**Propuesta de Arquitectura MVP SaaS – Zentry**

Fecha: 2025-10-07

# Resumen ejecutivo

Recomendación: Monolito modular multi-tenant con Base de Datos Shared Schema (columna 'tenant\_id'), manteniendo el stack actual (Node 18, Express, Sequelize, TypeScript). Objetivo noviembre: Aislamiento de datos por tenant, subdominios por tenant, auth multi-tenant, despliegue productivo con observabilidad y rollback. Motivo: menor riesgo y tiempo (3–4 semanas) que microservicios, con camino claro de evolución.

* Arquitectura MVP SaaS: Monolito Modular Multi-tenant (diseño hexagonal)
* Base de Datos: Shared Schema con 'tenant\_id' e índices compuestos
* Frontend: Angular 18 se mantiene sin cambios de framework
* Entrega noviembre: listo para vender con aislamiento por tenant

# Alcance

* Backend multi-tenant (middleware de tenant, repositorios con scoping, auth/autorización)
* Migraciones de BD: agregar 'tenant\_id', índices y unicidad por tenant a tablas de negocio
* Subdominios {tenant}.zentry.com y/o header seguro X-Tenant-Id desde el gateway
* CI/CD, logging/metrics por tenant, rate limiting, backups y runbook
* No incluye: cambio de framework frontend ni migración a microservicios en esta fase

# Arquitectura técnica

## Estilo

Monolito modular con principios de Arquitectura Hexagonal/Clean Architecture. Capas: Dominio (reglas por módulo), Aplicación (casos de uso), Infraestructura (Express, Sequelize, adaptadores). Límites bien definidos para futura extracción por 'strangler pattern'.

## Modelo multi-tenant

* DB Shared Schema: columna obligatoria 'tenant\_id' en todas las tablas de negocio y tablas de unión relacionadas
* Índices y unicidad: índices compuestos que incluyan 'tenant\_id'; convertir UNIQUE(col) a UNIQUE(tenant\_id, col)
* Integridad: FKs alineadas al tenant; validación en la capa de dominio para evitar cruces
* Multi-country (si aplica): agregar 'country\_code' como dimensión; índices por (tenant\_id, country\_code, …)

## Resolución de tenant en runtime

* Subdominio {tenant}.zentry.com como fuente principal; fallback por header X-Tenant-Id
* Middleware: resuelve tenantId al inicio y lo inyecta en un contexto por request
* Repositorios/ORM: scoping automático 'WHERE tenant\_id = <contexto>' en lecturas/escrituras

## Autenticación y autorización

* JWT con tenant\_id, user\_id y roles/claims
* Policies/guards por módulo verificando pertenencia al tenant y rol

## Observabilidad y seguridad

* Logging/metrics trazadas por tenantId (y country si aplica)
* Rate limiting por tenant e IP; auditoría de acciones con tenant\_id
* Backups diarios y pruebas de restore; secretos por entorno

# Despliegue y lanzamiento

## Infraestructura

* Contenedores Docker (Node 18) para el backend monolítico; Nginx/Ingress como gateway
* Entornos: dev, staging, prod con mismas imágenes y distinta configuración
* CI/CD: pipeline con tests, migraciones automáticas, despliegue y healthchecks

## Dominios y TLS

* DNS wildcard \*.zentry.com apuntando al gateway
* SSL wildcard con renovación automática
* Enrutamiento por subdominio hacia backend; inyección de X-Tenant-Id si se requiere

## Base de datos

* Migraciones: agregar tenant\_id, índices y unicidades por tenant a las 18 tablas priorizadas
* Backfill: asignación de tenant\_id a datos existentes con criterio definido
* Seeds: creación de tenants y planes/feature flags básicos

## Plan de rollout

* Canary/piloto con 1–2 tenants
* Smoke tests de endpoints críticos por tenant
* Feature flags por tenant/plan si aplica; estrategia de rollback clara

# Cronograma estimado (3–4 semanas)

## Semana 1

* Clasificación de tablas (tenant-owned/global) y migraciones base: tenant\_id, índices y unicidad por tenant
* Middleware de tenant y contexto por request

## Semana 2

* Repositorios con scoping automático; refactor de endpoints críticos
* Tests E2E con 2 tenants para validar aislamiento de datos

## Semana 3

* JWT con tenant\_id y roles; subdominios y gateway en producción
* Observabilidad (logs/metrics por tenant) y rate limiting por tenant

## Semana 4

* Hardening, QA de regresión, optimización de índices, runbooks, canary y salida a prod

# Criterios 'listo para vender' (MVP SaaS)

* Aislamiento de datos por tenant validado en E2E
* Acceso por subdominio y autenticación multi-tenant operativos
* Operabilidad: logs/metrics, rate limit, backups y rollback funcionando
* Onboarding: creación de tenants, usuarios y roles; planes/flags básicos

# Riesgos y mitigación

* Consultas cross-tenant o joins globales: cubrir con scoping obligatorio y tests E2E
* Unicidades globales (p.ej. email): convertir a unicidad por tenant o validar en dominio
* Reportes globales: filtrar explícitamente por tenant o generar vistas por tenant

# Coste y capacidad

* Capacidad suficiente para 20–30 usuarios y 3–10 países por cliente
* Coste operativo bajo: un monolito escalable horizontalmente; monitoreo por tenant

# Evolución futura (post-MVP)

* Schema-per-tenant si se requiere mayor aislamiento/compliance
* Extracción gradual a microservicios por módulos con alto throughput o ciclos propios

# Aclaración clave: ¿es solamente monolito?

En despliegue, sí, es un monolito (una aplicación), lo que simplifica y acelera el MVP. En diseño, es modular/hexagonal con límites definidos; esto permite extraer servicios cuando el producto escale, sin reescribir todo. Es arquitectura SaaS desde el día 1 (multi-tenant, subdominios, auth y operaciones por tenant), aun sin microservicios.

# Frontend (Angular 18)

No se cambia de framework. El frontend sólo debe incluir el tenant en la base URL o enviar X-Tenant-Id; el trabajo principal está en backend + despliegue.