flowchart TD

A[Requisição HTTP] --> B{Header x-tenant?}

B -- Sim --> C[Slug = header]

B -- Não --> D[Extrai subdomínio<br/>do hostname]

D --> E{Slug encontrado?}

E -- Não --> F[Lança 400: slug não informado]

C --> G[Prisma main busca tenant por slug]

E -- Sim --> G

G --> H{Tenant existe?}

H -- Não --> I[Lança 400: tenant inexistente]

H -- Sim --> J[Obtém cliente Prisma do tenant<br/>via TenantPrismaManager]

J --> K[Enriquece Request (tenantId, tenantClient)]

K --> L[Controller NestJS]

L --> M[Service do módulo (ex.: TCliService)]

M --> N[CRUD via Prisma do tenant]

N --> O[Retorno da API]

flowchart TD

A[POST /tenants<br/>{ name, slug }] --> B[TenantsController]

B --> C[TenantsService.createTenant]

C --> D[Compoõe dbName = tenant\_slug]

D --> E[Prisma main verifica slug/db duplicados]

E --> F{Já existe?}

F -- Sim --> G[Lança 400: slug em uso]

F -- Não --> H[Usa Pool administrativo<br/>para executar CREATE DATABASE<br/>com template PG\_TEMPLATE]

H --> I{CREATE DATABASE ok?}

I -- Não --> J[Lança erro ao criar banco]

I -- Sim --> K[Registra tenant no banco main]

K --> L[Retorna dados do novo tenant]

**Visão Geral**

* package.json:1 mostra um backend NestJS 11 com Prisma 6, PostgreSQL (pg) e validação via class-validator; scripts padrão de build/test e tarefa migrate:tenants para propagar schemas multi-tenant.
* src/app.module.ts:26 carrega configuração global (ConfigModule) e registra módulos de domínio (TenantsModule, TCliModule, TUsersModule) além de serviços core como PrismaService e TenantPrismaManager.

**Bootstrap e Configuração**

* src/main.ts:4 sobe a aplicação com NestFactory, habilita CORS e escuta na porta definida em process.env.PORT.
* .env:5 concentra URLs e credenciais do Postgres (main, template e dados administrativos); mantenha esse arquivo fora de versionamento público.

**Multi-tenancy e Prisma**

* prisma/schema.prisma:1 descreve o banco “main” com tabela Tenant, responsável por mapear slug → db\_name.
* src/prisma.service.ts:1 expõe um PrismaClient conectado ao banco main.
* src/tenant-prisma.manager.ts:7 instancia e memoriza PrismaClient específicos de cada banco de tenant usando credenciais do .env.
* src/middleware/tenant-context.middleware.ts:17 intercepta requisições, resolve o tenant via header x-tenant ou subdomínio, valida no banco main e injeta tenantClient (Prisma do ERP) e tenantId no objeto Request.
* prisma/erp.prisma:1 é o schema do ERP template (centenas de tabelas legadas) e gera o client multi-tenant em prisma/generated/erp.

**Gestão de Tenants**

* src/tenants/tenants.controller.ts:7 expõe POST /tenants para criar novos tenants.
* src/tenants/tenants.service.ts:45 clona o banco template (PG\_TEMPLATE) para tenant\_<slug>, valida duplicidade e registra metadados no banco main; encerra o pool administrativo no onModuleDestroy.

**Módulos de Domínio**

* src/t\_cli/TCli.controller.ts:7 e src/t\_cli/TCli.service.ts:8 exemplificam o padrão: controllers recebem RequestWithPrisma e delegam a services que operam diretamente sobre o PrismaClient específico do tenant (CRUD para t\_cli).
* src/t\_users/TUsers.controller.ts:7 e src/t\_users/TUsers.service.ts:8 seguem a mesma estrutura para t\_users; demais diretórios (t\_emp, t\_comanda, etc.) repetem o padrão com DTOs class-validator armazenados em src/<módulo>/dto.

**Automação e Operações**

* Docs.txt:1 documenta o fluxo de manutenção de schema ERP: prisma migrate dev no template, migrate deploy, prisma generate e npm run migrate:tenants.
* scripts/migrate-tenants.ts:14 percorre os tenants cadastrados, sobrescreve ERP\_TEMPLATE\_URL com cada banco físico e roda prisma db push --accept-data-loss para alinhar todos os schemas.