# 🤖 PROMPT DE EXECUÇÃO: SALES OS ULTIMATE

Atue como: Senior Principal Software Architect & CTO Contexto: Você deve iniciar a construção do Sales OS Ultimate, um ecossistema SaaS B2B Omnichannel (Voz, IA, CRM). Diretriz: Siga a metodologia Tree-of-Thought. Explore as dependências de cada ciclo antes de gerar o código. Use linguagem direta e técnica.

🚀 DIRETIVA DE ARQUITETURA E EXECUÇÃO: SALES OS ULTIMATE (EXPANDIDO)Entidade: Senior Principal Software Architect & CTO

Missão: Fusão e blindagem do Sales Prospector AI + Voice AI Agent.Objetivo: Criar o SaaS B2B mais robusto, escalável e performático do mercado global, estabelecendo um novo padrão de excelência técnica.

Status: APROVADO PARA EXECUÇÃO IMEDIATA🗺️

PARTE 1: EXECUÇÃO TÁTICA DETALHADA (16 CICLOS / 155 ITENS)

PARTE 2: VISÃO ESTRATÉGICA E OPTIMIZAÇÕES TÉCNICAS

## 🗺️ PARTE 1: EXECUÇÃO TÁTICA DETALHADA (16 CICLOS / 155 ITENS)

### 🟢 CICLO 1: BLINDAGEM E FUNDAÇÃO ARQUITETURAL

**Foco:** Segurança Zero-Trust, Consistência de Dados e Organização Modular do Código.

**1. Monorepo Estruturado e Modular** 📍 Local: `/` (Raiz) 💻 **Diretriz Técnica:** Implementar arquitetura Monorepo utilizando Nx ou Turborepo com regras de "Strict Boundaries". A estrutura deve separar rigidamente `apps` (deployables: web, api, worker) de `libs` (reusable: shared-types, ui-kit, core-logic). O objetivo é compartilhar interfaces TypeScript e lógica de negócio sem criar acoplamento circular ou dependências implícitas. 🛡️ **Gestão de Falhas:** Para evitar o "Dependency Hell" e conflitos de versões, utilizar o `syncpack` no pipeline de CI/CD. Ele deve forçar versões únicas de pacotes críticos (ex: React, NestJS, Prisma) em todo o repositório, quebrando o build se houver discrepâncias.

**2. Backend BFF (Backend for Frontend) & API Gateway** 📍 Local: `apps/api/src/main.ts` 💻 **Diretriz Técnica:** Centralizar todo o acesso externo no Backend. O frontend deve ser "burro" em relação a serviços terceiros. Implementar um padrão de Gateway que orquestra chamadas para Gemini, Vapi, Stripe, etc., transformando e sanitizando os dados antes de entregá-los ao cliente. 🛡️ **Gestão de Falhas:** Falhas críticas no Boot (ex: conexão DB) devem ser tratadas com um `try/catch` fatal no bootstrap, emitindo um log JSON estruturado para stdout antes de encerrar o processo, garantindo que o orquestrador (K8s) saiba o motivo exato da falha.

**3. Gestão de Segredos e Configuração (.env)** 📍 Local: `apps/api/.env` 💻 **Diretriz Técnica:** Eliminar qualquer segredo do código client-side. Migrar chaves ( `GEMINI_KEY` , `SUPABASE_KEY` , `STRIPE_SECRET` ) para injeção via variáveis de ambiente no servidor. Utilizar bibliotecas como

`dotenv-safe` ou validação via Zod ( `env.validation.ts` ) para garantir tipagem forte das configs. 🛡️ **Gestão de Falhas:** Se uma chave crítica estiver faltando no startup, o servidor deve abortar a inicialização imediatamente ( `process.exit(1)` ). É preferível não subir a subir com configuração parcial e insegura.

**4. Rate Limiting Global e Distribuído** 📍 Local: `apps/api/src/guards` 💻 **Diretriz Técnica:** Implementar `ThrottlerGuard` baseado em Redis. Definir limites granulares: 100 req/min por IP para rotas públicas e limites específicos por Tenant para rotas autenticadas, protegendo contra ataques DDoS e abuso de recursos (Noisy Neighbors). 🛡️ **Gestão de Falhas:** Se o Redis cair, o sistema deve fazer fallback automático para um Rate Limiter em memória (In-Memory Map) temporário, garantindo proteção básica e evitando que a API pare de responder por dependência de infra.

**5. Validação Zod Global e Sanitização** 📍 Local: `apps/api/src/pipes` 💻 **Diretriz Técnica:** Ativar `app.useGlobalPipes(new ZodValidationPipe({ whitelist: true }))` . Validar rigorosamente Body, Query e Params contra schemas Zod. A opção `whitelist` deve remover automaticamente quaisquer propriedades não declaradas no DTO, prevenindo injeção de dados sujos. 🛡️ **Gestão de Falhas:** Payloads inválidos devem retornar estritamente `400 Bad Request` com uma mensagem de erro sanitizada e legível para o front, removendo stack traces ou detalhes internos da estrutura do banco.

**6. Orquestração Local com Docker Compose** 📍 Local: `docker-compose.yml` 💻 **Diretriz Técnica:** Definir serviços isolados em rede interna ( `internal-network` ): Postgres, Redis, API, Web, Worker e Mailhog. Garantir paridade total entre o ambiente de desenvolvimento e produção para eliminar o clássico "funciona na minha máquina". 🛡️ **Gestão de Falhas:** Conflitos de porta são comuns. Utilizar variáveis de ambiente dinâmicas no `.env` (ex: `${PORT:-3000}` ) para permitir que desenvolvedores ajustem portas locais sem alterar o arquivo versionado.

**7. Logs Estruturados (JSON) e Contextuais** 📍 Local: `libs/shared/src/logger` 💻 **Diretriz Técnica:** Implementar logger com Winston ou Pino configurado para saída JSON. Cada log deve conter metadados contextuais obrigatórios: `trace_id` , `tenant_id` , `user_id` , `level` e `context` . Isso permite ingestão e query eficiente em ferramentas como Datadog ou Elastic. 🛡️ **Gestão de Falhas:** Para evitar estouro de disco em ambientes sem log driver configurado, implementar rotação de logs automática ( `logrotate` ) dentro do container ou configurar o driver de log do Docker para limitar o tamanho do arquivo.

**8. Exception Filter Global Padronizado** 📍 Local: `apps/api/src/filters` 💻 **Diretriz Técnica:** Criar um filtro global que captura `HttpException` e erros não tratados. A resposta para o cliente deve seguir o padrão RFC 7807 (Problem Details), mascarando stack traces de erros 500 em produção. 🛡️ **Gestão de Falhas:** Erros desconhecidos devem ser logados com severidade `ERROR` e retornar uma mensagem genérica "Internal Server Error" acompanhada de um `trace_id` para suporte, sem expor vulnerabilidades de código.

**9. Health Checks e K8s Probes** 📍 Local: `apps/api/src/health` 💻 **Diretriz Técnica:** Implementar endpoints `/health/live` (Liveness Probe: o app está rodando?) e

`/health/ready` (Readiness Probe: DB e Redis estão conectados?). O K8s usa isso para reiniciar pods travados ou remover pods não prontos do balanceador de carga. 🛡️ **Gestão de Falhas:** Dependências lentas não devem derrubar o pod imediatamente. Configurar timeouts agressivos (ex: 3s) na verificação de saúde para evitar que um DB lento cause um restart loop em cascata na API.

**10. Path Aliases e Mapeamento TypeScript** 📍 Local: `tsconfig.base.json` 💻 **Diretriz Técnica:** Configurar mapeamento estrito de `@shared/*` para `libs/shared/src/*` e `@modules/*` para módulos de domínio. Isso facilita refatoração e evita importações relativas frágeis e ilegíveis como `../../../../utils`. 🛡️ **Gestão de Falhas:** O build deve falhar se forem detectadas importações relativas que violem os limites dos módulos. Adicionar regra no ESLint ( `no-restricted-imports` ) para proibir padrões `../` profundos.

## 🟢 CICLO 2: IDENTITY, ACCESS CONTROL & MULTI-TENANCY

**Foco:** Isolamento Lógico de Dados, Autenticação Robusta e Controle de Acesso.

**11. Tabela Organizations (Tenancy)** 📍 Local: `prisma/schema.prisma` 💻 **Diretriz Técnica:** Modelar a entidade `Organization` como a raiz de todos os dados. Campos: `{ id, name, slug, plan, stripeId, settings (JSONB) }`. Todo dado no sistema deve ter uma FK direta ou indireta para esta tabela. 🛡️ **Gestão de Falhas:** Migrações de banco críticas podem falhar em produção. Implementar rollback automático no pipeline de CD se o comando `prisma migrate deploy` retornar erro.

**12. Tabela Members e Associação** 📍 Local: `prisma/schema.prisma` 💻 **Diretriz Técnica:** Modelar `Member` como tabela de junção `{ userId, orgId, role, status }`. Criar índice composto `@@unique([userId, orgId])` para garantir unicidade e performance em queries de pertinência. 🛡️ **Gestão de Falhas:** Evitar registros órfãos configurando Foreign Keys com `OnDelete: Cascade`. Se uma organização for deletada, todos os membros associados devem ser limpos automaticamente pelo motor do banco.

**13. Sincronização Supabase Auth (Webhooks)** 📍 Local: `supabase/functions` 💻 **Diretriz Técnica:** Utilizar Webhooks do Supabase Auth ( `on_auth_user_created` ) para chamar a API interna e criar o registro `User` no Postgres local. Isso mantém o banco de aplicação sincronizado com o provedor de identidade. 🛡️ **Gestão de Falhas:** Webhooks podem falhar. Configurar a Retry Queue do Supabase (Dead Letter Queue) para garantir entrega eventual, ou implementar um job cron de reconciliação diária de usuários.

**14. RLS Rigoroso (Row Level Security)** 📍 Local: `supabase/migrations` 💻 **Diretriz Técnica:** Para acesso direto via Supabase Client (se usado), implementar `CREATE POLICY` forçando o filtro `org_id`. A policy deve extrair o `org_id` dos claims do JWT ( `auth.jwt() -> 'app_metadata' -> 'org_id'` ). 🛡️ **Gestão de Falhas:** O vazamento de dados entre tenants é inaceitável. Executar testes de penetração automatizados no CI que tentam acessar dados da "Org B" usando credenciais da "Org A". O teste deve falhar se retornar qualquer registro.

**15. Sistema de Convites Seguro** 📍 Local: `apps/api/src/auth` 💻 **Diretriz Técnica:** Gerar links de convite contendo JWTs assinados com segredo do servidor, contendo `{ email,`

`orgId, role }` e validade estrita de 48h. Não armazenar o token em texto plano, apenas o hash ou validar a assinatura stateless. 🛡️ **Gestão de Falhas:** Se o token expirar ou for inválido, redirecionar o usuário para uma página amigável com a ação clara "Solicitar novo convite", evitando telas de erro genéricas.

**16. RBAC (Role-Based Access Control) Declarativo** 📍 Local: `apps/api/src/decorators` 💻 **Diretriz Técnica:** Criar Decorator `@Roles('admin', 'manager', 'sdr')` e um Guard global. O Guard deve interceptar a request, ler o usuário do contexto e validar se sua role na organização atual permite o acesso ao recurso. 🛡️ **Gestão de Falhas:** Tentativas de acesso não autorizado devem retornar `403 Forbidden` com mensagem padronizada, e o incidente deve ser logado como evento de segurança no Audit Log.

**17. Middleware de Tenant Context** 📍 Local: `apps/api/src/middleware` 💻 **Diretriz Técnica:** Middleware global que extrai o header `X-Tenant-ID`. Ele deve validar se o usuário autenticado realmente pertence àquela organização e injetar o objeto `Organization` no contexto da Request ( `req.org` ). 🛡️ **Gestão de Falhas:** Prevenção de Spoofing: Nunca confiar apenas no header. O middleware deve cruzar o `X-Tenant-ID` com a lista de organizações permitidas no token ou no banco de dados do usuário.

**18. Audit Logs Imutáveis e Assíncronos** 📍 Local: `apps/api/src/audit` 💻 **Diretriz Técnica:** Implementar serviço de auditoria que insere logs de forma assíncrona (Fire-and-Forget ou via Fila) para não impactar a latência da request principal. Schema: `{ actorId, action, targetResource, diff, metadata, ip, userAgent }`. 🛡️ **Gestão de Falhas:** Tabela de logs cresce exponencialmente. Utilizar Postgres Partitioning (particionamento por mês/ano) desde o dia 1 para garantir performance de escrita e facilidade de arquivamento.

**19. Sessão Híbrida (Secure Cookies + JWT)** 📍 Local: `apps/api/src/auth` 💻 **Diretriz Técnica:** Armazenar o `Refresh Token` (longa duração) em Cookie `HttpOnly` (inacessível via JS). O `Access Token` (curta duração) fica em memória no cliente. Isso previne roubo de sessão via XSS e permite rotação silenciosa de tokens. 🛡️ **Gestão de Falhas:** Cookies devem ter flags obrigatórias: `Secure` (apenas HTTPS), `SameSite=Strict` (previne CSRF) e `HttpOnly`.

**20. Proteção CORS Estrita** 📍 Local: `apps/api/src/main.ts` 💻 **Diretriz Técnica:** Configurar CORS aceitando apenas domínios whitelisted exatos (ex: `app.salesos.com` ). Proibir o uso de wildcards `*` ou regex permissivos em ambiente de produção. 🛡️ **Gestão de Falhas:** Requests sem header `Origin` (comuns em scripts server-side maliciosos) ou de origens não autorizadas devem ser bloqueadas imediatamente pelo middleware antes de processar qualquer lógica.

### 🟢 CICLO 3: MOTOR DE IA ENTERPRISE (TEXTO)

**Foco:** Inteligência Contextual, Abstração de Provedores e Otimização de Custos.

**21. Gateway Único de IA (Adapter Pattern)** 📍 Local: `apps/api/src/ai` 💻 **Diretriz Técnica:** Criar uma camada de abstração ( `AIService` ) que implementa o padrão Adapter. O código de negócio chama `ai.generate()`, e o adaptador decide se rota para OpenAI, Gemini ou Anthropic baseado em configuração ou disponibilidade. 🛡️ **Gestão de Falhas:** Implementar

Circuit Breaker. Se o provedor principal (ex: OpenAI) der timeout ou erro 5xx consecutivamente, o tráfego deve ser roteado automaticamente para o provedor de backup (ex: Azure/Gemini).

**22. Streaming Real-time (Server-Sent Events)** 📍 Local: `apps/api/src/ai` 💻 **Diretriz Técnica:** Utilizar Server-Sent Events (SSE) para streamar a resposta da IA token a token. Usar `res.write('data: chunk')` e manter a conexão aberta. Isso melhora drasticamente a percepção de velocidade (Time to First Byte). 🛡️ **Gestão de Falhas:** Quedas de rede interrompem o stream. O cliente deve ser capaz de reconectar enviando o `Last-Event-ID`, e o servidor deve (idealmente) retomar ou indicar que a geração precisa ser reiniciada.

**23. Cache Semântico (Redis + Vector)** 📍 Local: `apps/api/src/ai` 💻 **Diretriz Técnica:** Antes de chamar a LLM, gerar um embedding da pergunta e buscar no Redis/Vector DB se existe uma pergunta semanticamente idêntica respondida recentemente. Se similaridade > 0.95, retornar cache. 🛡️ **Gestão de Falhas:** O cache pode ficar obsoleto se os dados do cliente mudarem (RAG). Implementar invalidação forçada de cache (tag-based invalidation) sempre que documentos de contexto forem atualizados.

**24. Prompt Manager Centralizado (DB)** 📍 Local: `apps/api/src/prompts` 💻 **Diretriz Técnica:** Armazenar templates de prompts no banco de dados ( `slug`, `template`, `variables` ). Nunca hardcode prompts no TypeScript. Isso permite que Engenheiros de Prompt iterem sem precisar de deploy de código. 🛡️ **Gestão de Falhas:** Consultar DB a cada request é lento. Cachear os templates de prompt compilados na memória da API (LRU Cache) com TTL curto (ex: 5 min).

**25. Versionamento de Prompts** 📍 Local: `apps/api/src/prompts` 💻 **Diretriz Técnica:** Adicionar coluna `version` e `is_active`. A API deve suportar queries como `?v=latest` ou `?v=1.2`. Isso permite A/B testing de prompts e rollback instantâneo em caso de regressão na qualidade da IA. 🛡️ **Gestão de Falhas:** Regressão de Prompt (IA começa a alucinar). Ter um botão de "Rollback to Previous Stable" no painel administrativo que reverte a versão `active` instantaneamente.

**26. RAG Pipeline de Ingestão** 📍 Local: `apps/worker/src/rag` 💻 **Diretriz Técnica:** Pipeline ETL robusto usando LangChain. Passos: Extração de Texto (PDF/Docx) -> Limpeza -> Splitting (RecursiveCharacterTextSplitter) -> Embedding -> Upsert no Vector DB. 🛡️ **Gestão de Falhas:** Arquivos corrompidos ou protegidos por senha travam o worker. Envolver o processamento em try/catch, logar o erro específico, pular o arquivo e notificar o usuário na UI ("Falha ao processar arquivo X").

**27. Busca Vetorial Híbrida (PgVector)** 📍 Local: `apps/api/src/search` 💻 **Diretriz Técnica:** Utilizar PgVector no Postgres. Realizar busca híbrida: Keyword Search (BM25 para termos exatos) + Vector Search (Cosine Similarity para contexto). Re-rankear resultados para melhor precisão. 🛡️ **Gestão de Falhas:** Queries vetoriais sequenciais são lentas em grandes datasets. Criar índices `HNSW` (Hierarchical Navigable Small World) no Postgres para performance aproximada ultra-rápida, em vez de `IVFFlat` ou scan sequencial.

**28. Fallback de Modelos em Cascata** 📍 Local: `libs/shared/src/ai` 💻 **Diretriz Técnica:** Lógica de retry inteligente: `try { callGeminiFlash() } catch { callGPT4o() }`. Tentar modelo mais barato/rápido primeiro; se falhar ou a qualidade for baixa (detectada via validação), escalar para modelo mais robusto. 🛡️ **Gestão de Falhas:** Monitorar custos. Se o fallback para o modelo caro for ativado em mais de 10% das chamadas, disparar alerta para o time de engenharia investigar a estabilidade do modelo primário.

**29. Detecção e Mitigação de Alucinação** 📍 Local: `apps/api/src/ai` 💻 **Diretriz Técnica:** Implementar passo de verificação (Self-Correction/Reflection). Após a geração, fazer uma chamada rápida de IA menor: "A resposta X é suportada pelos fatos Y do contexto? Responda Sim/Não". Se Não, regenerar. 🛡️ **Gestão de Falhas:** Falsos positivos no filtro podem bloquear respostas úteis. Permitir que o usuário "Vote" na resposta para flaggar falsos positivos e usar esses dados para fine-tuning do avaliador.

**30. Calculadora de Tokens e Estimativa de Custo** 📍 Local: `libs/shared/src/utils` 💻 **Diretriz Técnica:** Utilizar bibliotecas de tokenização ( `tiktoken`, `gpt-tokenizer` ) no backend para contar tokens exatos antes de enviar a request. Armazenar o custo estimado baseado na tabela de preços do model provider. 🛡️ **Gestão de Falhas:** Discrepância de faturamento. Realizar reconciliação noturna (job cron) comparando os logs de uso internos com a API de usage do OpenAI/Google para detectar vazamentos ou erros de cálculo.

### 🟢 CICLO 4: AGENTE DE VOZ & TELEFONIA

**Foco:** Comunicação Real-time, Baixa Latência e Robustez de Rede.

**31. WebSocket Gateway Dedicado** 📍 Local: `apps/api/src/gateway` 💻 **Diretriz Técnica:** Configurar um namespace Socket.io separado `/voice` otimizado para tráfego binário e alta frequência. Usar MessagePack em vez de JSON para serialização de dados de áudio se necessário. 🛡️ **Gestão de Falhas:** Alta latência ou perda de pacotes (Jitter). Monitorar métricas de conexão. Se o Jitter > 200ms, enviar evento para o front exibir alerta "Conexão Instável" e degradar graciosamente (ex: desligar vídeo ou recursos extra).

**32. Integração VAPI/Twilio com Webhooks Seguros** 📍 Local: `apps/api/src/voice` 💻 **Diretriz Técnica:** Endpoints dedicados para receber eventos de telefonia: `call.initiated`, `call.speaking`, `call.ended`. Processar metadados da chamada em tempo real para atualizar o CRM. 🛡️ **Gestão de Falhas:** Segurança de Webhooks é crítica. Validar obrigatoriamente a assinatura criptográfica (HMAC-SHA256) do provedor (Vapi/Twilio) para garantir que a requisição é legítima e não um ataque simulado.

**33. Máquina de Estados de Chamada (XState)** 📍 Local: `libs/shared/src/voice` 💻 **Diretriz Técnica:** Modelar o ciclo de vida da chamada com XState: `Idle -> Dialing -> Ringing -> Connected` ( `Speaking` / `Listening` ) `-> Ending -> Ended`. O estado deve ser a única fonte de verdade para a UI. 🛡️ **Gestão de Falhas:** Estados zumbis (travado em "Dialing"). Implementar timeouts de guarda em cada transição. Se ficar em `Dialing` por > 60s sem evento, forçar transição para `Idle` e notificar erro.

**34. Interrupção Inteligente (VAD - Voice Activity Detection)** 📍 Local: `apps/web/src/voice` 💻 **Diretriz Técnica:** Implementar VAD local no navegador (via WebAssembly ou Worklet de Áudio). Ao detectar fala do usuário enquanto a IA fala, emitir evento `interrupt` imediato para o servidor parar o stream de áudio (Clearing the buffer). 🛡️ **Gestão de Falhas:** Ruído de fundo disparando interrupções. Implementar ajuste dinâmico de limiar de ruído (Noise Gate) ou permitir que o usuário ajuste a sensibilidade do microfone.

**35. Componente Active Call Persistente (Overlay)** 📍 Local: `apps/web/src/layout` 💻 **Diretriz Técnica:** O componente de chamada deve viver fora da árvore de rotas principal do React (em um Context Provider no topo ou via Portals), garantindo que a chamada não caia se o usuário navegar entre páginas do CRM. 🛡️ **Gestão de Falhas:** Fechamento acidental da aba. Adicionar listener `window.onbeforeunload` que dispara um popup nativo de confirmação "Há uma chamada ativa. Deseja realmente sair?" se o estado for `Connected` .

**36. Buffer de Áudio e Transcodificação** 📍 Local: `apps/worker/src/voice` 💻 **Diretriz Técnica:** Workers de processamento de áudio devem lidar com conversão de formatos (ulaw/alaw <-> pcm/mp3) em tempo real. Utilizar Streams do Node.js para processar chunks sem carregar o áudio todo na RAM. 🛡️ **Gestão de Falhas:** Buffer Overflow (latência acumulada). Se a fila de processamento de áudio ficar muito grande, descartar pacotes antigos (Drop Frames) para priorizar o áudio "do agora" e reduzir o delay percebido.

**37. Gravação e Upload Automático (Stream)** 📍 Local: `apps/api/src/storage` 💻 **Diretriz Técnica:** Não esperar a chamada terminar para começar o upload. Streamar a gravação diretamente para S3 (Multipart Upload) ou usar Presigned URLs conforme os chunks chegam. 🛡️ **Gestão de Falhas:** Falha no upload para S3. Salvar temporariamente em disco local (volume persistente) e criar um Job de Retry no BullMQ para subir o arquivo assim que a conexão restabelecer.

**38. Análise de Sentimento em Tempo Real (NLP)** 📍 Local: `apps/worker/src/nlp` 💻 **Diretriz Técnica:** Processar transcrições conforme chegam. Analisar polaridade do texto e, se possível, prosódia (tom de voz). Classificar sentimento de -1 (Hostil) a 1 (Positivo). 🛡️ **Gestão de Falhas:** Resultados inconclusivos ou texto muito curto. Se o confidence score for baixo (< 0.6), classificar como "Neutro" por padrão para evitar falsos alertas de "Cliente Irritado".

**39. Detecção de Voicemail (AMD)** 📍 Local: `apps/api/src/voice` 💻 **Diretriz Técnica:** Answering Machine Detection. Analisar o padrão de áudio inicial (silêncio longo, "bip"). Se detectado correio de voz, a IA deve esperar o bip e deixar uma mensagem pré-gravada ou gerada. 🛡️ **Gestão de Falhas:** Falso positivo (confundir humano com máquina). Configurar a IA para, na dúvida, iniciar com uma saudação genérica e segura ("Alô? É o [Nome]?") que funciona para ambos os casos.

**40. Painel de Controle de Voz (Softphone UI)** 📍 Local: `apps/web/src/voice` 💻 **Diretriz Técnica:** Interface flutuante com controles completos: Mute, Keypad (DTMF), Hold, Transfer. Usar Optimistic UI para feedback instantâneo ao clicar nos botões. 🛡️ **Gestão de Falhas:** Delay visual entre clique e ação. Exibir spinners de loading minúsculos dentro dos botões de ação crítica (ex: Hold) até receber confirmação do socket de que o comando foi executado.

## 🟢 CICLO 5: ENGENHARIA DE DADOS & WORKERS

**Foco:** Processamento Assíncrono, Escalabilidade de Background Jobs e Resiliência de Dados.

**41. Setup BullMQ (Message Queue)** 📍 Local: `libs/shared/src/queue` 💻 **Diretriz Técnica:** Implementar arquitetura de filas robusta. Filas separadas por prioridade e tipo: `critical` (emails de senha, pagamentos), `voice` (processamento áudio), `default` (enrichment). Usar pool de conexões Redis dedicado. 🛡️ **Gestão de Falhas:** Redis OOM (Out of Memory). Configurar política de expiração para logs de jobs concluídos/falhos (ex: manter apenas últimas 24h ou 1000 jobs) para não entupir a memória do Redis.

**42. Worker de Enriquecimento de Leads** 📍 Local: `apps/worker/src/enrich` 💻 **Diretriz Técnica:** Job que consulta APIs externas (Apollo, Clearbit, LinkedIn). Implementar lógica de "Merge Inteligente" para atualizar campos vazios do lead sem sobrescrever dados manuais inseridos pelo usuário. 🛡️ **Gestão de Falhas:** Rate Limits de APIs externas. Configurar o BullMQ com `backoff` exponencial automático. Se bater no limite, o job espera 1m, 2m, 4m, etc., antes de tentar novamente.

**43. Scraper Ético e Robusto** 📍 Local: `apps/worker/src/scraper` 💻 **Diretriz Técnica:** Cluster de Puppeteer ou Playwright. Navegar até o site do lead, extrair meta tags, tecnologias usadas (Wappalyzer logic) e contatos. Respeitar `robots.txt` onde aplicável. 🛡️ **Gestão de Falhas:** Bloqueios e Captchas. Se encontrar Captcha ou Cloudflare challenge, abortar imediatamente para não queimar o IP e marcar o lead com flag "Revisão Manual Necessária". Não tentar bypass agressivo.

**44. Importação CSV via Stream** 📍 Local: `apps/api/src/leads` 💻 **Diretriz Técnica:** Utilizar `busboy` para upload multipart e `csv-parser` para ler o arquivo linha a linha (Stream). Validar e inserir em batches (ex: 1000 registros) no banco. 🛡️ **Gestão de Falhas:** Estouro de Memória (Heap Out of Memory). Nunca carregar o arquivo CSV inteiro em variável. O processamento deve ser estritamente via Streams/Iterators para suportar arquivos de gigabytes com uso mínimo de RAM.

**45. Lead Scoring V2 (Algoritmo Híbrido)** 📍 Local: `apps/api/src/score` 💻 **Diretriz Técnica:** Calcular score (0-100) baseado em perfil (Cargo, Tamanho Empresa) e comportamento (Abriu Email, Clicou Link). Fórmula configurável: `Score = (Profile * 0.3) + (Engagement * 0.7)`. 🛡️ **Gestão de Falhas:** Dados sujos geram scores errados. Implementar pipelines de normalização (trim, lowercase, formatação de telefone) antes de alimentar o algoritmo de scoring.

**46. Job Scheduler Distribuído (Cron)** 📍 Local: `apps/worker/src/cron` 💻 **Diretriz Técnica:** Utilizar Repeatable Jobs do BullMQ em vez de `node-cron` local. Isso garante que, se houver 10 réplicas do worker rodando, o job agendado execute apenas uma vez (singleton). 🛡️ **Gestão de Falhas:** Concorrência e Duplicidade. Utilizar Lock distribuído via Redis (Redlock) se a tarefa for crítica e não puder ser executada duas vezes simultaneamente sob nenhuma hipótese.

**47. Validação de E-mail (MX & SMTP)** 📍 Local: `apps/worker/src/verify` 💻 **Diretriz Técnica:** Pipeline de verificação em 3 etapas: 1. Sintaxe (Regex), 2. Domínio (DNS MX Records),

3. Existência (SMTP Handshake `RCPT TO` sem enviar dados). 🛡️ **Gestão de Falhas:** Timeouts de DNS e Greylisting. Cachear resultados de domínios válidos conhecidos (gmail, outlook) por 7 dias para evitar consultas repetitivas e bloqueios.

**48. Deduplicação Inteligente** 📍 Local: `apps/api/src/leads` 💻 **Diretriz Técnica:** Criar índice único no banco `(org_id, email)`. No insert, usar `ON CONFLICT DO NOTHING` ou `DO UPDATE` (Upsert). Detectar duplicação difusa (Fuzzy Matching) em nomes/empresas similares. 🛡️ **Gestão de Falhas:** Merges complexos automáticos podem destruir dados. Em caso de conflito de dados divergentes (ex: telefones diferentes para mesmo email), criar uma "Sugestão de Merge" para o usuário resolver na UI.

**49. Exportação Assíncrona de Dados** 📍 Local: `apps/worker/src/export` 💻 **Diretriz Técnica:** Para exports grandes, o endpoint API apenas enfileira o job. O worker gera o CSV/Excel, faz upload para bucket privado (S3) com expiração de 24h e notifica o usuário via WebSocket/Email com o link de download. 🛡️ **Gestão de Falhas:** Arquivos gigantes (> 100MB). O sistema deve automaticamente zipar o arquivo e, se necessário, dividi-lo em partes menores para evitar timeouts de download no navegador do cliente.

**50. Gestão de Falhas (DLQ - Dead Letter Queue)** 📍 Local: `apps/worker/src/main` 💻 **Diretriz Técnica:** Configurar Dead Letter Queue para todos os workers. Jobs que falham após N tentativas vão para a DLQ. Persistir o erro e stack trace no banco ( `JobErrors` ) para análise post-mortem. 🛡️ **Gestão de Falhas:** Fila de DLQ enchendo silenciosamente. Configurar alerta crítico (PagerDuty/Slack) se a DLQ acumular mais de X itens, indicando um bug sistêmico ou falha em serviço externo.

### 🟢 CICLO 6: OMNICHANNEL INBOX

**Foco:** Centralização da Comunicação, Sincronização e Fluxo de Trabalho Unificado.

**51. Timeline Unificada (SQL View)** 📍 Local: `prisma/views` 💻 **Diretriz Técnica:** Criar uma SQL View ou Query Union performática que agrega tabelas `Calls` , `Emails` , `WhatsAppMessages` , `Notes` e `Tasks` , ordenadas cronologicamente. Isso simplifica drasticamente a query do frontend. 🛡️ **Gestão de Falhas:** Paginação lenta em tabelas grandes. Utilizar paginação baseada em cursor ( `seek pagination` ) usando o timestamp do evento, em vez de `OFFSET/LIMIT` , para manter performance constante.

**52. WhatsApp API Integration** 📍 Local: `apps/api/src/wa` 💻 **Diretriz Técnica:** Webhook Receiver para API Business (Meta) ou gateways terceiros (Z-API/Evolution). Descriptografar payload, normalizar número de telefone (E.164) e associar ao Lead correspondente. 🛡️ **Gestão de Falhas:** Mensagens fora de ordem. O frontend deve reordenar as mensagens baseando-se estritamente no timestamp da mensagem vindo da API, e não na ordem de chegada do webhook.

**53. Editor de Templates com Variáveis** 📍 Local: `apps/web/src/editor` 💻 **Diretriz Técnica:** Implementar editor Rich Text (Lexical ou TipTap). Suportar injeção de variáveis `{{nome}}` , `{{empresa}}` . Validar templates no backend antes de salvar para garantir sintaxe correta. 🛡️ **Gestão de Falhas:** Variável faltando no momento do envio. O sistema deve bloquear o envio e

destacar visualmente na UI quais variáveis não puderam ser resolvidas para aquele lead específico.

**54. Motor de Cadência (Workflow Engine)** 📍 Local: `apps/worker/src/cadence` 💻 **Diretriz Técnica:** Engine de fluxo de trabalho. Lead entra na cadência -> Espera 1 dia -> Envia Email 1 -> Se não abrir -> Espera 2 dias -> Call Task. O estado atual de cada lead na cadência deve ser persistido. 🛡️ **Gestão de Falhas:** Loops infinitos ou estados inconsistentes. Implementar um limite hardcode de passos diários por lead e validações de sanidade antes de processar a próxima etapa.

**55. Tracking de Links e Aberturas** 📍 Local: `apps/api/src/link` 💻 **Diretriz Técnica:** Reescrever links nos emails para domínio de tracking ( `trk.salesos.com/xyz` ). Ao clicar, registrar User Agent, IP, Timestamp, redirecionar com 301. Usar pixel transparente 1x1 para tracking de abertura. 🛡️ **Gestão de Falhas:** Cliques de Bots de Segurança. Filtrar User-Agents conhecidos de scanners de email e IPs de datacenters de segurança para não inflar métricas de clique falsamente.

**56. Detecção de Resposta (Reply Detection)** 📍 Local: `apps/worker/src/inbox` 💻 **Diretriz Técnica:** Monitorar inbox. Ao receber email, fazer parse do `Subject` (remover "Re:") e `References` headers para thread matching. Se for resposta de um lead em cadência, pausar a cadência automaticamente ("Goal Reached"). 🛡️ **Gestão de Falhas:** Respostas automáticas (Out of Office). Analisar headers `Auto-Submitted` e padrões de texto OOO. Se for OOO, *não* pausar a cadência (ou agendar retomada para data futura), configurável pelo usuário.

**57. Sincronização Bidirecional de Agenda** 📍 Local: `apps/api/src/cal` 💻 **Diretriz Técnica:** Integração Google Calendar / Outlook via OAuth2. Usar Webhooks de "push notifications" dos provedores para detectar novos eventos e sincronizar disponibilidade em tempo real. 🛡️ **Gestão de Falhas:** Token Revogado. Se o refresh token falhar, marcar a integração como "Desconectada" e notificar o usuário imediatamente via e-mail e banner na plataforma.

**58. Inbox Real-time (Socket)** 📍 Local: `apps/api/src/gateway` 💻 **Diretriz Técnica:** Quando um novo evento (email/wpp) é processado, emitir evento `msg:new` para a sala Socket.io da organização ( `room:org_id` ). O frontend atualiza a lista e incrementa contadores de não lidos. 🛡️ **Gestão de Falhas:** Desconexão silenciosa do socket. Implementar mecanismo de polling de fallback (ex: a cada 30s) se a conexão socket estiver instável, garantindo que o usuário eventualmente veja novas mensagens.

**59. Sistema de Tags Flexível** 📍 Local: `apps/api/src/tags` 💻 **Diretriz Técnica:** Relação Many-to-Many `LeadTags` . Permitir criação de tags com cores hex customizáveis. Tags devem ser indexadas para filtro rápido no LeadBoard. 🛡️ **Gestão de Falhas:** Poluição de Tags (Tag Sprawl). Implementar limite lógico de tags por organização (ex: 50 ou 100) e UI de merge de tags similares para manter a taxonomia limpa.

**60. Sync de E-mail (IMAP/Gmail API)** 📍 Local: `apps/worker/src/mail` 💻 **Diretriz Técnica:** Para provedores genéricos, usar `IMAP Idle` para escutar novos emails. Para Gmail/Outlook, preferir APIs oficiais. Sincronizar apenas pastas relevantes (Inbox, Sent). 🛡️ **Gestão de Falhas:**

Quota de API e Throttling. Implementar "Sync Incremental" (trazer apenas mensagens após o último `historyId` ou data) e respeitar rigorosamente os limites de rate limit do provedor de email.

### 🟢 CICLO 7: COBRANÇA E LIMITES

**Foco:** Gestão Financeira, Controle de Recursos e Monetização.

**61. Wallet System Transacional (Ledger)** 📍 Local: `apps/api/src/billing` 💻 **Diretriz Técnica:** Implementar sistema de créditos (Wallet) usando padrão de Ledger (Dupla Entrada). Toda movimentação deve ter um registro de débito e um de crédito. A operação de débito deve ser atômica. 🛡️ **Gestão de Falhas:** Race Conditions (gastar saldo duas vezes). Utilizar Row Locking ( `SELECT FOR UPDATE` ) no Postgres durante a transação de débito para garantir consistência.

**62. Custo Diferenciado por Recurso** 📍 Local: `libs/shared/src/config` 💻 **Diretriz Técnica:** Mapa de configuração de custos centralizado: `VOICE_MIN=10 credits` , `EMAIL_VERIFY=1 credit` , `AI_GEN=2 credits` . O sistema deve consultar essa config antes de debitar. 🛡️ **Gestão de Falhas:** Preço Zero ou Negativo. Adicionar validação no startup da aplicação que impede o boot se houver configurações de preço inválidas ou zeradas acidentalmente.

**63. Integração Stripe (Webhooks e Checkout)** 📍 Local: `apps/api/src/pay` 💻 **Diretriz Técnica:** Usar Stripe Checkout para segurança PCI. Processar webhooks `invoice.paid` e `checkout.session.completed` para adicionar créditos ou renovar assinatura. 🛡️ **Gestão de Falhas:** Webhook perdido. Implementar um Job Cron diário de reconciliação que busca eventos recentes na API do Stripe e verifica se foram processados no banco local.

**64. Middleware de Quota e Bloqueio** 📍 Local: `apps/api/src/guards` 💻 **Diretriz Técnica:** Guard global ou decorator `@Quota('voice')` . Verifica se `balance > 0` antes de deixar a requisição chegar no controller. Retornar `402 Payment Required` se saldo insuficiente. 🛡️ **Gestão de Falhas:** Bypass de Guard. A validação de saldo deve ser repetida na camada de Serviço (Double Check), dentro da transação do banco, para garantir que o saldo não acabou durante o processamento.

**65. Soft Limits e Alertas de Consumo** 📍 Local: `apps/worker/src/alert` 💻 **Diretriz Técnica:** Monitorar consumo. Disparar e-mails de alerta quando o consumo atingir 80% e 95% da franquia mensal ou saldo de créditos. 🛡️ **Gestão de Falhas:** Spam de alertas. Implementar flag `alert_sent_at` e garantir que apenas 1 alerta de cada tipo seja enviado a cada ciclo de faturamento ou a cada 24h.

**66. Geração de Faturas PDF** 📍 Local: `apps/api/src/billing` 💻 **Diretriz Técnica:** Gerar PDF detalhado ("Invoice") usando Puppeteer ou bibliotecas de PDF server-side. Incluir discriminativo de uso (minutos voz, emails enviados). 🛡️ **Gestão de Falhas:** Layout quebrado em dados longos. Implementar testes visuais automatizados ou templates HTML resilientes que lidam com quebra de página corretamente.

**67. Planos e Tiers (Feature Flagging)** 📍 Local: `libs/shared/src/const` 💻 **Diretriz Técnica:** Enum `PLAN_LIMITS` definindo capacidades por plano (ex: Basic, Pro, Enterprise). Usar esses limites para habilitar/desabilitar features na UI e na API. 🛡️ **Gestão de Falhas:** Upgrade de Plano. Ao fazer upgrade, limpar imediatamente caches relacionados a limites do usuário (Redis) para que as novas features apareçam sem delay.

**68. Trial Management Automático** 📍 Local: `apps/worker/src/trial` 💻 **Diretriz Técnica:** Job diário que verifica organizações em Trial. Se `createdAt + 7 days < now`, marcar status como `EXPIRED` e bloquear acesso, enviando email de "Trial Ended". 🛡️ **Gestão de Falhas:** Abuso de Trials. Implementar fingerprinting de navegador e bloqueio de e-mails descartáveis/temporários no registro para dificultar a criação serial de contas trial.

**69. Gestão de Assinaturas (Portal)** 📍 Local: `apps/api/src/sub` 💻 **Diretriz Técnica:** Utilizar o Stripe Customer Portal para permitir que o usuário gerencie cartões, faça upgrades/downgrades e cancelamentos, delegando a complexidade de UI para o Stripe. 🛡️ **Gestão de Falhas:** Cancelamento Imediato. Se o usuário cancelar, a assinatura deve permanecer válida até o fim do "Period End". O webhook `customer.subscription.deleted` é que deve remover o acesso final.

**70. Histórico de Transações e Logs** 📍 Local: `apps/api/src/billing` 💻 **Diretriz Técnica:** Expor tabela `Ledger` para o usuário como "Extrato Financeiro". Implementar paginação e filtros por data/tipo. 🛡️ **Gestão de Falhas:** Lentidão em contas antigas. Arquivar transações com mais de 1 ano em tabelas de histórico ou Cold Storage, mantendo a tabela principal leve.

### 🟢 CICLO 8: OBSERVABILIDADE

**Foco:** Monitoramento Proativo, Business Intelligence e Visibilidade.

**71. Dashboard Socket (Live Metrics)** 📍 Local: `apps/api/src/gateway` 💻 **Diretriz Técnica:** Servidor emite evento de broadcast a cada 5-10s com métricas agregadas (Calls Active, Emails Sent Today) para usuários conectados no Dashboard. 🛡️ **Gestão de Falhas:** Carga excessiva no Socket. Se o número de conexões simultâneas passar de 10k, aumentar o intervalo de broadcast dinamicamente ou usar amostragem para reduzir CPU.

**72. Gráfico Funil (Sankey/Bar)** 📍 Local: `apps/web/src/charts` 💻 **Diretriz Técnica:** Utilizar bibliotecas robustas (Recharts/Visx). O backend deve entregar dados já agregados via SQL (`GROUP BY stage`), evitando processamento pesado no front. 🛡️ **Gestão de Falhas:** Dados insuficientes. Exibir "Empty States" educacionais ("Faça sua primeira chamada para ver dados aqui") em vez de gráficos vazios ou quebrados.

**73. Métricas de Voz Detalhadas** 📍 Local: `apps/api/src/stats` 💻 **Diretriz Técnica:** Calcular KPIs: Duração Média de Chamada (ACD), Taxa de Conexão, Custo por Minuto. Armazenar metadados técnicos (Jitter, Packet Loss) para debug. 🛡️ **Gestão de Falhas:** Outliers estatísticos. Excluir chamadas com duração < 3s (falha de conexão ou caixa postal imediata) das médias para não distorcer os relatórios de performance humana.

**74. Logs de Uso IA (Token Audit)** 📍 Local: `apps/api/src/ai` 💻 **Diretriz Técnica:** Logar cada interação com LLM: Prompt Tokens, Completion Tokens, Model Used, Latency. Essencial para análise de custo e otimização. 🛡️ **Gestão de Falhas:** Privacidade (PII) nos logs. Sanitizar o conteúdo do prompt/response antes de salvar no log de auditoria se houver risco de dados sensíveis, mantendo apenas a contagem de tokens.

**75. Monitoramento Infra (Prometheus Stack)** 📍 Local: `infra/k8s` 💻 **Diretriz Técnica:** Deploy de Prometheus para coleta de métricas e Grafana para visualização. Sidecars em pods para expor métricas de aplicação e sistema. 🛡️ **Gestão de Falhas:** Perda de histórico de métricas. Configurar Volume Persistente (PVC) para o Prometheus para garantir que dados de performance sobrevivam a restarts do cluster.

**76. Alertas de Anomalia de Negócio** 📍 Local: `apps/worker/src/mon` 💻 **Diretriz Técnica:** Monitorar taxas de erro de negócio (ex: falha em envio de email > 5%). Se `error_rate > threshold` , disparar alerta no Slack da engenharia. 🛡️ **Gestão de Falhas:** Falso positivo por baixo volume. Usar janelas deslizantes de tempo (ex: média dos últimos 5 min) e volume mínimo de eventos para disparar alerta.

**77. Heatmap de Atividade** 📍 Local: `apps/web/src/charts` 💻 **Diretriz Técnica:** Matriz `[Dia da Semana][Hora do Dia]` . Intensidade da cor baseada no número de interações. Ajuda a identificar melhores horários para ligar. 🛡️ **Gestão de Falhas:** Timezone Confusion. Todos os dados devem ser armazenados em UTC e convertidos para o Timezone do navegador do usuário apenas no momento da renderização.

**78. Leaderboard Gamificado (Redis)** 📍 Local: `apps/api/src/game` 💻 **Diretriz Técnica:** Utilizar Redis Sorted Sets ( `ZADD leader:sales score member` ). Operações de ranking são O(log(N)), extremamente rápidas. 🛡️ **Gestão de Falhas:** Empates no ranking. Definir critério de desempate determinístico (ex: quem atingiu a pontuação primeiro ganha a posição superior).

**79. Relatórios Agendados (Email Digest)** 📍 Local: `apps/worker/src/report` 💻 **Diretriz Técnica:** Renderizar relatório em HTML, converter para PDF/Excel e enviar por e-mail semanalmente. 🛡️ **Gestão de Falhas:** Bloqueio de anexo. Se o arquivo gerado for maior que 10MB, fazer upload para S3 e enviar um link seguro de download com expiração, em vez de anexar no email.

**80. Integração Sentry (Error Tracking)** 📍 Local: `apps/web/src/main` 💻 **Diretriz Técnica:** Inicializar Sentry no Front e Back. Ativar Distributed Tracing para conectar erros do frontend com logs do backend. 🛡️ **Gestão de Falhas:** Estouro de cota do Sentry. Configurar `tracesSampleRate` e `replaysSessionSampleRate` baixos em produção (ex: 10%) e 100% em dev/staging.

### 🟢 CICLO 9: QUALIDADE E TESTES

**Foco:** Prevenção de Regressão, Confiança no Deploy e Cultura de Qualidade.

**81. Testes Unitários Abrangentes** 📍 Local: `*.spec.ts` 💻 **Diretriz Técnica:** Utilizar Vitest pela velocidade. Mocar todas as dependências externas (DB, APIs). Focar em lógica de

negócios pura, utils e services. 🛡️ **Gestão de Falhas:** Snapshots frágeis. Evitar testes de Snapshot em lógica complexa ou UI volátil, pois quebram facilmente e tendem a ser atualizados sem revisão. Preferir asserções explícitas.

**82. Testes de Integração (API)** 📍 Local: `apps/api/test` 💻 **Diretriz Técnica:** Supertest + Docker Compose. Subir um DB Postgres limpo para testes. Testar o fluxo completo HTTP -> Controller -> Service -> DB. 🛡️ **Gestão de Falhas:** Sujeira no Banco. Executar cada teste dentro de uma transação de banco de dados e dar Rollback no `afterEach`, garantindo isolamento total entre testes.

**83. Testes E2E (End-to-End)** 📍 Local: `apps/web/e2e` 💻 **Diretriz Técnica:** Playwright. Cobrir fluxos críticos ("Happy Paths"): Login, Importação de Lead, Início de Chamada. Testar em Chromium, Firefox e WebKit. 🛡️ **Gestão de Falhas:** Flakiness (Testes intermitentes). Configurar auto-retry (max 2 tentativas) no CI para testes que falham por questões de rede ou timing, e investigar a causa raiz de testes instáveis.

**84. CI Pipeline Otimizado** 📍 Local: `.github/workflows` 💻 **Diretriz Técnica:** Pipeline paralelo. Cachear `node_modules` (pnpm store). Usar `Nx Affected` para rodar lint/test/build apenas nos projetos que foram alterados no PR. 🛡️ **Gestão de Falhas:** Builds lentos. Monitorar tempo de CI. Se passar de 10min, investir em máquinas maiores (GitHub Runners) ou otimizar camadas de cache Docker.

**85. Husky Hooks (Pre-Commit)** 📍 Local: `.husky/pre-commit` 💻 **Diretriz Técnica:** Configurar `lint-staged`. Rodar Lint, Prettier e Type Check apenas nos arquivos staged antes de permitir o commit. 🛡️ **Gestão de Falhas:** Bypass de Hooks. O CI é a fonte da verdade. Mesmo com Husky, o pipeline do servidor deve rodar todas as verificações novamente para garantir que nada passou via `git commit --no-verify`.

**86. Storybook para Design System** 📍 Local: `libs/ui/.storybook` 💻 **Diretriz Técnica:** Documentar todos os componentes de UI (Botões, Inputs, Cards) isoladamente. Facilita o desenvolvimento visual sem precisar rodar o app inteiro. 🛡️ **Gestão de Falhas:** Regressão Visual. Integrar com Chromatic ou ferramenta similar para detectar mudanças visuais não intencionais (pixel diff) nos componentes a cada PR.

**87. SonarQube (Análise Estática)** 📍 Local: `sonar-project.properties` 💻 **Diretriz Técnica:** Configurar Quality Gate. O PR não passa se houver novos Code Smells, Bugs Críticos ou Duplicação de Código acima do limite. 🛡️ **Gestão de Falhas:** Falsos Positivos. Configurar arquivo de exclusão rigoroso para arquivos de teste, mocks e configurações geradas automaticamente.

**88. Configuração de Ambientes e Secrets** 📍 Local: `config/envs` 💻 **Diretriz Técnica:** Manter arquivos `.env.example` atualizados. Validar schemas de variáveis de ambiente no CI/CD antes do deploy. 🛡️ **Gestão de Falhas:** Vazamento de Secrets. Adicionar step no CI (gitleaks) que scaneia o código fonte em busca de padrões de chaves privadas ou senhas hardcoded antes do merge.

**89. Banco de Teste Efêmero (RAM Disk)** 📍 Local: `docker-compose.test.yml` 💻 **Diretriz Técnica:** Configurar o container do Postgres de teste para usar `tmpfs` (montar data dir na RAM). Isso acelera drasticamente a execução de testes de integração que fazem muito I/O. 🛡️ **Gestão de Falhas:** Persistência. Irrelevante neste caso, pois o banco é volátil por design. O foco é velocidade de execução do pipeline.

**90. Code Coverage Mínimo** 📍 Local: `vitest.config.ts` 💻 **Diretriz Técnica:** Definir threshold global de 80% de cobertura. O build falha se a cobertura cair. 🛡️ **Gestão de Falhas:** Cobertura viciada. Métricas de cobertura não garantem qualidade. Reforçar no Code Review que testes devem verificar comportamento, não apenas "passar pelas linhas".

### 🟢 CICLO 10: UX/UI E OTIMIZAÇÃO

**Foco:** Performance Percebida, Acessibilidade e Retenção de Usuário.

**91. i18n Setup (Internacionalização)** 📍 Local: `apps/web/src/i18n` 💻 **Diretriz Técnica:** Setup de `react-i18next`. Carregar arquivos de tradução JSON (PT-BR, EN, ES) via Lazy Loading (separar por namespace: `common`, `auth`, `dashboard`) para não inflar o bundle inicial. 🛡️ **Gestão de Falhas:** Chave de tradução faltando. Configurar fallback para o idioma padrão (Inglês) e emitir warning no console em ambiente de desenvolvimento para alertar o dev.

**92. PWA (Progressive Web App)** 📍 Local: `apps/web/vite.config` 💻 **Diretriz Técnica:** Plugin `vite-plugin-pwa`. Gerar Manifest e Service Worker. Permitir instalação no Desktop/Mobile. Configurar estratégia de cache `Stale-While-Revalidate` para assets estáticos. 🛡️ **Gestão de Falhas:** Cache obsoleto (App antigo carregando). Implementar UX de "Nova versão disponível" (Toast) que força a atualização do Service Worker e reload da página quando um novo deploy é detectado.

**93. Acessibilidade (WCAG 2.1)** 📍 Local: `apps/web/src` 💻 **Diretriz Técnica:** Instalar `eslint-plugin-jsx-a11y`. Garantir navegação via teclado (Tab Index), contraste de cores adequado e ARIA Labels em elementos interativos sem texto visível. 🛡️ **Gestão de Falhas:** Teste apenas visual. Realizar auditorias manuais periódicas utilizando leitores de tela (NVDA/VoiceOver) para garantir usabilidade real para deficientes visuais.

**94. Code Splitting e Lazy Loading** 📍 Local: `apps/web/src/routes` 💻 **Diretriz Técnica:** Utilizar `React.lazy()` e `Suspense` em nível de rota. O bundle JS da página "Settings" não deve ser baixado se o usuário só acessa o "Dashboard". 🛡️ **Gestão de Falhas:** Chunk Load Error (Deploy novo deletou chunks antigos). Implementar Error Boundary global que captura erros de carregamento de chunk e oferece um botão "Recarregar Aplicação" para o usuário baixar a nova versão.

**95. Dark Mode Nativo** 📍 Local: `libs/ui/src/theme` 💻 **Diretriz Técnica:** Utilizar classes `dark:` do Tailwind CSS. Persistir preferência no LocalStorage e respeitar `prefers-color-scheme` do sistema operacional como default. 🛡️ **Gestão de Falhas:** Flash of Unstyled Content (Flicker branco ao carregar). Injetar script minúsculo e bloqueante no `<head>` do HTML para ler o storage e aplicar a classe `dark` antes do React hidratar.

**96. Virtualização de Listas Grandes** 📍 Local: `apps/web/src/list` 💻 **Diretriz Técnica:** Utilizar `react-window` ou `tanstack-virtual` para renderizar grids de Leads com milhares de linhas. Renderizar apenas os itens visíveis na viewport + pequeno buffer. 🛡️ **Gestão de Falhas:** Perda de posição do scroll. Implementar lógica para restaurar a posição exata do scroll quando o usuário navega para o detalhe de um lead e volta para a lista.

**97. Otimização de Imagens Next-Gen** 📍 Local: `apps/web/src/assets` 💻 **Diretriz Técnica:** Converter assets para WebP/AVIF no tempo de build. Utilizar atributo `loading="lazy"` para imagens abaixo da dobra. 🛡️ **Gestão de Falhas:** Imagem quebrada (404). Componente de Imagem deve ter manipulador `onError` que substitui automaticamente a fonte quebrada por um Placeholder genérico (Avatar ou Ícone).

**98. Skeleton Screens (Loading States)** 📍 Local: `libs/ui/src/skeleton` 💻 **Diretriz Técnica:** Criar componentes Skeleton que imitam o layout final (largura/altura). Usar animação de pulso sutil. Evitar spinners de tela cheia que bloqueiam a percepção de progresso. 🛡️ **Gestão de Falhas:** Layout Shift (CLS). O Skeleton deve ter dimensões idênticas ao conteúdo carregado para evitar que a página "pule" quando os dados chegarem, o que prejudica o Core Web Vitals.

**99. Responsividade Mobile-First** 📍 Local: `apps/web/src` 💻 **Diretriz Técnica:** Utilizar prefixos Tailwind `md:`, `lg:` extensivamente. Garantir touch targets de no mínimo 44px para botões em mobile. Esconder colunas de tabela menos importantes em telas pequenas. 🛡️ **Gestão de Falhas:** Hover em touch devices. Não depender de estados `:hover` para exibir informações críticas, pois eles não existem (ou funcionam mal) em touchscreens.

**100. Onboarding Guiado (User Tour)** 📍 Local: `apps/web/src/tour` 💻 **Diretriz Técnica:** Implementar `Driver.js` ou similar. Criar um tour interativo passo-a-passo no primeiro login, destacando elementos chave da UI. 🛡️ **Gestão de Falhas:** Usuário preso no tour. O botão "Pular Tutorial" deve ser sempre visível e acessível. O estado "tour completado" deve ser salvo no banco para não importunar o usuário novamente.

### 🟢 CICLO 11: SEGURANÇA AVANÇADA

**Foco:** Conformidade Legal (GDPR/LGPD), Defesa em Profundidade e Auditoria.

**101. Redação de PII (Data Masking)** 📍 Local: `libs/shared/src/log` 💻 **Diretriz Técnica:** Middleware de logger deve interceptar objetos e aplicar Regex para mascarar CPF, Email, Cartão de Crédito ( `***` ) antes de escrever no stdout/arquivo. 🛡️ **Gestão de Falhas:** Vazamento em logs de erro. Garantir que stack traces ou objetos de erro não contenham dumps de dados de usuário não sanitizados.

**102. Criptografia em Repouso (Encryption at Rest)** 📍 Local: `infra/postgres` 💻 **Diretriz Técnica:** Habilitar criptografia de disco (LUKS/dm-crypt) no volume do banco de dados ou usar TDE (Transparent Data Encryption) se disponível na versão Enterprise/Cloud do Postgres. 🛡️ **Gestão de Falhas:** Perda de chaves de criptografia. Backup da Master Key deve ser armazenado em um Cofre (Vault) seguro, off-site e com controle de acesso rigoroso. Sem a chave, os backups são inúteis.

**103. Rotação de Chaves Automatizada** 📍 Local: `scripts/security` 💻 **Diretriz Técnica:** Scripts para rotacionar segredos (JWT Secret, API Keys) periodicamente. O app deve suportar múltiplas chaves (atual e anterior) durante o período de transição para evitar downtime. 🛡️ **Gestão de Falhas:** Downtime durante rotação. A aplicação deve tentar validar com a chave nova; se falhar, tenta a antiga. Após a propagação, a chave antiga é revogada.

**104. WAF (Web Application Firewall)** 📍 Local: `infra/nginx` 💻 **Diretriz Técnica:** Configurar regras no Nginx ou Cloudflare WAF. Rate limiting por IP, bloqueio de padrões SQL Injection, XSS e User-Agents maliciosos conhecidos. 🛡️ **Gestão de Falhas:** Bloqueio de tráfego legítimo. Monitorar logs do WAF em modo "Simulação" antes de ativar o bloqueio (Block Mode) para tunar as regras e criar whitelists necessárias.

**105. Anti-Brute Force e Account Lockout** 📍 Local: `apps/api/src/auth` 💻 **Diretriz Técnica:** Implementar delay exponencial (1s, 2s, 4s...) após falhas de login. Após 5-10 tentativas falhas, bloquear a conta temporariamente (15min). 🛡️ **Gestão de Falhas:** Negação de Serviço no Usuário. Permitir desbloqueio via e-mail (Magic Link) para que o usuário legítimo possa recuperar o acesso caso tenha esquecido a senha, sem esperar o tempo de bloqueio.

**106. Vulnerability Scan Contínuo** 📍 Local: `.github/workflows` 💻 **Diretriz Técnica:** Integrar Trivy ou Snyk no pipeline. Escanear imagens Docker base e dependências npm ( `package.json` ) em busca de CVEs conhecidas a cada build. 🛡️ **Gestão de Falhas:** Build quebrando por vulnerabilidades low-severity. Configurar o scanner para falhar o build apenas em vulnerabilidades `High` ou `Critical` que tenham fix disponível.

**107. Política de Senhas Forte (NIST Guidelines)** 📍 Local: `libs/shared/src/val` 💻 **Diretriz Técnica:** Não exigir troca periódica de senha (obsoleto). Exigir complexidade, comprimento mínimo (12 chars) e verificar contra listas de senhas vazadas (Pwned Passwords) usando `zxcvbn` para medir entropia. 🛡️ **Gestão de Falhas:** UX ruim na criação de senha. Exibir medidor de força visual em tempo real e feedback claro sobre quais requisitos faltam, em vez de erro genérico ao submeter.

**108. Logout Remoto e Revogação** 📍 Local: `apps/api/src/auth` 💻 **Diretriz Técnica:** Endpoint para revogar Refresh Tokens específicos ou todos os tokens de um usuário. Isso permite "Sair de todos os dispositivos" em caso de roubo de conta. 🛡️ **Gestão de Falhas:** Delay na revogação. Como o Access Token (JWT) é stateless, a revogação só tem efeito quando ele expira e o cliente tenta usar o Refresh Token revogado. Manter o tempo de vida do Access Token curto (ex: 5-15 min).

**109. Gestão de Termos de Uso (ToS)** 📍 Local: `apps/api/prisma` 💻 **Diretriz Técnica:** Tabela `UserTerms` rastreando qual versão dos termos o usuário aceitou e quando. Se os termos mudarem (versão incrementada), o login deve ser bloqueado até novo aceite. 🛡️ **Gestão de Falhas:** Bloqueio legal. Forçar um modal de "Li e Aceito" bloqueante no próximo login do usuário após atualização dos termos.

**110. Captcha Invisível (Smart Challenge)** 📍 Local: `apps/web/src/auth` 💻 **Diretriz Técnica:** Utilizar Cloudflare Turnstile ou Google reCAPTCHA v3 (Invisible). Analisa o comportamento do

usuário e pontua o risco sem exigir interação (clicar em hidrantes). 🛡️ **Gestão de Falhas:** Script de Captcha bloqueado por AdBlockers. Implementar fallback para um desafio visual simples hospedado localmente ou permitir login se o IP tiver alta reputação no histórico interno.

### 🟢 CICLO 12: ECOSSISTEMA

**Foco:** Extensibilidade, Integrações de Terceiros e Canais Adicionais.

**111. Extensão Chrome (Sales Helper)** 📍 Local: `apps/chrome-ext` 💻 **Diretriz Técnica:** Desenvolver extensão Manifest V3. Content Script injeta sidebar no LinkedIn/Sales Navigator. Comunica com a API principal via background service worker. 🛡️ **Gestão de Falhas:** Mudança no DOM do LinkedIn. Seletores CSS quebram frequentemente. Usar seletores resilientes ( `data-control-name` , texto âncora) ou Computer Vision básico para identificar elementos na tela.

**112. API Pública para Desenvolvedores** 📍 Local: `apps/api/src/public` 💻 **Diretriz Técnica:** Expor endpoints REST documentados. Autenticação via API Keys ( `sk_live_...` ) com escopos limitados ( `leads:read` , `calls:write` ). Throttling estrito e segregado da API interna. 🛡️ **Gestão de Falhas:** Abuso de API. Monitoramento automático de anomalias. Se uma chave fizer requests erráticos ou excessivos, revogar automaticamente e notificar o admin da org.

**113. Webhooks de Saída (Eventos para Clientes)** 📍 Local: `apps/worker/src/hook` 💻 **Diretriz Técnica:** Permitir que clientes cadastrem URLs para receber eventos ( `lead.created` , `call.finished` ). Worker dispara POST com payload JSON assinado. 🛡️ **Gestão de Falhas:** Endpoint do cliente fora do ar. Implementar política de retry exponencial (até 3 dias). Se falhar consistentemente, desativar o webhook para economizar recursos.

**114. App Mobile (Wrapper Híbrido)** 📍 Local: `apps/mobile` 💻 **Diretriz Técnica:** Utilizar Capacitor.js para empacotar o PWA existente como app nativo iOS/Android. Adicionar plugins nativos para Push Notifications, Contatos e Dialing. 🛡️ **Gestão de Falhas:** Crash em código nativo. Integrar SDK do Sentry Mobile para capturar stack traces da camada nativa (Java/Swift) além do JS.

**115. Plugin Email (Outlook/Gmail)** 📍 Local: `apps/outlook-add` 💻 **Diretriz Técnica:** Desenvolver Add-in usando Office.js e Google Apps Script. Sidebar lateral no cliente de e-mail mostrando dados do CRM sobre o remetente. 🛡️ **Gestão de Falhas:** Autenticação no contexto do plugin. Gerar tokens de sessão específicos para o Add-in, evitando que o usuário precise digitar senha repetidamente dentro do Outlook.

**116. Marketplace de Integrações** 📍 Local: `apps/web/src/market` 💻 **Diretriz Técnica:** Interface Grid listando integrações (HubSpot, Salesforce, Pipedrive). Fluxo OAuth2 padronizado para conectar contas e mapear campos. 🛡️ **Gestão de Falhas:** Tokens de integração expirados. Job de verificação periódica que testa a validade dos Refresh Tokens das integrações ativas e notifica o usuário para reconectar se necessário.

**117. Importador Universal de Dados** 📍 Local: `apps/web/src/import` 💻 **Diretriz Técnica:** UI de mapeamento de colunas ("De-Para"). O usuário sobe CSV, o sistema adivinha o mapeamento ( `Nome -> full_name` ) e permite ajuste manual. Validação de dados no client-side antes do

upload. 🛡️ **Gestão de Falhas:** Erros de tipo de dados. Executar validação "Dry-Run" em uma amostra (primeiras 50 linhas) para alertar sobre erros de formatação (ex: datas inválidas) antes de processar o arquivo todo.

**118. JS SDK (NPM Package)** 📍 Local: `packages/sdk` 💻 **Diretriz Técnica:** Biblioteca TypeScript isomórfica (Node/Browser) distribuída via NPM. Wrapper tipado para a API Pública, facilitando a vida de devs parceiros. 🛡️ **Gestão de Falhas:** Versionamento. Manter compatibilidade reversa estrita. Se precisar quebrar contrato, lançar nova versão major e manter suporte à anterior na API via versionamento de rota.

**119. Embed Form (Captura de Leads)** 📍 Local: `apps/api/src/embed` 💻 **Diretriz Técnica:** Gerar snippet de script `<script src="...">` que renderiza um formulário de captura no site do cliente. Envia dados via POST seguro com proteção CORS e Honeypot. 🛡️ **Gestão de Falhas:** XSS e Spam. Sanitização rigorosa de todos os inputs recebidos via formulário público. Uso de Captcha invisível obrigatório.

**120. SSO Social (Login Facilitado)** 📍 Local: `apps/api/src/auth` 💻 **Diretriz Técnica:** Implementar Passport Strategies para "Login com Google" e "Login com Microsoft". Essencial para reduzir fricção no onboarding. 🛡️ **Gestão de Falhas:** Conta duplicada. Se o usuário fizer login social com um e-mail que já existe (criado via senha), fazer o link automático das contas após verificação de propriedade do e-mail.

### 🟢 CICLO 13: ENTERPRISE FEATURES

**Foco:** Requisitos Corporativos, SLA e Customização Avançada.

**121. SAML/SSO Corporativo (Okta/AD)** 📍 Local: `apps/api/src/sso` 💻 **Diretriz Técnica:** Utilizar Jackson (BoxyHQ) ou Auth0 Enterprise para abstrair a complexidade do protocolo SAML/OIDC. Permitir que empresas conectem seu Active Directory. 🛡️ **Gestão de Falhas:** Expiração de Certificados IdP. Implementar monitoramento da validade dos metadados XML do provedor e alertar administradores da organização 30 dias antes da expiração.

**122. Domínio Personalizado (CNAME)** 📍 Local: `infra/proxy` 💻 **Diretriz Técnica:** Permitir que o cliente acesse via `crm.cliente.com` . Roteamento via CNAME no DNS. Provisionamento automático de certificados SSL via Let's Encrypt e Proxy Reverso (Traefik/Caddy). 🛡️ **Gestão de Falhas:** Falha na validação DNS. Verificar a propagação do registro CNAME antes de tentar emitir o certificado SSL para evitar rate limits da Let's Encrypt.

**123. Whitelabel e Temas Customizados** 📍 Local: `apps/web/src/theme` 💻 **Diretriz Técnica:** Injetar variáveis CSS (CSS Custom Properties) no `:root` baseado na configuração da Organização carregada. Permitir customizar cor primária, logo e favicon. 🛡️ **Gestão de Falhas:** Contraste de Acessibilidade. Calcular automaticamente a cor de contraste (texto preto ou branco) sobre a cor primária escolhida pelo usuário para garantir legibilidade.

**124. Hierarquia de Times e Permissões** 📍 Local: `apps/api/src/org` 💻 **Diretriz Técnica:** Estrutura de árvore (Adjacency List) para times: Regional -> Gerência -> Squad. Permissões de visualização de dados podem ser restritas a "Apenas meu time" ou "Árvore abaixo". 🛡️ **Gestão**

**de Falhas:** Referência Circular. Validar no backend que um time não pode ser pai de si mesmo ou de seus ancestrais ao mover nós na árvore.

**125. Monitor SLA e Status Page** 📍 Local: `apps/web/src/status` 💻 **Diretriz Técnica:** Página de status pública e privada (para Enterprise com SLA contratual). Mostrar uptime histórico e incidentes ativos. 🛡️ **Gestão de Falhas:** Violação de SLA. Se o uptime cair abaixo do contratado (ex: 99.9%), o sistema deve gerar automaticamente um relatório de crédito de serviço para compensação financeira, se aplicável.

**126. IP Whitelist (Firewall de Aplicação)** 📍 Local: `apps/api/src/guard` 💻 **Diretriz Técnica:** Middleware que verifica se o IP do cliente está na lista de CIDRs permitidos para aquela organização. Requisito comum em bancos e seguradoras. 🛡️ **Gestão de Falhas:** Auto-Lockout (Admin se bloqueia). Permitir bypass de IP Whitelist através de um fluxo de recuperação de acesso via token de e-mail ou código de backup físico.

**127. Filas de Processamento Prioritárias** 📍 Local: `libs/shared/src/q` 💻 **Diretriz Técnica:** Configurar BullMQ com prioridades. Jobs de clientes Enterprise recebem prioridade mais alta no consumo dos workers. 🛡️ **Gestão de Falhas:** Starvation (Clientes pequenos nunca processam). Garantir que a fila de baixa prioridade tenha pelo menos uma fatia mínima garantida de processamento (Weight-based scheduling), para não travar totalmente o plano Basic.

**128. Data Export Corporativo (Dump)** 📍 Local: `apps/worker/src/dump` 💻 **Diretriz Técnica:** Ferramenta de exportação completa. Gera um dump SQL ou JSON de todos os dados da organização para fins de backup ou migração. 🛡️ **Gestão de Falhas:** Timeout e Tamanho. Streamar o dump diretamente para armazenamento frio (S3 Glacier) e enviar link assinado por e-mail. Nunca tentar servir o download diretamente da API.

**129. Audit Cold Storage (Long Term)** 📍 Local: `infra/storage` 💻 **Diretriz Técnica:** Mover logs de auditoria antigos (> 90 dias) do Postgres para S3 Glacier ou Parquet files. Mais barato e atende requisitos de compliance de retenção de 5 anos. 🛡️ **Gestão de Falhas:** Recuperação Lenta. Deixar claro na UI que a consulta de logs arquivados é assíncrona e pode levar horas para ser disponibilizada ("Solicitar Recuperação").

**130. Chat Suporte VIP (Intercom Integration)** 📍 Local: `apps/web/src/help` 💻 **Diretriz Técnica:** Widget de chat (Intercom/Zendesk) que só carrega ou aparece para usuários flaggeados como VIP/Enterprise. Roteamento direto para fila de suporte sênior. 🛡️ **Gestão de Falhas:** Agentes Offline. Se não houver suporte VIP online, oferecer formulário de contato com promessa de SLA de resposta (ex: 1 hora) e escalonar via PagerDuty para o time de plantão.

### 🟢 CICLO 14: INTELIGÊNCIA ESTRATÉGICA

**Foco:** Insights de Negócio, Previsibilidade e Diferenciação por Dados.

**131. Benchmark de Mercado Anonimizado** 📍 Local: `apps/worker/src/stat` 💻 **Diretriz Técnica:** Agregar dados de todos os tenants para criar benchmarks ("Sua taxa de abertura de 20% está acima da média do mercado de 15%"). 🛡️ **Gestão de Falhas:** Privacidade de Dados.

Aplicar Differential Privacy e garantir que nenhum benchmark seja gerado se o grupo de amostra for menor que X empresas, para impossibilitar a identificação de concorrentes específicos.

**132. Score Preditivo de Conversão (ML)** 📍 Local: `apps/worker/src/ml` 💻 **Diretriz Técnica:** Treinar modelo de Regressão Logística ou Random Forest simples usando histórico de leads fechados. Prever probabilidade de fechamento para novos leads. 🛡️ **Gestão de Falhas:** Model Drift (Modelo desatualizado). Re-treinar o modelo mensalmente com dados novos. Monitorar a acurácia do modelo e desligar a predição se a performance cair abaixo de um limiar aceitável.

**133. Motor de Recomendações ("Next Best Action")** 📍 Local: `apps/api/src/rec` 💻 **Diretriz Técnica:** Rule Engine ou ML que sugere a próxima ação: "Ligue agora", "Envie email", "Pesquise no LinkedIn". Baseado em tempo desde última interação e score. 🛡️ **Gestão de Falhas:** Fadiga de Decisão (Spam de sugestões). Limitar o número de recomendações diárias por lead (ex: max 1 ação ativa) para manter o foco do vendedor.

**134. Monitoramento de Concorrentes (News API)** 📍 Local: `apps/worker/src/news` 💻 **Diretriz Técnica:** Job que busca notícias sobre empresas alvo e seus concorrentes via Google News API / Bing Search. Usar NLP para filtrar relevância. 🛡️ **Gestão de Falhas:** Ruído (Notícias irrelevantes). Filtragem estrita de keywords e análise de sentimento. Permitir que o usuário refine as keywords de monitoramento.

**135. Playbooks AI Gerativos** 📍 Local: `apps/api/src/gen` 💻 **Diretriz Técnica:** Gerar sequências de cadência inteiras (Templates de Email, Scripts de Call) personalizadas para o nicho do cliente usando LLMs. "Crie uma cadência para vender SaaS para dentistas". 🛡️ **Gestão de Falhas:** Alucinação de Melhores Práticas. O playbook gerado deve passar por uma etapa de revisão humana obrigatória na UI antes de ser ativado e disparar emails.

**136. A/B Testing de Prompts de Venda** 📍 Local: `apps/api/src/ab` 💻 **Diretriz Técnica:** Permitir criar duas variantes de email/script na cadência. Distribuir aleatoriamente 50/50. Medir conversão e declarar vencedor automaticamente. 🛡️ **Gestão de Falhas:** Viés Estatístico. Não declarar vencedor até atingir significância estatística (tamanho de amostra adequado). Alertar se o teste estiver rodando há muito tempo sem resultado conclusivo.

**137. Simulação de Cenários (Monte Carlo)** 📍 Local: `apps/api/src/sim` 💻 **Diretriz Técnica:** Ferramenta "What If": "Se eu aumentar o time em 2 pessoas, quanto vendo a mais?". Simulação baseada nas taxas de conversão históricas do funil. 🛡️ **Gestão de Falhas:** Custo Computacional. Limitar a complexidade da simulação e o horizonte de tempo. Executar em background se a projeção for muito complexa.

**138. Classificação Automática de Objeções** 📍 Local: `apps/worker/src/nlp` 💻 **Diretriz Técnica:** Analisar transcrições de chamadas perdidas. Classificar motivo: "Preço", "Concorrente", "Sem Interesse". Usar Zero-shot Classification (Bart/GPT). 🛡️ **Gestão de Falhas:** Classificação Errada. Incluir categoria "Outros" e permitir que o vendedor corrija a classificação manualmente, retroalimentando o sistema.

**139. Feedback Loop de IA** 📍 Local: `apps/api/src/feed` 💻 **Diretriz Técnica:** Mecanismo de Thumbs Up/Down em toda geração de IA. Salvar o prompt, a resposta e o feedback para dataset de fine-tuning futuro. 🛡️ **Gestão de Falhas:** Vandalismo de Dados. Ignorar feedbacks de usuários com comportamento suspeito ou que dão downvote em massa em tudo.

**140. Kill Switch de IA (Parada de Emergência)** 📍 Local: `libs/shared/src/feat` 💻 **Diretriz Técnica:** Chave global no Redis `AI_STOP`. Se ativada, todos os serviços de IA param imediatamente e o sistema volta para modo manual. Útil em caso de alucinação em massa ou custo descontrolado. 🛡️ **Gestão de Falhas:** Pânico Desnecessário. O Kill Switch deve parar apenas novos jobs de IA, mantendo o restante do SaaS (CRM, Email) funcionando perfeitamente.

### 🟢 CICLO 15: PROCESSOS E ESCALA

**Foco:** Documentação, Governança de Engenharia e Manutenibilidade a Longo Prazo.

**141. Portal de Documentação (Developer Hub)** 📍 Local: `apps/docs` 💻 **Diretriz Técnica:** Docusaurus ou GitBook. Documentar arquitetura, setup, APIs e guias de contribuição. Manter versionado junto com o código. 🛡️ **Gestão de Falhas:** Docs Desatualizados. Implementar "Broken Link Checker" no CI. Exigir atualização de docs no Definition of Done de features que alteram API.

**142. Runbooks de Incidentes (SRE)** 📍 Local: `docs/ops` 💻 **Diretriz Técnica:** Guias passo-a-passo para incidentes comuns: "Redis Down", "API Latency High", "Database Locked". Reduz o MTTR (Mean Time To Recovery). 🛡️ **Gestão de Falhas:** Acesso durante falha. Manter cópia offline (PDF) dos runbooks acessível aos SREs caso o GitHub/Docs esteja inacessível.

**143. Guia de Contribuição e Estilo** 📍 Local: `CONTRIBUTING.md` 💻 **Diretriz Técnica:** Regras claras sobre Conventional Commits, fluxo de PR, padrões de código e setup de ambiente. 🛡️ **Gestão de Falhas:** PRs fora do padrão. Configurar PR Template que obriga o preenchimento de checklist de conformidade antes de solicitar review.

**144. Feature Flags (Decoupling Deploy from Release)** 📍 Local: `libs/shared/src/flag` 💻 **Diretriz Técnica:** Utilizar Unleash ou PostHog. Permitir fazer deploy de código desligado e ativar gradualmente (Canary Release) para % dos usuários. 🛡️ **Gestão de Falhas:** Serviço de Flags offline. O código deve sempre ter valores default hardcoded seguros caso a API de Feature Flags esteja inacessível.

**145. Release Notes Automatizado** 📍 Local: `scripts/release` 💻 **Diretriz Técnica:** Utilizar `semantic-release`. Analisar commits, determinar versão (Major/Minor/Patch), gerar Changelog, criar Tag git e publicar release. 🛡️ **Gestão de Falhas:** Changelog sujo. Configurar filtro para ignorar commits do tipo `chore`, `docs`, `refactor` e `test` nas notas de release públicas, focando em `feat` e `fix`.

**146. Métricas DORA (DevOps Research)** 📍 Local: `apps/worker/src/met` 💻 **Diretriz Técnica:** Coletar e visualizar: Deployment Frequency, Lead Time for Changes, Change Failure Rate, Time to Restore Service. 🛡️ **Gestão de Falhas:** Drift de Metas. Revisão mensal das

métricas. Se a frequência de deploy cair e o lead time subir, é sinal de dívida técnica acumulada ou processo burocrático.

**147. Governança de Backlog** 📍 Local: `docs/process` 💻 **Diretriz Técnica:** Padronização de Issues. Definição de templates para Bug Report e Feature Request. Política de "Stale Issues" para fechar itens inativos. 🛡️ **Gestão de Falhas:** Backlog infinito. Automação que fecha issues sem atividade há 90 dias com mensagem amigável, mantendo o foco no que é relevante hoje.

**148. RFCs Arquiteturais (Request for Comments)** 📍 Local: `docs/rfcs` 💻 **Diretriz Técnica:** Processo de decisão técnica. Antes de grandes mudanças, escrever documento (Contexto, Opções, Decisão) e abrir para debate do time. 🛡️ **Gestão de Falhas:** Decisões unilaterais ruins. Tornar obrigatório o review de RFC por pelo menos um Arquiteto Sênior antes de iniciar a implementação de mudanças estruturais.

**149. Setup Script (One-Command Start)** 📍 Local: `scripts/setup.sh` 💻 **Diretriz Técnica:** Script idempotente que configura tudo: `docker pull`, `db seed`, `npm install`, `env setup`. O objetivo é `npm run setup` e estar pronto para codar. 🛡️ **Gestão de Falhas:** Diferença de Ambiente. Validar versões de Node, Docker e NPM no início do script e falhar com instruções claras se os requisitos não forem atendidos.

**150. Alerta e Controle de Custos Cloud** 📍 Local: `infra/cost` 💻 **Diretriz Técnica:** Configurar AWS Budgets ou equivalentes. Definir Spend Caps no Supabase/Vercel. 🛡️ **Gestão de Falhas:** Estouro de orçamento. Script que desliga automaticamente recursos não críticos (ambientes de dev/staging) se o orçamento mensal projetado for excedido.

### 🚀 BÔNUS: CICLO 16 - A VANTAGEM INJUSTA (MOAT & SCALE)

**Foco:** Efeitos de Rede, FinOps Extremo e Resiliência ao Caos.

**151. Data Consortium (Inteligência Compartilhada)** 📍 Local: `apps/worker/src/consortium` 💻 **Diretriz Técnica:** Criar um sistema de reputação de telefones/emails global. Se 5 clientes diferentes marcam um número como "Fax" ou "Inexistente", ele é flaggeado globalmente (anonimizado). O valor do SaaS cresce com o nº de usuários (Network Effect). 🛡️ **Gestão de Falhas:** Envenenamento de Dados (Data Poisoning). Exigir consenso de múltiplas organizações não relacionadas e confiáveis antes de aplicar uma flag global.

**152. BYOK (Bring Your Own Key) & BYOM** 📍 Local: `apps/api/src/ai/config` 💻 **Diretriz Técnica:** Permitir que clientes Enterprise insiram suas próprias chaves de API (OpenAI, Azure, Twilio). Isso remove o custo do COGS do SaaS e resolve questões de compliance, pois os dados trafegam na conta do cliente. 🛡️ **Gestão de Falhas:** Chave inválida ou revogada. Validar a chave no momento que é salva. Implementar fallback gracioso: se a chave do cliente falhar, usar a do sistema (com alerta) ou bloquear a feature com mensagem explicativa.

**153. Analytics OLAP (ClickHouse/Tinybird)** 📍 Local: `infra/analytics` 💻 **Diretriz Técnica:** Mover a ingestão de eventos de analytics do Postgres para um banco OLAP colunar (ClickHouse). Postgres não escala para agregações em tempo real de milhões de linhas. 🛡️ **Gestão de Falhas:** Latência de Sincronização. Utilizar ingestão via streaming (Kafka/Redpanda)

para garantir consistência eventual de segundos entre o evento ocorrer e aparecer no dashboard.

**154. Infraestrutura Spot (FinOps)** 📍 Local: `infra/k8s/nodepools` 💻 **Diretriz Técnica:** Configurar Node Pools específicos para rodar `apps/worker` (processamento stateless de IA/Dados) usando Spot Instances (AWS/GCP). Isso reduz o custo de computação em até 90%. 🛡️ **Gestão de Falhas:** Preempção (Desligamento súbito). O K8s lida com novos pods, mas os workers devem lidar com sinais `SIGTERM` graciosamente, devolvendo o job para a fila sem corromper dados.

**155. Chaos Engineering (Teste de Destruição)** 📍 Local: `apps/worker/src/chaos` 💻 **Diretriz Técnica:** Em ambiente de Staging, utilizar ferramentas como Chaos Mesh para simular falhas reais periodicamente: derrubar Redis, aumentar latência de rede, matar pods aleatórios. Garantir que o sistema se recupere sozinho. 🛡️ **Gestão de Falhas:** Vazamento para Produção. Isolar estritamente o Chaos Mesh via Namespaces e RBAC do Kubernetes. Jamais permitir que ele tenha permissões no cluster de produção.

## 🗺️ PARTE 2: VISÃO ESTRATÉGICA E OPTIMIZAÇÕES TÉCNICAS

**Status:** PRÉ-REQUISITOS PARA ESCALABILIDADE GLOBAL

### 🏗️ 1. Infraestrutura & Escalabilidade (The Foundation)

- **Banco de Dados (Sharding/Partitioning):** O design do `schema.prisma` deve prever nativamente o particionamento por `tenantId`. Em cenários multi-tenant massivos, isso evita o problema do "Vizinho Barulhento" (um cliente grande deixando o banco lento para todos) e prepara o terreno para Sharding físico futuro.
- **Cache Layer Multinível (Redis):**
  - **L1 (In-Memory):** Caches de curtíssima duração para configs críticas.
  - **L2 (Redis):** Cachear respostas de endpoints públicos e configurações de tenant.
  - **Auth Cache:** Cachear sessões de usuário para validação de Auth ultrarrápida, permitindo revogação imediata sem bater no DB a cada request.
- **Edge Computing e CDN:**
  - Mover o serving do `apps/web` para a Edge (Vercel/Cloudflare Pages) para latência mínima global.
  - Mover middlewares de segurança (IP Whitelist, Geo-Blocking) para Cloudflare Workers, bloqueando ameaças antes que toquem a infraestrutura cloud principal.

### 🧠 2. Backend & IA (The Brain)

- **Arquitetura Orientada a Eventos (EDA):** Migrar de chamadas de serviço síncronas para eventos assíncronos. Ex: `sales.closed` -> emite evento -> `analytics`, `email`, `billing` escutam independentemente. Isso garante desacoplamento total e escalabilidade independente de serviços.

- **Vector Database (RAG) como Memória de Longo Prazo:** Inserção obrigatória de Qdrant ou Pinecone. O `prompt.service.ts` não deve apenas receber inputs, mas consultar ativamente o banco vetorial para recuperar contexto histórico e preferências do lead antes de gerar qualquer resposta.
- **Resiliência de Voz:** Implementação de "Circuit Breaker" e "Fallback Providers" no `google-speech.service.ts`. Se o serviço primário de voz falhar, o sistema deve comutar transparentemente para OpenAI Whisper ou Azure sem derrubar a chamada telefônica.

### 🎨 3. Frontend & UX (The Experience)

- **Real-time Optimistic UI:** No LeadBoard (Kanban), a interface deve mentir para o usuário: mover o card atualiza a UI instantaneamente. A request vai em background. Se der erro, a UI reverte. Isso cria a sensação de "Zero Latency" essencial para ferramentas de alta produtividade.
- **Modo Offline (Local-First):** Utilizar Service Workers e IndexedDB ( `RxDB` ou `TanStack Query Persist` ) para permitir que vendedores consultem e editem leads mesmo em zonas de sombra ou aviões. Sincronização ocorre quando a conexão voltar.
- **Observabilidade de Cliente (RUM):** Monitoramento real de Core Web Vitals no browser do usuário. Precisamos saber se o SaaS está lento para um cliente em Tóquio antes que ele abra um ticket de suporte.

### 🛡️ 4. Segurança & Compliance (The Trust)

- **Auditoria Imutável (WORM):** Envio de logs críticos do `audit.interceptor.ts` para buckets de armazenamento com trava de retenção (Write Once, Read Many), garantindo