GENSPARK: ARQUITETURA TÉCNICA COMPLETA - DOCUMENTO SUPREMO

1. VISÃO GERAL DO SISTEMA

1.1 Introdução e Contexto

Propósito do sistema: O Genspark é uma plataforma de geração de conteúdo avançada impulsionada por IA, projetada para criar documentos, apresentações, pesquisas profundas e mídia visual com alta fidelidade.

Escala e requisitos: O sistema foi arquitetado para suportar milhões de requisições diárias com um SLO de latência p95 menor que 3 segundos para gerações interativas. A infraestrutura deve garantir alta disponibilidade (99.95%) e consistência eventual para dados distribuídos.

Princípios fundamentais:

- **Modularidade:** Desacoplamento estrito entre serviços de IA, lógica de negócios e persistência.
- **Escalabilidade:** Capacidade de escalar horizontalmente pods de computação e inferência de forma independente.
- **Resiliência:** Design "fail-safe" com retries inteligentes, circuit breakers e degradação graciosa.

1.2 Arquitetura de Alto Nível

O sistema segue uma arquitetura de microserviços orientada a eventos, dividida em quatro camadas principais:

1. **Presentation Layer:** Aplicações client-side (Web/Mobile) que se comunicam via REST e WebSockets.
2. **API Gateway Layer:** Ponto único de entrada responsável por roteamento, rate limiting, autenticação e terminação SSL.
3. **Application Services Layer:** Conjunto de serviços de domínio (Auth, Content, Search, etc.) que executam a lógica de negócios e orquestram workflows.
4. **Data Layer:** Persistência poliglota incluindo SQL (PostgreSQL), NoSQL (Redis), Vector Stores (Qdrant) e Object Storage (S3).

Padrões de comunicação: A comunicação interna prioriza gRPC para chamadas síncronas de baixa latência entre serviços críticos e Apache Kafka para mensageria assíncrona e consistência eventual.

1.3 Stack Tecnológico Completo

Camada	Tecnologias Principais
Frontend	React 18, Next.js 14, TypeScript, Zustand, TanStack Query, WebSockets
Backend	Python 3.11+, FastAPI, Go (serviços críticos), Node.js (real-time)
AI/ML	vLLM, TensorRT, PyTorch, Transformers, LangChain
Databases	PostgreSQL 15, Redis/Valkey, Qdrant/Weaviate (vector DB)
Infraestrutura	Kubernetes, Docker, Terraform, AWS/GCP
Observability	OpenTelemetry, Prometheus, Grafana, Jaeger, ELK Stack
Message Queue	Apache Kafka, RabbitMQ
Orchestration	Temporal.io

1.4 Decisões Arquiteturais Críticas

- **Microserviços vs Monolito:** Optou-se por microserviços para permitir o escalonamento independente de módulos intensivos em GPU (AI Inference) versus módulos I/O bound (Web API).
- **Escolha de Linguagens:** Python para o ecossistema de IA/ML; Go para serviços de alta performance e baixa latência (Storage/Proxy); Node.js para manuseio eficiente de conexões WebSocket.
- **Consistência vs Disponibilidade:** Em adesão ao teorema CAP, prioriza-se Disponibilidade (AP) para fluxos de geração de conteúdo e Consistência (CP) para transações financeiras e autenticação.
- **Event Sourcing:** Utilizado para auditoria completa de ações do usuário e reconstrução de estado.
- **CQRS:** Separação clara entre modelos de leitura (otimizados para queries rápidas) e escrita (focados em integridade).

2. ARQUITETURA DE BACKEND

2.1 Microserviços e Domínios

Auth Service

- **Responsabilidades:** Autenticação centralizada, gestão de ciclo de vida de tokens (JWT), e autorização (RBAC).
- **Tech stack:** FastAPI, OAuth2, PostgreSQL.
- **Dados:** Tabelas de usuários, sessões ativas e matriz de permissões.

Content Generation Service

- **Responsabilidades:** Orquestrador principal dos pipelines de geração de IA. Gerencia a fila de requisições e distribui para workers.
- **Tech stack:** Python, Temporal.io.
- **Recursos:** Integração com LLMs (GPT, Claude), gestão de templates de prompts dinâmicos.

Storage Service

- **Responsabilidades:** Abstração sobre o armazenamento de objetos (S3/MinIO), upload/download seguro e geração de URLs assinadas.
- **Tech stack:** Go.
- **Features:** Integração com CDN, indexação de metadados e garbage collection de arquivos temporários.

Search Service

- **Responsabilidades:** Busca semântica (vetorial) e lexical (full-text) sobre o conteúdo gerado.
- **Tech stack:** Python, Qdrant, Elasticsearch.
- **Pipeline:** Geração de embeddings em tempo real e busca híbrida com re-ranking.

Real-time Service

- **Responsabilidades:** Gestão de conexões WebSocket persistentes para colaboração e atualizações de UI.
- **Tech stack:** Node.js, Socket.io, Redis Pub/Sub.
- **Features:** Salas (Rooms) para documentos, tracking de presença de usuários.

Analytics Service

- **Responsabilidades:** Ingestão de eventos de telemetria e análise de uso do produto.
- **Tech stack:** Python, ClickHouse, Kafka.
- **Design:** Pipelines de agregação de alta velocidade para dashboards em tempo real.

2.2 API Gateway

Utiliza-se Kong ou Envoy como gateway de borda. Funcionalidades críticas incluem:

- **Rate Limiting:** Estratégias de Token Bucket e Sliding Window para prevenir abusos.
- **Routing:** Descoberta dinâmica de serviços e roteamento baseado em path.
- **Segurança:** Terminação TLS, validação de JWT na borda, headers de segurança (CORS, HSTS).
- **Resiliência:** Circuit breakers globais e políticas de retry configuráveis.

2.3 Sistema de Autenticação e Autorização

O fluxo baseia-se em OAuth2 com OIDC. Tokens de acesso (JWT) têm vida curta (15 min) e são renovados via Refresh Tokens (7 dias). O controle de acesso (RBAC) define papéis como Admin, Editor e Viewer, mapeados para permissões granulares no nível do recurso.

2.4 Orchestration Layer (Temporal.io)

O Temporal.io gerencia workflows de longa duração garantindo que o estado não seja perdido em caso de falhas.

Content Generation Workflow:

```
StartWorkflow → ValidateInput → RouteToSpecialist →
[ParallelTasks: GenerateText, FetchImages, CreateAudio] →
ComposeOutput → SaveToStorage → NotifyUser → End
```

Multi-Step Agent Workflow:

```
AgentStart → AnalyzeIntent → PlanSteps →
Loop[ExecuteStep → ValidateOutput → UpdateContext] →
FinalComposition → PersistResult → End
```

2.5 Message Queues e Event Streaming (Kafka)

Kafka atua como a espinha dorsal assíncrona. Tópicos principais incluem `user.events`, `content.generation.requests` e `analytics.events`. O Schema Registry garante a evolução segura dos contratos de dados (Avro/Protobuf).

2.6 Caching Strategies (Redis/Valkey)

Implementação de cache em múltiplas camadas:

- **L1 (Memory):** Cache local LRU na aplicação para dados estáticos.
- **L2 (Redis):** Cache distribuído para dados "quentes" e sessões. Design de chave: `{service}:{resource}:{id}:{version}`.
- **L3 (CDN):** Cache de borda para assets estáticos e mídia pública.

2.7 Database Design (PostgreSQL)

O esquema principal é normalizado para integridade. Tabelas chave:

- **users:** Armazena credenciais e perfis.
- **contents:** Metadados de conteúdo, links para storage e embeddings. Particionada por data/tipo.
- **generation_history:** Logs detalhados de gerações para auditoria e análise de custos.

Estratégia de Sharding: Sharding horizontal baseado em `user_id` para distribuir a carga de escrita, gerenciado via tabela de roteamento no Redis.

3. SISTEMA DE IA E ML

3.1 Model Serving Infrastructure

A infraestrutura de inferência utiliza clusters de GPUs A100 gerenciados via Kubernetes. O **vLLM** é utilizado para servir LLMs com otimizações de *continuous batching* e *PagedAttention*. O **TensorRT** acelera modelos menores. Um registro de modelos permite versionamento semântico e testes A/B (Canary deployments).

3.2 Prompt Engineering System

Sistema modular onde o prompt final é composto dinamicamente:

```
[Core Principles]
+ [Task Router - detecta tipo de tarefa]
+ [Domain Module - carrega especialista]
+ [Few-Shot Examples - 2-3 exemplos relevantes]
+ [Output Format Spec - JSON schema esperado]
+ [User Input - query processado]
= Final Prompt (~4K tokens)
```

Técnicas como *Chain-of-thought* e *ReAct* são aplicadas automaticamente dependendo da complexidade da tarefa.

3.3 Context Management

Gerenciamento de contexto híbrido. O contexto efêmero (sessão ativa) vive no Redis com TTL curto. O contexto persistente (histórico longo) é armazenado no Postgres e recuperado via busca vetorial. Um algoritmo de "Context Pruning" comprime conversações antigas mantendo apenas entidades e decisões críticas.

3.4 Multi-Modal Pipelines

Pipelines especializados para diferentes tipos de mídia:

- **Texto:** Prompting complexo, verificação de fatos, formatação Markdown.
- **Imagem:** Geração paralela (DALL-E, Stable Diffusion), filtro de segurança (NSFW), seleção via CLIP score.
- **Vídeo/Áudio:** Orquestração de APIs externas (Veo, ElevenLabs), sincronização e transcodificação (FFmpeg).

3.5 Agent Orchestration System

Baseado no padrão ReAct. Um "Router Agent" analisa a intenção e delega para "Specialist Agents" (Document, Slide, Code). Ferramentas (Tools) são chamadas de forma segura em ambientes sandbox.

3.6 Tool Calling Framework

Permite que os agentes executem ações reais: Web Search (Serper), execução de código (sandbox), cálculos e operações de arquivo. Cada ferramenta possui esquema estrito de entrada/saída e timeouts definidos.

3.7 Fine-tuning e Continuous Learning

Pipeline automatizado que coleta feedback do usuário e dados de correções. Utiliza LoRA (Low-Rank Adaptation) para fine-tuning eficiente de modelos base (Llama 3, Mixtral) em clusters dedicados, seguido de validação rigorosa antes do deploy.

4. ENGENHARIA DE DADOS

4.1 Data Pipeline Architecture

Arquitetura Lambda combinando processamento em lote (Batch) e tempo real (Speed). A camada Batch (Airflow + Spark) processa dados históricos para o Data Warehouse. A camada Speed (Kafka Streams) alimenta dashboards em tempo real.

4.2 ETL/ELT Processes

Extração via CDC (Debezium) dos bancos de produção. Transformação utilizando **dbt** para criar tabelas "Gold" prontas para análise no Data Warehouse.

4.3 Feature Stores

Garante consistência entre treino e inferência. Features "Online" (Redis) para baixa latência e "Offline" (S3/Parquet) para treinamento de modelos.

4.4 Embedding Generation e Vector Databases

Todo conteúdo gerado passa por um pipeline de embedding (CLIP/SentenceTransformers) e é indexado no **Qdrant**. Isso habilita busca semântica, recomendação e recuperação de contexto (RAG).

4.5 Data Warehouse Design

Modelagem Star Schema. Tabelas fato (ex: `fact_content_generations`) contêm métricas, e tabelas dimensão (ex: `dim_users`, `dim_models`) contêm atributos descritivos.

4.6 Real-time Analytics

Uso de **ClickHouse** para análises OLAP de altíssima velocidade sobre streams de eventos, permitindo consultas de agregações em milissegundos.

4.7 Data Governance e Compliance

Classificação rigorosa de dados. PII é criptografado ou mascarado. Processos automatizados garantem conformidade com GDPR (Direito ao esquecimento, exportação de dados).

5. ARQUITETURA DE FRONTEND

5.1 React/Next.js Architecture

Built with Next.js 14 (App Router). Estratégia de renderização híbrida: páginas de marketing via SSG, dashboard via CSR, e páginas de SEO via SSR. Otimização de bundle via Code Splitting e Lazy Loading de componentes pesados.

5.2 State Management (Zustand)

Lojas globais separadas por domínio (Auth, Content, Editor, UI). Persistência seletiva no LocalStorage para manter sessão e preferências do usuário.

5.3 Real-time Communication

Client Socket.io gerencia a conexão WebSocket. Implementa "Optimistic UI" para feedback instantâneo e lida com reconexões e sincronização de estado após períodos offline.

5.4 Component Library

Baseada em **shadcn/ui** e Radix UI. Design Tokens (CSS Variables) para cores e espaçamento, suportando temas (Dark/Light mode).

5.5 Performance Optimization

Uso intensivo de TanStack Query para cache de servidor no cliente. Pré-carregamento (prefetching) de rotas e dados no hover. Otimização automática de imagens via componente Next.js Image.

5.6 Offline-First Capabilities

Service Workers (Workbox) para cache de assets e API GET requests. IndexedDB (Dexie) armazena rascunhos e fila de sincronização para operações de escrita quando offline.

6. INFRAESTRUTURA E DEVOPS

6.1 Kubernetes Clusters

Multi-cluster setup (Prod, Staging). Node pools segregados: General Purpose para APIs, GPU Pool para inferência (com A100s), e Memory-Optimized para Redis/Vector DB.

6.2 CI/CD Pipelines

GitHub Actions orquestra build, teste e deploy. Estratégias de deploy incluem Blue-Green para serviços críticos e Canary para novos modelos de ML. Rollbacks automatizados baseados em métricas de erro.

6.3 Infrastructure as Code (Terraform)

Toda a infraestrutura é definida em código (Terraform), modularizada por serviço e ambiente. Estado remoto seguro no GCS com locking.

6.4 Monitoring e Observability

Stack baseada em OpenTelemetry. Tracing distribuído (Jaeger) para acompanhar requisições através dos microserviços. Métricas (Prometheus) e Dashboards (Grafana) para monitoramento de saúde e negócios.

6.5 Logging Aggregation

Logs estruturados (JSON) coletados via Fluent Bit e enviados para Elasticsearch (ELK Stack). Políticas de retenção por tier (Hot/Warm/Cold).

6.6 Disaster Recovery

Backups automáticos do PostgreSQL (Full + WAL archiving) e S3 (Versioning + Replication). Plano de DR testado trimestralmente com RTO de 1h e RPO de 15min.

6.7 Auto-scaling Strategies

HPA (Horizontal Pod Autoscaler) para escalar serviços baseados em CPU/Memória. KEDA para escalar workers baseado na profundidade das filas do Kafka/Redis. Cluster Autoscaler para provisionar nós físicos.

7. FLUXOS CRÍTICOS DETALHADOS

7.1 User Request → Response

Fluxo síncrono/assíncrono híbrido. O request passa pelo Gateway e Auth. O Backend inicia um workflow no Temporal e retorna um ID. O Frontend se conecta via WebSocket para receber atualizações de progresso em tempo real (Server-Sent Events ou WS push) enquanto o conteúdo é gerado e salvo no S3.

7.2 Content Generation Pipeline

```
Input → Analysis → Routing → Generation → Post-processing → Storage → Output
```

O pipeline analisa a intenção, roteia para um agente especialista, executa pesquisas se necessário, gera o conteúdo (paralelizando slides/seções), valida qualidade, compõe o JSON final, salva e notifica.

7.3 Multi-Step Agent Workflow

Para "Deep Research", um agente Planejador decompõe a query em sub-perguntas. Agentes executores buscam respostas em paralelo. Um agente Sintetizador consolida tudo em um relatório final estruturado.

7.4 Real-time Collaboration Flow

Utiliza Operational Transformation (OT) ou CRDTs. As edições são enviadas como operações (insert/delete) via WebSocket. O servidor resolve conflitos e faz broadcast para outros usuários na mesma sala.

7.5 Search e Retrieval Flow

Query do usuário → Embedding da query. Busca vetorial (Qdrant) + Busca Keyword (Elasticsearch). Fusão de resultados (Reciprocal Rank Fusion) → Re-ranking (Cross-Encoder) → Resposta final personalizada.

7.6 File Processing Pipeline

Upload S3 → Trigger de evento. Worker baixa arquivo, extrai texto (OCR se necessário), limpa, divide em chunks, gera embeddings e indexa no Vector DB. Notifica conclusão.

8. SEGURANÇA E COMPLIANCE

8.1 Security Layers

Defesa em profundidade: Proteção DDoS (Cloudflare), WAF, mTLS entre serviços, validação estrita de input, e isolamento de rede (VPC).

8.2 Data Encryption

Criptografia em repouso (AES-256) em bancos e buckets. Criptografia em trânsito (TLS 1.3) obrigatória. Gestão de chaves via AWS KMS com rotação automática.

8.3 Rate Limiting e DDoS Protection

Limites configurados no Gateway (Kong) e proteção volumétrica na borda (Cloudflare). Circuit Breakers protegem serviços internos de sobrecarga.

8.4 Content Moderation

Filtros de entrada (proibição de termos) e saída (classificadores de toxicidade/NSFW). Conteúdo flaggado vai para fila de revisão humana.

8.5 Privacy e GDPR

Mecanismos automatizados para exportação e exclusão de dados de usuários. Gestão granular de consentimento e logs de auditoria de acesso a dados sensíveis.

8.6 Audit Logging

Logs imutáveis de todas as ações administrativas e de acesso a dados, armazenados em armazenamento "WORM" (Write Once Read Many) para compliance.

9. PERFORMANCE E ESCALABILIDADE

9.1 Load Balancing Strategies

Balanceamento L7 via Envoy. Algoritmos variados: Least Request para inferência, Least Connections para WebSockets, Consistent Hashing para cache.

9.2 Database Sharding

Sharding manual de tabelas críticas por User ID. Particionamento temporal de tabelas de logs/histórico.

9.3 CDN Architecture

Cloudflare como CDN principal. Cache agressivo de assets estáticos e cache controlado (stale-while-revalidate) para conteúdo público gerado.

9.4 Caching Layers

Padrões Cache-Aside para leitura e Write-Behind para eventos de analytics. Proteção contra "Cache Stampede" usando expiração probabilística.

9.5 Query Optimization

Uso extensivo de índices (B-Tree, GIN para JSONB). Análise contínua de planos de execução. Queries lentas são refatoradas ou movidas para Read Replicas.

9.6 Resource Pooling

Pools de conexões (PgBouncer para Postgres, pools internos para Redis) para evitar overhead de handshake e limitar concorrência.

10. CASOS DE USO ESPECÍFICOS

10.1 Document Generation Flow

Geração de relatórios longos: Pesquisa inicial → Geração de esboço → Geração de seções em paralelo → Montagem final e formatação PDF.

10.2 Slide Creation Pipeline

Tópico → Pesquisa → Estrutura de slides (títulos/bullets) → Seleção de imagens para cada slide → Aplicação de tema visual → Exportação PPTX.

10.3 Deep Research System

Investigação iterativa: O sistema formula perguntas, busca, lê resultados, refina perguntas e repete até atingir profundidade/confiança suficiente antes de sintetizar.

10.4 Image/Video Generation Workflow

Prompt → Expansão do Prompt (LLM) → Geração (Modelo de Imagem/Vídeo) → Post-processing (Upscaling, Codec conversion) → Delivery.

Versão: 1.0 | **Data:** 2026-02-10 | **Autores:** Equipe de Engenharia Genspark | **Classificação:** Interno / Confidencial