Plataforma SaaS Multi-Tenant de Facturación Electrónica

📌 Visión del Proyecto

Construir una **plataforma SaaS empresarial** que permita a múltiples empresas gestionar su facturación, clientes, suscripciones y pagos desde un único sistema seguro, escalable y moderno.

El objetivo principal es demostrar habilidades reales de ingeniería de software: arquitectura moderna, modelado de dominio complejo, seguridad, escalabilidad e integración con servicios externos.

Este proyecto debe sentirse como un producto que podría venderse.

🕑 Objetivos Técnicos

- Diseñar una arquitectura profesional lista para producción.
- Implementar multi-tenancy real.
- Construir lógica de negocio compleja (no solo CRUD).
- Aplicar buenas prácticas de seguridad.
- Integrar servicios externos (ej: DIAN sandbox).
- Utilizar infraestructura moderna.
- Mantener código limpio, testeable y documentado.

Descripción Técnica (para CV / LinkedIn)

Plataforma SaaS multi-tenant de facturación electrónica construida con Java y Spring Boot, basada en una arquitectura orientada a eventos, con control de concurrencia, seguridad robusta e integración con servicios externos para gestión fiscal. Diseñada para soportar múltiples organizaciones con aislamiento de datos y alta escalabilidad.

💣 Arquitectura General

Estilo arquitectónico

👉 **Modular Monolith (recomendado para portafolio)**

¿Por qué? - Más fácil de construir solo. - Demuestra madurez arquitectónica. - Permite evolucionar a microservicios.

Módulos sugeridos:

- Identity & Access
- Tenant Management
- Billing Core
- Payments
- Subscription Engine
- Notification Service
- Reporting
- External Integrations

⚙️ Stack Tecnológico

Backend

- **Java 21**
- **Spring Boot 3**
- Spring Security
- Spring Data JPA
- Hibernate

UI

- **Vaadin** (para panel administrativo empresarial)

Base de Datos

- PostgreSQL

Infraestructura

- Docker
- Docker Compose

Testing

- JUnit
- Testcontainers
- Mockito

Documentación API

- OpenAPI / Swagger

📀 Stack PRO (Altamente Diferenciador)

No es obligatorio desde el inicio — puedes incorporarlo progresivamente.

- Redis (caching)
- RabbitMQ o Kafka (event-driven)
- GitHub Actions (CI/CD)
- Deploy en cloud (Render, Railway, AWS, Fly.io, etc.)

Multi-Tenancy (Pieza Clave del Proyecto)

Estrategia recomendada:

👉 **Database per tenant (ideal para demostrar nivel)**

Alternativa más simple: 👉 Schema por tenant.

Cada empresa debe tener: - Usuarios aislados - Facturas aisladas - Configuración propia - Impuestos propios

Esto impresiona mucho a reclutadores.

🔐 Seguridad

Implementar:

- JWT Authentication
- RBAC (Role-Based Access Control)

Roles ejemplo:

- Owner
- Admin
- Accountant
- Viewer

Extras recomendados: - Rate limiting - Password hashing (BCrypt) - Auditoría de acciones

---

📦 Modelado de Dominio (Core del Sistema)

Entidades principales

Tenant

Empresa dentro de la plataforma.

User

Usuarios asociados a un tenant.

Customer

Clientes de la empresa.

Invoice

Debe incluir: - estado - impuestos - moneda - fechas - items - totales

InvoiceItem

Productos o servicios facturados.

Payment

Registro de pagos.

Subscription

Planes recurrentes.

Plan

Definición de precios.

🧾 Lógica de Negocio Avanzada

Aquí es donde te diferencias.

Implementa cosas como:

- Estados de factura (draft, issued, paid, overdue)
- Cálculo de impuestos
- Prorrateo de suscripciones
- Reintentos de pago
- Generación automática de facturas

• Notificaciones

Esto NO es CRUD.

Es ingeniería.

📡 Arquitectura Orientada a Eventos

Ejemplo de flujo:

Factura creada → Evento → - Generar PDF - Enviar email - Notificar webhook - Actualizar métricas

Beneficios: - Bajo acoplamiento - Escalabilidad - Diseño moderno

---

🕐 Control de Concurrencia

Muy pocos portafolios incluyen esto.

Implementa:

👉 Optimistic Locking con `@Version`

Ejemplo: Evitar que dos usuarios editen la misma factura simultáneamente.

Esto grita nivel profesional.

🕐 Integraciones Externas

Altamente recomendado

👉 Integración con **DIAN (sandbox)**

No tiene que ser perfecta. Solo demostrar capacidad de integración.

Opcionales: - Stripe - PayPal - Servicio de emails

📊 Panel Empresarial (Vaadin)

Construye una UI limpia y profesional:

Dashboard

- ingresos
- facturas pendientes
- MRR
- churn

Módulos UI

- Clientes
- Facturación
- Suscripciones
- Reportes
- Usuarios

Enfócate en usabilidad empresarial. No necesitas animaciones.

🧪 Testing (Te hace ver Senior)

Prioriza:

- Tests de servicios
- Tests de repositorios
- Tests de integración

Evita solo tests triviales.

🗜️ DevOps Básico

Mínimo recomendado:

- Dockerfile
- docker-compose

Nivel superior:

- CI/CD pipeline
- Deploy automático

📈 Cómo Hacer que el Repo Impresione

Incluye:

README profesional

Con: - arquitectura - diagramas - decisiones técnicas - tradeoffs

Diagramas

Usa: - C4 - ERD

Commits claros

Evita: "fix stuff"

Prefiere: "implement tenant isolation strategy"

🕐 Features que Te Ponen en el Top 10%

No necesitas todas — con 2 ya destacas mucho.

- Event-driven
- Cache con Redis
- Multi-tenancy real
- Auditoría
- Webhooks
- Exportación a PDF
- Feature flags

🖨️ Roadmap Sugerido

Fase 1 — Base

- Setup del proyecto
- Seguridad
- Tenant
- Usuarios

Fase 2 — Billing Core

- Clientes
- Facturas
- Items

Fase 3 — Lógica avanzada

- impuestos
- estados

- pagos

Fase 4 — Eventos

- mensajería
- notificaciones

Fase 5 — Infra

- Docker
- deploy

Fase 6 — Integración externa

- DIAN sandbox

🖍️ Regla de Oro

No construyas esto como tarea.

Constrúyelo como si una empresa dependiera de él.

Ese mindset cambia tu carrera.

💡 Nombre del Proyecto (Opcional)

Algunas ideas:

- NimbusBilling
- TenantFlow
- LedgerCloud
- OrbitInvoice
- CoreBill

Elige algo que suene a producto real.

🪠 Resultado Esperado

Al terminar este proyecto deberías poder decir:

👉 "Puedo diseñar y construir sistemas empresariales modernos."

Y los reclutadores lo van a notar.