Guia Completo – Projeto SaaS Multi-Tenant em Java

Objetivo

Este guia serve como blueprint para criar um projeto SaaS multi-tenant de nível profissional usando Java e Spring Boot. Ideal para portfólio e entrevistas para vagas de desenvolvedor backend Java.

Conceitos principais

SaaS: Software como serviço, acessado via internet.
Multi-tenant: Várias empresas usam a mesma aplicação com dados isolados.
Exemplo: sistema de agendamento onde cada empresa possui seus próprios dados.

Projeto sugerido

SaaS de agendamento para clínicas, barbearias ou consultores. Cada empresa é um tenant isolado.

Módulos principais

Módulo	Descrição
Auth	Login, registro, JWT e roles
Tenant	Cadastro de empresas e planos
Scheduling	Serviços, agenda e horários
Billing (simples)	Plano free/premium e limites

Modelagem Multi-tenant

Abordagem recomendada: isolamento por coluna (tenant_id). Todas as entidades possuem tenant_id para garantir separação de dados.

Implementação no Spring

1. Criar TenantContext (ThreadLocal).
2. Resolver tenant via filtro/interceptor.
3. Incluir tenant_id nas queries ou usar filtros do Hibernate.

Segurança

Spring Security + JWT contendo userId, tenantId e roles. Garante autenticação e isolamento por empresa.

Stack recomendada

Backend: Spring Boot, Spring Security, JPA, PostgreSQL.
Infra: Docker, Docker Compose, AWS EC2.
Extras: Redis, Flyway, S3.

Estrutura de pastas

com.seusaaas
- auth
- tenant
- scheduling
- billing
- shared (security, tenancy, config)

Diferenciais que impressionam recrutadores

• Multi-tenant funcional
• Deploy online
• Swagger
• Testes automatizados

• Logs estruturados
• Cache por tenant

Deploy

Use Docker + AWS EC2. Opcionalmente configure subdomínios ou simule com tenant.localhost.

Roadmap de 8 semanas

Semanas 1-2: Auth + JWT
Semana 3: Tenant base
Semanas 4-5: Multi-tenant funcional
Semana 6: Agendamento
Semana 7: Docker + deploy
Semana 8: README e testes

Como apresentar no currículo

Projeto SaaS multi-tenant com Spring Boot, isolamento por tenant_id, autenticação JWT, deploy em AWS e containerização com Docker.

Dica final

Não precisa ser perfeito. Precisa ser funcional, bem documentado e online. Clareza impressiona mais que complexidade.