trust, ABAC/RBAC, cifrado, logging).
- Opera con **multi-idioma completo**, sin hardcodear textos (base en inglés).
- Documenta TODO de forma automática y consistente:
    - Cada archivo nuevo → MD de estructura
    - Cada variable nueva → MD de variables
    - Estructura DB → MD de base de datos
    - Antes de crear archivos → leer documentación para nombres/ubicación

---

1) Principios innegociables

1. **Modularidad real**: Core no conoce lógica de negocio específica de un vertical; solo define contratos.
2. **Sin hardcodeo**:
    - Cero textos en UI/Backend fuera del sistema i18n.
    - Cero "strings mágicas" para roles, eventos o riesgos.
3. **Auditoría y trazabilidad por defecto**:
    - Toda acción relevante genera evento auditable.
    - Todo evento se "ancora" a blockchain.
4. **Seguridad por diseño**:
    - Principio de mínimo privilegio.
    - Todo acceso pasa por un "Policy Engine".
5. **Calidad bancaria**:
    - UI consistente, sobria, alta legibilidad, micro-interacciones, accesibilidad.
6. **Documentación como parte del build**:
    - Si no se documenta, no está terminado.

---

2) Arquitectura del sistema

2.1 Capas

Core Platform

- Identity & Access (RBAC/ABAC)
- Tenant management (multi-tenant)
- Event Bus (audit + domain events)
- Risk Engine (normalización, scoring, estado)
- Analytics & Dashboard
- i18n Engine (frontend+backend)
- Storage (DB, cache, object store)
- Blockchain Anchoring (hashing + anchoring + verify)
- Module Runtime (carga, permisos, migraciones, UI extension)

Modules

- Módulos instalables (Payroll, Legal, Contracts, etc.)
- Cada módulo implementa:
  - Conectores de datos
  - Detectores de riesgo
  - UI extension (pantallas, widgets)
  - Migraciones DB propias
  - Eventos y políticas propias

---

3) Estructura del repositorio

Estructura propuesta (monorepo):

```
/docs
  /architecture
  /database
  /modules
  /variables
  /standards
  /runbooks

/apps
  /web          # Frontend (panel bancario)
  /api          # Backend API (core)
  /worker       # Jobs/queues (detección riesgos, sincronización,
anchoring)

/packages
  /core         # librerías core (events, risk engine, i18n, security)
  /ui           # design system bancario (componentes, tokens)
  /i18n         # utilidades i18n compartidas
  /sdk          # SDK para módulos (contratos, helpers)
  /blockchain   # anchoring, verification, hashing

/modules
  /payroll
  /legal
  /contracts
  /production
```

```
  /crm

/scripts
  generate_docs
  lint
  validate_modules
  check_i18n
```

Regla: **Core jamás importa módulos**. Los módulos se registran dinámicamente.

---

4) Sistema de módulos instalables

4.1 Contrato mínimo de un módulo

Cada módulo debe exponer un manifiesto `module.json` (o TS) con:

- `id`: string único (kebab-case). Ej: `payroll`
- `version`: semver
- `name_i18n_key`: clave i18n (ej: `modules.payroll.name`)
- `permissions`: listado de permisos que añade
- `policies`: reglas ABAC/RBAC
- `db_migrations_path`: migraciones aisladas del módulo
- `risk_providers`: lista de detectores
- `ui_extensions`: rutas/páginas/widgets

4.2 Interfaces obligatorias (SDK)

- `registerModule(context)`
  Registra:
  - detectores de riesgo
  - pantallas/wigets
  - comandos / endpoints del módulo (si aplica)
- `getRiskFindings(tenantId, timeframe)`
  Retorna hallazgos con un formato **normalizado** (ver 5)
- `getModuleHealth()`
  Estado: OK/WARN/FAIL

4.3 Aislamiento

- Cada módulo:
  - Tiene su propio namespace de DB (`module_<id>_*`)
  - Tiene su propio prefijo de eventos: `module.<id>.`
  - Tiene su propio set i18n: `i18n/en/modules/<id>.json`

---

5) Motor de riesgos (Core)

5.1 Modelo único de riesgo

Todos los módulos deben emitir riesgos usando el mismo esquema:

RiskFinding

- `finding_id` (UUID)
- `tenant_id`
- `module_id`
- `category` (enum core: legal, payroll, security, ops, finance…)
- `severity` (enum: low, medium, high, critical)
- `likelihood` (0..1)
- `impact_eur` (number, opcional)
- `title_i18n_key`
- `description_i18n_key`
- `evidence_refs[]` (links internos a docs/records)
- `entities[]` (empleado, contrato, cliente…)
- `status` (new, acknowledged, mitigated, resolved, false_positive)
- `detected_at`, `updated_at`
- `recommended_actions[]` (i18n keys)
- `policy_tags[]` (para ABAC)

5.2 Estados de riesgo

- **Actuales**: riesgo vigente con evidencia.
- **Inminentes**: riesgo con condición de disparo próxima (fechas, thresholds, patrones).
- **Tendenciales**: riesgo que sube por señales.

El dashboard del core debe tener:

- Vista general (KPIs)
- Timeline
- Heatmap de severidad
- “Top exposures”
- Drill-down por entidad (empresa, empleado, contrato, proceso)

---

6) Multi-idioma estricto (base en inglés)

6.1 Reglas

- Idioma base: **en**

- Prohibido:
  - textos hardcodeados en UI
  - mensajes de error en string literal
- Todo texto tiene clave i18n.

6.2 Estructura de i18n

Frontend:

```
/apps/web/i18n
  /en/common.json
  /en/core.json
  /en/modules/payroll.json
  /es/common.json
  /es/core.json
  /es/modules/payroll.json
```

Backend:

- Mensajes de auditoría y errores también usan i18n keys (para logs y UI).

6.3 Convención de claves

- Core: `core.section.feature.label`
- Módulos: `modules.<id>.feature.label`
- Errores: `errors.<scope>.<code>`

---

7) Trazabilidad blockchain (auditoría inmutable)

7.1 Concepto

Cada evento importante genera un registro auditable.
Ese registro se resume en un hash y se **ancora en blockchain**.

7.2 Qué se ancla

- Evento normalizado (JSON canonical)
- Hash: `sha256(canonical_event_json)`
- Se agrupan hashes en lotes (batch) para eficiencia:
  - Merkle root del lote
  - Anclaje del root en blockchain

7.3 Niveles

- **Nivel 1**: hash por evento + lote diario
- **Nivel 2**: lote por tenant / criticidad
- **Nivel 3**: “evidence pack” firmable (pdf + hash + proof)

7.4 Verificación

Debe existir una funcionalidad:

- “Verify audit proof”:
  - recalcula hash
  - verifica inclusión en Merkle
  - verifica transacción en cadena
  - muestra evidencia

---

8) Seguridad extrema (bancaria)

8.1 Identidad y acceso

- Multi-tenant estricto
- RBAC + ABAC:
  - Roles (admin, auditor, manager, viewer, module-operator)
  - Atributos (departamento, país, entidad, riesgo, nivel)
- MFA obligatorio para roles privilegiados
- Session management con rotación y device binding (si aplica)

8.2 Cifrado

- En tránsito: TLS
- En reposo: cifrado DB + secrets manager
- Campos sensibles: cifrado a nivel de aplicación (envelope encryption)

8.3 Auditoría

- Todo:
  - login/logout
  - cambios de permisos
  - cambios de configuración
  - acciones que afecten riesgos
  - exportaciones (PDF, CSV, packs)

8.4 Defensa

- Rate limiting
- WAF / bot protection
- Hardening headers

- Validación estricta de inputs
- Zero trust entre servicios

---

9) UI/UX bancaria “máximo nivel”

9.1 Design system

- Tokens (espaciado, tipografía, elevación, bordes)
- Componentes consistentes:
  - Tables, filters, search, drawers, modals
  - Risk cards, severity chips, timeline
- Accesibilidad AA:
  - navegación teclado
  - contraste
  - estados focus
- Microinteracciones sobrias

9.2 Reglas de UX

- Todo “acción crítica” requiere confirmación + audit event
- Exportaciones deben mostrar:
  - qué incluye
  - periodo
  - firma/hash
- Los riesgos deben ser “drill-downable” en 2 clicks máximo

---

10) Documentación obligatoria (automatizada)

10.1 Regla general

Cada vez que se cree o modifique algo:

- Actualizar documentación correspondiente
- Si no hay doc → no se da por finalizado

10.2 Documentos obligatorios

1. **Estructura de archivos (por cada archivo nuevo)**

- Crear: `/docs/architecture/file-map.md` o, mejor:

- Crear un MD por archivo:
  - `/docs/architecture/files/<path_sanitized>.md`

Contenido mínimo:

- Ruta
- Propósito
- Dependencias
- Entradas/salidas
- Eventos que emite
- i18n keys que usa

2. **Variables / Constantes / Config**

- `/docs/variables/registry.md`
- Cada nueva variable/constante/feature flag:
  - nombre
  - tipo
  - scope
  - default
  - dónde se usa
  - impacto seguridad

3. **Base de datos**

- `/docs/database/schema.md`
- Por tabla:
  - campos, tipos, constraints
  - índices
  - relaciones
  - qué eventos la tocan
- Para módulos:
  - `/docs/database/modules/<id>.md`

10.3 “Antes de crear archivo”: lectura obligatoria

Claude Code debe **siempre**:

1. Leer `/docs/standards/naming.md`
2. Leer `/docs/standards/architecture.md`
3. Leer `/docs/variables/registry.md` (evitar duplicados)
4. Leer `/docs/database/schema.md` (evitar inconsistencias)
5. Leer `/docs/modules/module-sdk.md`

---

11) Convenciones de nombres

- Carpetas: kebab-case

- Archivos TS/JS: kebab-case
- Clases: PascalCase
- Funciones: camelCase
- Enums: PascalCase
- Eventos: `core.<domain>.<action>` o `module.<id>.<action>`
- Tablas core: `core_*`
- Tablas módulo: `module_<id>_*`

---

12) Checklist obligatorio para Claude Code (cada PR)

A. Arquitectura

- ¿Está el código en la capa correcta?
- ¿Core no depende de módulos?

B. i18n

- ¿Cero strings hardcodeados?
- ¿Claves añadidas en EN y luego ES?

C. Seguridad

- ¿Permisos ABAC/RBAC aplicados?
- ¿Evento de auditoría creado?

D. Blockchain audit

- ¿Se genera hash de evento?
- ¿Se incluye en lote de anchoring?

E. Documentación

- MD del archivo creado/actualizado
- Variables registradas
- DB schema documentado si cambió

---

13) Prompt operativo para Claude Code (cópialo tal cual)

Aquí tienes un prompt directo para que Claude mantenga consistencia:

PROMPT (ES):

Eres Claude Code trabajando en un proyecto modular tipo “Palantir-like” con Core + módulos instalables. Debes seguir estrictamente la “Guía Maestra del Proyecto”.
Reglas innegociables:

1. Nada de textos hardcodeados: todo va por i18n con base en inglés y ficheros separados por idioma.
2. Todo cambio relevante debe emitir un evento auditable y anclarse (hash + lote) para trazabilidad blockchain.
3. Seguridad extrema: RBAC+ABAC, mínimo privilegio, auditoría total.
4. Módulos instalables: Core no importa módulos; módulos se registran por manifiesto y SDK.
5. UI/UX estilo bancario de máxima calidad y accesible.
6. Cada archivo nuevo crea su MD de documentación (ruta, propósito, dependencias, i18n keys, eventos).
7. Cada variable nueva se documenta en /docs/variables/registry.md.
8. Cada cambio de base de datos se documenta en /docs/database/schema.md y, si es módulo, en /docs/database/modules/<id>.md.
   Procedimiento obligatorio antes de codificar: leer /docs/standards/naming.md, /docs/standards/architecture.md, /docs/modules/module-sdk.md, /docs/variables/registry.md, /docs/database/schema.md.
   Entrega siempre: código + docs actualizadas + lista de i18n keys creadas + eventos emitidos + cambios de DB.

PROMPT (EN):

You are Claude Code working on a modular “Palantir-like” platform with a Core base plus installable modules. You must strictly follow the “Project Master Guide”.
Non-negotiable rules:

1. No hardcoded UI/backend text: everything must use i18n keys with English as the base language and per-language files.
2. Every relevant change must emit an auditable event and be blockchain-traceable (event hash + batch anchoring).
3. Extreme security: RBAC+ABAC, least privilege, full audit trail.
4. Installable modules: Core must not import modules; modules register via manifest + SDK contracts.
5. Banking-grade UI/UX with top accessibility.
6. Every new file must create/update its own documentation MD (path, purpose, dependencies, i18n keys, emitted events).
7. Every new variable/constant/flag must be added to /docs/variables/registry.md.
8. Every DB change must be documented in /docs/database/schema.md and, if module-specific, in /docs/database/modules/<id>.md.
   Mandatory procedure before coding: read /docs/standards/naming.md, /docs/standards/architecture.md, /docs/modules/module-sdk.md, /docs/variables/registry.md, /docs/database/schema.md.
   Always deliver: code + updated docs + list of created i18n keys + emitted events + DB changes.

Siguiente paso (sin preguntas, directo)

Para que esta guía sea ejecutable desde ya, lo siguiente que haría es entregarte **3 archivos de documentación iniciales listos para repo**, con plantillas cerradas para que Claude solo "rellene":

1. `/docs/standards/architecture.md` (Core, módulos, contratos, eventos)
2. `/docs/standards/naming.md` (nombres, i18n keys, eventos, tablas)
3. `/docs/modules/module-sdk.md` (interfaces exactas + ejemplo de módulo mínimo)

Si me dices "adelante", te los escribo completos en el siguiente mensaje (copiar/pegar al repo).

añade que tendremos una web que explica, documenta, muestra que hace cada modulo del proyecto, que hace este servicio, tendrá un sistema de selección de planes, compra de modulos escalables, etc... por stripe, 3 planes, 375€, 2350€, 7500€ mes o pagos anuales con 2 meses de descuento.

Perfecto. A continuación **integro y amplío la Guía Maestra del Proyecto** añadiendo **oficialmente la Web Comercial + Sistema de Planes + Marketplace de Módulos**, manteniendo **coherencia arquitectónica, legal y técnica** para que Claude Code la siga sin desviaciones.

Este bloque **no es marketing**, es **parte estructural del sistema**.

EXTENSIÓN DE LA GUÍA MAESTRA

Web Comercial, Marketplace de Módulos y Sistema de Planes

---

14) Web oficial del proyecto (Web Platform)

14.1 Objetivo de la web

La web **NO es solo una landing**. Es:

- Canal comercial
- Centro de documentación pública
- Catálogo vivo de módulos
- Sistema de planes y facturación

- Punto de entrada para clientes enterprise
- Extensión coherente de la plataforma (misma calidad bancaria)

Debe transmitir:

- Solidez
- Transparencia
- Arquitectura real
- Confianza institucional

14.2 Funciones obligatorias de la web

A) Explicación del servicio

- Qué es la plataforma
- Qué problemas resuelve
- Enfoque modular
- Filosofía de seguridad y trazabilidad
- Diferenciación frente a software tradicional

B) Documentación pública (curada)

- Qué hace el Core
- Qué es el motor de riesgos
- Qué es la trazabilidad blockchain
- Qué significa "auditoría inmutable"
- Cómo funcionan los módulos (sin revelar IP crítica)

⚠ **Regla**:
La documentación pública **deriva** de la documentación técnica interna, pero:

- resumida
- explicativa
- orientada a negocio

Nunca duplicada manualmente → debe poder mantenerse alineada.

---

15) Catálogo de módulos (Marketplace)

15.1 Concepto

La plataforma dispone de un **Marketplace de Módulos**, donde cada módulo:

- Se explica claramente
- Tiene ficha propia

- Indica riesgos que cubre
- Indica entidades que analiza
- Indica dependencias
- Indica compatibilidad con planes

Esto refuerza la idea de:

"No compras software, construyes tu sistema."

15.2 Página de cada módulo (estructura)

Cada módulo tendrá una página estándar:

Información funcional

- Qué hace
- Qué riesgos detecta
- Qué tipo de empresas lo usan
- Qué datos necesita
- Qué decisiones permite tomar

Información técnica (light)

- Tipo de módulo (riesgo / operativo / analítico)
- Integración con Core
- Nivel de trazabilidad blockchain
- Nivel de impacto en auditoría

Información comercial

- Incluido en qué planes
- Coste adicional (si aplica)
- Escalabilidad

---

16) Sistema de planes (pricing oficial)

16.1 Planes base (mensuales)

Los planes se aplican **al Core**, no a los módulos.

Plan	Precio mensual	Perfil
Essential	**375€ / mes**	PYMEs, gestorías pequeñas
Professional	**2.350€ / mes**	Empresas medias, grupos
Enterprise	**7.500€ / mes**	Grandes empresas, fintech

16.2 Pago anual

- Pago anual con **2 meses de descuento**
- Equivalente a pagar **10 meses**
- Reflejado claramente en UI y factura

Ejemplo:

- Essential anual → 3.750€
- Professional anual → 23.500€
- Enterprise anual → 75.000€

16.3 Qué incluye cada plan (conceptual)

Essential

- Core básico
- Motor de riesgos
- Dashboard
- Blockchain anchoring nivel estándar
- Nº limitado de módulos activos

Professional

- Core completo
- IA de riesgos avanzada
- Blockchain anchoring avanzado
- Más módulos simultáneos
- Soporte prioritario

Enterprise

- Todo lo anterior
- SLA
- Multi-entorno
- Auditoría extendida
- Integraciones a medida
- Posibilidad de módulos privados

Los módulos **no son el plan**:
el plan define **capacidad**, los módulos definen **funcionalidad**.

---

17) Compra y gestión de módulos

17.1 Compra modular escalable

- Los módulos pueden:
  - Estar incluidos en un plan
  - O comprarse como add-on
- El sistema debe permitir:
  - Activar / desactivar módulos
  - Ver impacto en riesgos
  - Ver consumo de capacidad

17.2 Stripe como sistema de pagos

Obligatorio:

- Stripe Subscriptions
- Stripe Checkout
- Stripe Customer Portal

Funcionalidades:

- Cambio de plan
- Upgrade / downgrade
- Pago mensual / anual
- Gestión de módulos
- Facturación automática
- Webhooks para sincronizar estado

---

18) Integración Stripe ↔ Core Platform

18.1 Entidades clave

Subscription

- tenant_id
- plan_id
- billing_cycle (monthly / yearly)
- status
- stripe_subscription_id

ModuleLicense

- tenant_id
- module_id
- active
- included_in_plan / paid_addon
- limits (si aplica)

18.2 Reglas técnicas

- El Core **no confía en la web**
- El Core valida siempre:
  - Plan activo
  - Módulos permitidos
  - Límites de uso

Stripe → Webhook → Core → Policy Engine

Nada se habilita solo por frontend.

---

19) UX/UI de la web (nivel bancario)

19.1 Diseño

- Misma identidad visual que la app
- Tipografía sobria
- Colores institucionales
- Nada "marketiniano"
- Layout limpio, serio, técnico

Inspiración:

- Banca privada
- Software financiero institucional
- Plataformas regulatorias

19.2 UX principles

- El visitante entiende:
  - qué problema resuelve
  - cómo escala
  - cuánto cuesta
  - qué está comprando
- Transparencia > promesas
- Claridad > hype

---

20) Documentación y coherencia web ↔ producto

Regla crítica

Toda página de módulo y servicio debe poder trazarse a un módulo real del sistema.

Por tanto:

- Cada módulo tiene:
  - `module.json`
  - documentación técnica interna
  - documentación pública asociada
- Nunca se anuncia un módulo inexistente

---

21) Checklist adicional para Claude Code (Web & Billing)

Web

- Textos 100% i18n
- Estructura clara de módulos
- Información alineada con Core real

Planes

- Planes definidos en Core (no solo UI)
- Validación server-side

Stripe

- Webhooks verificados
- Sincronización segura
- Auditoría de cambios de plan

Documentación

- Página web ↔ módulo real
- Pricing documentado
- Cambios reflejados en MD

---

22) Actualización del prompt para Claude Code (añadido)

Añadir al prompt existente:

Además, el proyecto incluye una **Web Platform oficial** que documenta el servicio, explica cada módulo, ofrece un sistema de planes y permite la compra y gestión de módulos mediante Stripe.
La web debe mantener el mismo estándar de seguridad, i18n, calidad UI/UX bancaria y coherencia técnica que la aplicación principal.
Los planes disponibles son: 375€, 2.350€ y 7.500€ mensuales, con opción anual (2 meses de descuento).
Ninguna funcionalidad, módulo o plan puede existir solo a nivel web: todo debe estar validado y aplicado en el Core Platform mediante políticas y licencias.