FACULDADE SENAI FATESG
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Angelo Augusto Leal Ferreira De Souza
Kalleb
Guilherme Xavier

Keeply: Solução de Backup Simplificada

Goiânia, 2025

Sumário

1- INTRODUÇÃO

O presente trabalho aborda o desenvolvimento do **Sistema Keeply**, uma solução de backup distribuído composta por dois principais componentes:

1. o **Keeply Backup Agent**, responsável por varrer diretórios, deduplicar e empacotar dados em contêineres comprimidos, e
2. o **Keeply Backup System (Frontend Web)**, responsável por oferecer uma camada de apresentação moderna para monitorar, restaurar e gerenciar backups em um painel web.

O estudo delimita-se à análise da **arquitetura integrada** desses componentes em ambientes corporativos de pequeno e médio porte e em uso pessoal avançado, em cenários onde é necessário equilibrar custo de armazenamento, janela de recuperação, integridade dos dados e experiência de uso.

O problema central investigado consiste em **reduzir a redundância na transferência e armazenamento de dados**, proporcionando **restauração rápida e auditável**, ao mesmo tempo em que se oferece **visibilidade em tempo quase real** para usuários e equipes de TI, mesmo quando estes não possuem acesso direto à infraestrutura de backend.

O objetivo geral é **projetar, especificar e documentar uma arquitetura coerente de sistema distribuído**, na qual o agente de backup, o backend (banco de dados e storage) e o frontend web cooperam para entregar um ciclo completo de backup e restauração.

Como objetivos específicos, destacam-se:

- (i) estabelecer uma **arquitetura modular** no agente que permita deduplicação por chunks, reuso de hashes e empacotamento eficiente em contêineres comprimidos;
- (ii) definir a **camada de persistência** de metadados (backups, versões, chunks, contêineres, dispositivos) em banco relacional auditável;
- (iii) projetar uma **interface web responsiva** em Next.js integrada a BaaS (como Supabase) para autenticação, listagem, métricas e restauração via

URLs assinadas;

- (iv) documentar requisitos funcionais e não funcionais do sistema como um todo, incluindo ambiente de produção e aspectos de observabilidade;
- (v) descrever o desenvolvimento do sistema proposto, enfatizando as principais telas e interações entre agente e painel web.

A justificativa apoia-se na crescente criticidade da proteção de dados em endpoints distribuídos, na adoção de serviços em nuvem e em modelos BaaS e SaaS, bem como na necessidade de **centralizar a experiência do usuário final** em um painel seguro, ao mesmo tempo em que se mantém um **agente local robusto e eficiente**. A atualidade decorre da expansão de backups em nuvem, da demanda por observabilidade e das discussões sobre LGPD e conformidade. A originalidade surge da combinação de:

- agente com **deduplicação por chunks, compressão ZSTD e cabeçalho próprio (KBC)**,
- backend relacional e storage compatível com S3,
- e frontend Next.js com **Supabase Auth, Storage e Realtime**, formando um **ecossistema integrado de backup**.

2- ESCOPO DO PROJETO:

O sistema Keeply é composto por dois blocos principais de solução que atuam de forma integrada:

1. **Keeply Backup Agent (Backend Local)**

    - Aplicação em Java 21 responsável por varrer diretórios, calcular hashes, gerar planos de backup FULL e INCREMENTAL, deduplicar dados por chunks, empacotar em contêineres comprimidos (.tar.zst) com cabeçalho KBC e enviar esses contêineres para um storage (filesystem local ou bucket compatível com S3).
    - Persiste metadados de backups, versões de arquivos, chunks e contêineres em banco relacional (PostgreSQL/Supabase/SQLite), permitindo rastreabilidade de versões e restauração granular.

2. **Keeply Backup System (Frontend Web)**

    - Aplicação web construída em Next.js (App Router) com React e Tailwind CSS.
    - Integração com Supabase para autenticação (Auth), banco de dados (Postgres gerenciado), storage de objetos e eventos Realtime.
    - Fornece landing pages, fluxos de login/registro/recuperação, dashboard autenticado para acompanhar backups, jobs, métricas e restauração por URLs assinadas.

2.1 Situação atual

- O **agente** encontra-se em estágio de **protótipo funcional**, com módulos de varredura, empacotamento, deduplicação, persistência relacional e integração com storage local ou S3 já descritos e parcialmente implementados.
- O **frontend** contempla **landing pages**, autenticação, dashboard com listagem de backups e jobs, integração com Supabase Realtime para refletir eventos, e APIs internas para geração de URLs assinadas e remoção de backups. Ainda

existem limitações como **criação de jobs via UI/API** e **persistência completa de metadados de upload**.

2.2 Objetivo geral

- Disponibilizar um **sistema distribuído de backup e restauração** com deduplicação global, compressão eficiente e painel web de observabilidade e controle.

2.3 Objetivos específicos

- **Agente**
  - Varredura configurável de diretórios com planos FULL/INCREMENTAL.
  - Deduplicação por chunks com reuso de hashes entre execuções.
  - Empacotamento em contêiner .tar.zst com cabeçalho KBC, offsets e manifestos de chunk.
  - Persistência de metadados (backups, versões, chunks, contêineres, histórico de jobs).
  - Upload/download por filesystem ou S3, com leitura por faixa e retomada.
  - Restauração completa ou parcial, validando hashes e atributos básicos.
  - Interface CLI e opcionalmente JavaFX, com logs estruturados e métricas básicas.
  - Scheduler para agendamento e bloqueio de concorrência.
- **Frontend**
  - Autenticação de usuários, com fluxo de login, registro e recuperação de senha.
  - Dashboard para listagem de backups, jobs e dispositivos associados.
  - Geração de URLs assinadas para download seguro de arquivos.
  - Remoção de backups com verificação de propriedade.
  - Assinatura de eventos Realtime para atualização instantânea da UI.
  - Exibição de métricas agregadas (quantidade de backups, tamanho total, uploads diários).
  - Landing page com seções de marketing, pricing, FAQ e formulários de

autenticação.

2.4 Tecnologias utilizadas (visão geral)

- **Backend/Agente**: Java 21, Maven, Spring Boot, bibliotecas de compressão (ZSTD), HTTP client, driver JDBC para banco relacional, SDK S3 (por exemplo, AWS SDK v2) e JavaFX para UI opcional.
- **Frontend**: Next.js, React, Tailwind CSS, TypeScript, Supabase JS SDK, APIs internas em rotas /api.
- **Infraestrutura/BaaS**: Supabase (Auth, Postgres, Storage, Realtime) e/ou banco relacional dedicado, storage compatível com S3, ambiente de deploy (plataforma de cloud ou container).

3- TAP

Título do projeto: Sistema Keeply de Backup Distribuído.

Gerente responsável: Responsável de Tecnologia / Líder Técnico designado pelo patrocinador.

Partes interessadas:

- Administrador de TI/DevOps (configura e monitora o agente e o painel);
- Operador de suporte (realiza restaurações e acompanha falhas);
- Usuários finais (instalam o agente nos dispositivos e acompanham seus backups pelo painel);
- Time de segurança e conformidade (valida políticas de acesso, criptografia e logs);
- Patrocinador/direção (aprova orçamento, prioriza roadmap e monitora indicadores de sucesso).

Necessidades do negócio:

- Reduzir custo de armazenamento de backups por meio de deduplicação eficiente;
- Diminuir o tempo de recuperação em incidentes de perda de dados;
- Aumentar visibilidade do ciclo de vida de backups em endpoints distribuídos;
- Facilitar a operação por equipes não técnicas por meio de um painel web claro;
- Aderir a boas práticas de integridade e auditabilidade, alinhando-se a exigências de proteção de dados.

Descrição do produto:

- Agente local de backup com deduplicação por chunks, compressão ZSTD e empacotamento em contêineres KBC, integrando-se a storage local ou S3.
- Backend/BaaS com banco relacional e storage de objetos para armazenar metadados e arquivos.
- Aplicação web para autenticação, visualização de backups, jobs e dispositivos, geração de URLs assinadas, remoção de dados e visualização de métricas.

Riscos iniciais:

- Indisponibilidade de storage ou rede durante operações de backup/restauração;
- Corrupção de contêineres ou perda de metadados;
- Vazamento de credenciais (S3, Service Role do Supabase, chaves de API);
- Falhas de performance em ambientes com recursos limitados;
- Implementação incompleta de RLS (Row-Level Security) e políticas de acesso no BaaS.

Premissas:

- Disponibilidade de JDK 21+ nas máquinas que executarão o agente;
- Disponibilidade de banco relacional (PostgreSQL/Supabase ou SQLite local) acessível ao agente;
- Provisionamento de bucket compatível com S3 ou storage equivalente;
- Infraestrutura Supabase (ou equivalente) provisionada para o frontend;
- Acesso HTTPS configurado entre frontend e BaaS;
- Permissões adequadas de leitura/escrita nos diretórios protegidos.

Restrições:

- Escopo atual sem multi-tenant completo e RBAC avançado;
- Políticas complexas de retenção automática fora do escopo inicial;
- Dependência de serviços de terceiros (Supabase, provedor de cloud) e seus SLAs.

Critérios de aceitação:

- Execuções de backup com deduplicação comprovada (redução de bytes armazenados);
- Contêineres gerados com hashes válidos e offsets consistentes para leitura aleatória;
- Restauração completa e parcial funcionando com verificação de integridade;
- Usuário autenticado consegue listar backups e jobs, gerar URLs assinadas e remover seus próprios backups;
- Eventos Realtime refletem novas execuções de backup sem recarregar a

página;

- Logs

4- EAP

1.0 Projeto Sistema Keeply

1.1 Planejamento

- 1.1.1 Levantamento e consolidação de requisitos (RF e RNF).
- 1.1.2 Definição da arquitetura: agente + backend relacional + storage + frontend.
- 1.1.3 Elaboração do TAP e análise de riscos.
- 1.1.4 Definição de cronograma e marcos de entrega.

1.2 Desenvolvimento do Agente

- 1.2.1 Implementação do módulo de scanner e geração de planos FULL/INCREMENTAL.
- 1.2.2 Implementação do packager com deduplicação, compressão ZSTD e cabeçalho KBC.
- 1.2.3 Implementação da camada de persistência de metadados (backups, versões, chunks, contêineres).
- 1.2.4 Implementação de provedores de storage (filesystem, S3 ou equivalente).
- 1.2.5 Implementação dos serviços de restauração (completa e parcial).
- 1.2.6 Implementação da interface CLI e, opcionalmente, UI JavaFX.
- 1.2.7 Implementação de scheduler e mecanismos de bloqueio de concorrência.

1.3 Desenvolvimento do Frontend Web

- 1.3.1 Implementação de landing pages, fluxos de login, registro e recuperação.
- 1.3.2 Implementação de AuthContext/hooks de sessão com Supabase Auth.
- 1.3.3 Implementação de APIs internas (/api/backups, /api/jobs, /api/metrics, /api/download).

- 1.3.4 Implementação de componentes de UI (cards, tabelas, gráficos, modais).
- 1.3.5 Integração com Supabase Realtime para atualização de jobs em tempo quase real.

1.4 Testes e Qualidade

- 1.4.1 Testes unitários no agente (scanner, packager, storage).
- 1.4.2 Testes integrados de upload/download e restauração de arquivos.
- 1.4.3 Testes de carga e simulação de falhas de rede.
- 1.4.4 Testes manuais e de integração no frontend (login, listagem, download, remoção).
- 1.4.5 Validação da integração Realtime e observabilidade (logs/métricas).

1.5 Implantação e Operação

- 1.5.1 Empacotamento do agente (fat-JAR, scripts de inicialização).
- 1.5.2 Build e deploy do frontend (ambientes dev, homologação e produção).
- 1.5.3 Configuração de variáveis de ambiente, credenciais e secrets.
- 1.5.4 Configuração de observabilidade (health checks, métricas, logs).
- 1.5.5 Criação de guia de instalação, operação e suporte.

1.6 Entregáveis

- 1.6.1 Código-fonte do agente e do frontend.
- 1.6.2 Documentação técnica (arquitetura, APIs, modelo de dados).
- 1.6.3 Relatório de testes e métricas de throughput/dedupe.
- 1.6.4 Material de treinamento e handover para operação.

5- ESPECIFICAÇÃO DE REQUISITOS

5.1 Requisitos Funcionais – Agente

- **RF-A01**: Varredura de diretórios configuráveis, com planos FULL/INCREMENTAL baseados em hashes e timestamps.
- **RF-A02**: Deduplicação de arquivos por chunks, com registro de manifestos de chunk reutilizáveis entre execuções.
- **RF-A03**: Empacotamento dos dados em contêiner .tar.zst com cabeçalho KBC, incluindo offsets e hashes cumulativos.
- **RF-A04**: Persistência de metadados de backups, versões de arquivos, chunks e contêineres em banco relacional.
- **RF-A05**: Upload e download de contêineres para storage local ou compatível com S3, com suporte a leitura por faixa e retomada.
- **RF-A06**: Restauração completa ou parcial, com validação de integridade (hashes) e restauração de atributos básicos de arquivos.
- **RF-A07**: Interface de controle via CLI (e opcionalmente JavaFX), permitindo iniciar, pausar, cancelar jobs, listar snapshots e visualizar logs.
- **RF-A08**: Registro de histórico de execuções de backup, incluindo horários, status, volume de dados e dedupe obtida.
- **RF-A09**: Configuração do agente via arquivo .env ou similar, com suporte a múltiplos ambientes (dev/qa/prod).

5.2 Requisitos Funcionais – Frontend

- **RF-F01**: Autenticar usuários via Supabase Auth (login, registro, recuperação de senha).
- **RF-F02**: Listar backups do usuário autenticado com paginação e filtros por dispositivo e status.
- **RF-F03**: Gerar URLs assinadas temporárias para download seguro de arquivos.
- **RF-F04**: Permitir remoção de backups pertencentes ao usuário, com validação de propriedade.
- **RF-F05**: Assinar eventos Realtime de jobs de backup para atualizar a UI sem recarregar a página.

- **RF-F06**: Exibir histórico de jobs, incluindo status, timestamps, tamanho processado e mensagens de erro.
- **RF-F07**: Mostrar métricas agregadas (quantidade de backups, tamanho total, uploads por dia, usuários/dispositivos) em cards e gráficos.
- **RF-F08**: Disponibilizar endpoints internos de health check e métricas para monitoramento externo.
- **RF-F09**: Oferecer landing page com seções de marketing, pricing, FAQ e formulários de autenticação integrados ao fluxo do sistema.

5.3 Requisitos Não Funcionais – Sistema como um Todo

- **RNF01 – Disponibilidade:**
  - O sistema deve tolerar falhas temporárias de rede e storage por meio de mecanismos de retry e feedback claro ao usuário.
- **RNF02 – Desempenho:**
  - O agente deve alcançar throughput compatível com ambientes locais (por exemplo, ≥ 80 MB/s em cenários favoráveis), permitindo ajustes de tamanho de chunk e nível de compressão.
  - O frontend deve carregar o dashboard principal em tempo aceitável em conexões de banda larga comuns (em torno de poucos segundos).
- **RNF03 – Segurança:**
  - Uso de hashes fortes para verificação de integridade.
  - Uso de TLS no tráfego entre agentes, frontend e BaaS/storage.
  - Armazenamento de credenciais em variáveis de ambiente ou mecanismos seguros, não sendo expostas em código-fonte ou cliente web.
  - Aplicação de RLS e regras de ownership para que usuários só acessem seus próprios dados.
- **RNF04 – Escalabilidade:**
  - Arquitetura do agente preparada para múltiplos repositórios por máquina.
  - Frontend stateless, apto a ser escalado horizontalmente e integrado a CDNs
- **RNF05 – Observabilidade:**
  - Logs estruturados e identificáveis por backup/job/contêiner.

- Métricas disponíveis (quantidade de arquivos, chunks, bytes processados, erros por período).
- Endpoints de health check e métricas no frontend/backend para consumo por ferramentas de monitoramento.

- **RNF06 – Confiabilidade:**
  - Geração de contêineres de forma atômica, com validação de integridade antes da marcação como “completo”.
  - Transações de metadados garantindo consistência entre backups, versões e contêineres.
- **RNF07 – Usabilidade:**
  - CLI com comandos curtos, ajuda contextual e feedback textual de progresso.
  - Frontend com interface intuitiva, mensagens claras, navegação consistente e design responsivo.
- **RNF08 – Acessibilidade e Compatibilidade:**
  - Frontend alinhado a práticas de acessibilidade (foco visível, contraste adequado, textos alternativos).
  - Compatibilidade com principais navegadores modernos e dispositivos (desktop e mobile).

6 – DIAGRAMAS DE CLASSE E DOMÍNIO DE DADOS

Esta seção apresenta os principais elementos de modelagem do sistema Keeply, organizados em três perspectivas complementares: **Domínio do Agente**, **Domínio do Frontend** e **Domínio do Banco de Dados**. Em conjunto, esses modelos descrevem como o agente de backup, o painel web e a camada de persistência cooperam para entregar o ciclo completo de backup, deduplicação, armazenamento e restauração.

6.1 Domínio do Agente

O domínio do agente representa os componentes responsáveis por varrer diretórios, deduplicar dados, empacotar arquivos em contêineres comprimidos e coordenar o ciclo de backup e restauração no dispositivo local.

A classe **BackupPlan** é responsável por representar um plano de backup, que pode ser do tipo *FULL* ou *INCREMENTAL*. Ela agrega uma coleção de objetos **FileMetadata** e armazena informações sobre o tipo de plano, horário de criação e repositório alvo. É a partir desse plano que o agente decide quais arquivos serão processados em cada execução.

A classe **FileMetadata** modela os metadados de um arquivo, incluindo caminho, tamanho, timestamps de criação/modificação e hash de conteúdo. Quando um arquivo já foi submetido a deduplicação, o objeto FileMetadata associa-se a um **ChunkManifest**, permitindo o reuso de chunks entre múltiplas execuções de backup sem necessidade de retransmitir o arquivo inteiro.

A classe **Chunk** representa o bloco de dados deduplicado em si, contendo atributos como hash, tamanho e posição no contêiner. Esses blocos são utilizados para montar ou remontar arquivos a partir dos dados efetivamente armazenados.

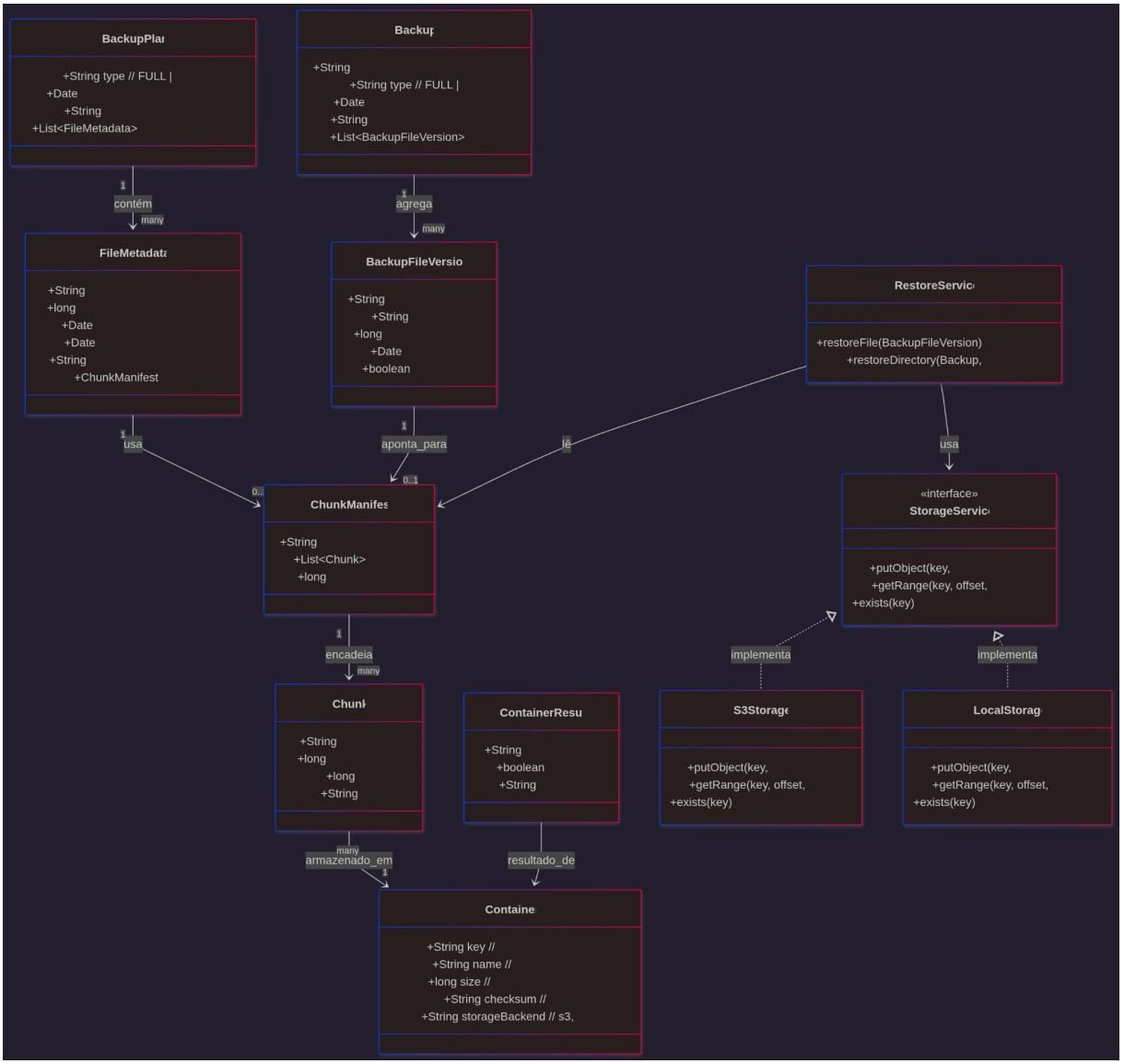
A classe **ChunkManifest** lista a sequência de chunks que compõe uma determinada versão de arquivo. Por meio dessa lista ordenada, o sistema consegue reconstruir o conteúdo completo de um arquivo a partir dos contêineres físicos, sem depender de cópias redundantes.

A classe **Container** representa o contêiner físico gerado pelo agente (por exemplo, um arquivo .tar.zst), incluindo cabeçalho próprio (KBC), lista de entradas, offsets e metadados como tamanho e checksum. Em alguns fluxos é utilizada uma variante conceitual **ContainerResult**, responsável por encapsular o resultado da operação de empacotamento ou upload do contêiner (sucesso, mensagem de erro, chave de armazenamento, entre outros).

As classes **Backup** e **BackupFileVersion** modelam a relação entre os snapshots lógicos e as versões de arquivo. A classe Backup agrega múltiplas BackupFileVersion e está associada a um repositório e a um conjunto de contêineres. Já BackupFileVersion relaciona o arquivo lógico com seus manifestos de chunks, permitindo rastrear quais blocos físicos são necessários para restaurar uma versão específica.

No que diz respeito à integração com o armazenamento, o sistema define o serviço **StorageService** como uma interface genérica, com operações como putObject, getRange e exists. A partir dela, são implementadas especializações como **S3Storage** (para storage compatível com S3) e **LocalStorage** (para filesystem local), permitindo alternar ou combinar provedores de armazenamento sem acoplar o agente a um único backend.

Por fim, o serviço **RestoreService** utiliza os manifestos de chunks (**ChunkManifest**) e a abstração de armazenamento (**StorageService**) para reconstruir arquivos durante o processo de restauração, validando integridade por meio de hashes e restaurando o conteúdo para o diretório de destino definido pelo usuário ou operador.


BackupPlan
+String type // FULL |
+Date
+String
+List<FileMetadata>
Backup
+String
+String type // FULL |
+Date
+String
+List<BackupFileVersion>
1
contém
many
1
agrega
many
FileMetadata
+String
+long
+Date
+Date
+String
+ChunkManifest
BackupFileVersio
+String
+String
+long
+Date
+boolean
RestoreServic
+restoreFile(BackupFileVersion)
+restoreDirectory(Backup,
1
usa
1
aponta_para
lê
usa
0..1
ChunkManifes
+String
+List<Chunk>
+long
«interface»
StorageServic
+putObject(key,
+getRange(key, offset,
+exists(key)
1
encadeia
many
implementa
implementa
Chunk
+String
+long
+long
+String
ContainerResu
+String
+boolean
+String
S3Storage
+putObject(key,
+getRange(key, offset,
+exists(key)
LocalStorag
+putObject(key,
+getRange(key, offset,
+exists(key)
many
armazenado_em
1
resultado_de
Containe
+String key //
+String name //
+long size //
+String checksum //
+String storageBackend // s3,

6.2 Domínio do Frontend

O domínio do frontend modela as entidades conceituais utilizadas pelo painel web para apresentar o estado do sistema ao usuário, sem expor diretamente a complexidade da camada de deduplicação e contêineres.

A classe conceitual **BackupJob** representa uma execução de backup do ponto de vista do painel. Ela inclui atributos como id, userId, deviceId, status, bytesTotal, bytesProcessados, startedAt, finishedAt e errorMessage. Esses dados permitem ao usuário acompanhar o progresso de execuções em tempo quase real, identificar falhas e entender o volume de dados processado por job.

A classe **BackupFile** modela um arquivo disponível para o usuário no painel, com atributos como id, userId, filePath, fileSize, uploadedAt e signedUrl. Ela representa a visão de "arquivo restaurável", normalmente construída a partir dos metadados de snapshots e arquivos armazenados no banco, mas exposta de forma simplificada para o usuário final, incluindo a URL assinada utilizada para download seguro.

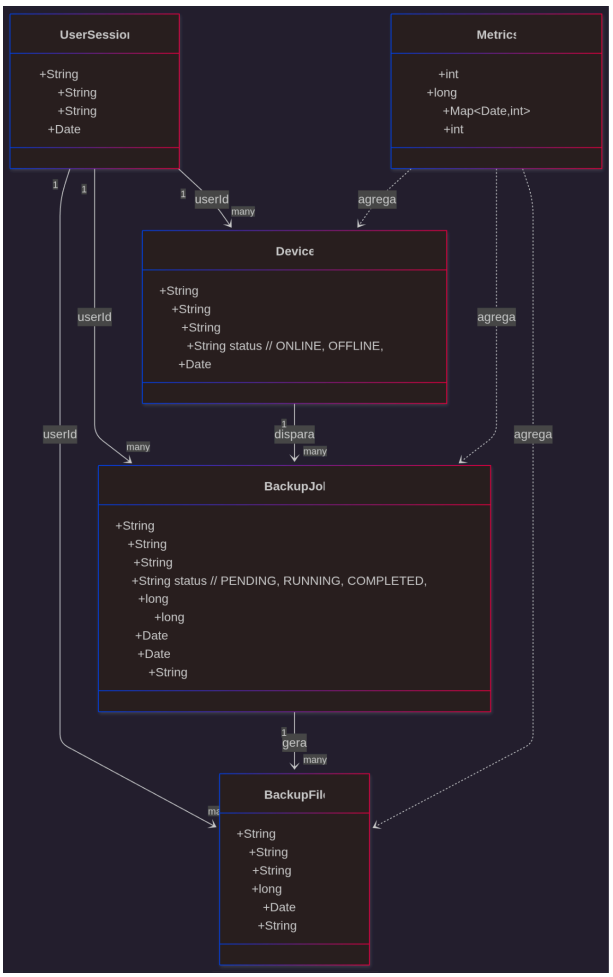
A classe **Device** representa um dispositivo ou instância do agente, com atributos como id, userId, hostname, lastSeen e status. Essa abstração permite ao painel agrupar informações de backup por máquina, indicando se o agente está ativo, inativo ou com status desconhecido, além de mostrar quando foi o último contato com o sistema.

A classe **UserSession** encapsula a sessão autenticada no frontend, incluindo referência ao usuário (user ou userId), accessToken, refreshToken e expiresAt. Ela serve como base para restringir o acesso às informações e para assinar operações que dependem de tokens válidos, como chamadas a APIs internas e inscrição em canais Realtime.

A classe **Metrics** agrega indicadores de uso e capacidade, como totalBackups, totalSize, uploadsByDay e uniqueDevices. Esses dados são utilizados para compor cards e gráficos no dashboard, oferecendo uma visão sintética da saúde do ambiente de backup.

Os principais relacionamentos nesse domínio podem ser descritos da seguinte forma: **UserSession** relaciona-se com **BackupJob**, **BackupFile** e **Device** por meio

do userId, uma vez que a sessão autenticada determina o escopo de dados visíveis no painel. Cada **Device** pode estar associado a múltiplos **BackupJob**, refletindo as execuções realizadas naquele equipamento. Um **BackupJob**, por sua vez, pode originar múltiplos **BackupFile**, já que uma única execução de backup pode processar vários arquivos.



6.3 Domínio do Banco de Dados

O domínio de banco de dados do Sistema Keeply foi projetado para sustentar, de forma integrada, tanto as operações do agente de backup quanto as funcionalidades do painel web. A modelagem segue o paradigma relacional, utilizando PostgreSQL (via Supabase) como tecnologia principal, o que possibilita o uso de chaves primárias, chaves estrangeiras, relacionamentos normalizados e políticas de segurança em nível de linha (RLS). Esses recursos são adequados a sistemas que exigem rastreabilidade, isolamento entre usuários e controle fino de acesso aos dados.

Do ponto de vista conceitual, o modelo organiza-se em alguns grupos de entidades principais:

Domínio de Identidade e Dispositivos

Esse núcleo é composto pelos usuários autenticados (providos por auth.users) e pelas entidades que representam agentes e dispositivos. A tabela agents associa cada instância do agente de backup a um usuário, a um identificador lógico de dispositivo e a atributos como hostname, sistema operacional, arquitetura e datas de primeiro e último contato. Esse domínio também contempla elementos de segurança operacional, como agent_api_keys (chaves de API vinculadas a agentes) e agent_heartbeats (batimentos periódicos que registram status e conectividade do agente), além de agent_tasks, que armazena comandos e tarefas enviadas pelo backend para execução no agente, como iniciar um backup, executar uma restauração ou aplicar uma reconfiguração.

Domínio de Execuções de Backup (Jobs e Snapshots)

A tabela backup_jobs representa cada execução de backup disparada por um agente para um determinado usuário e dispositivo, registrando tipo de job (FULL ou INCREMENTAL), caminho raiz protegido, status (pendente, em execução, concluído, falho ou cancelado), quantidade de arquivos processados, volume de dados, quantidade de chunks novos e reutilizados, além de timestamps de início e término.

A partir de cada job são gerados **snapshots**, modelados pela tabela snapshots, que representam o estado dos arquivos protegidos em um dado momento. Cada snapshot referencia o job de origem, o usuário proprietário e, opcionalmente, um snapshot pai, permitindo encadear backups incrementais a um backup completo.

Essa entidade armazena ainda metadados do contêiner físico (chave, nome, tamanho, checksum, backend de storage) e agregados como total de arquivos e bytes contidos.

Domínio de Arquivos e Chunks Deduplicados

O detalhamento dos arquivos incluídos em cada snapshot é realizado pela tabela snapshot_files, que registra, para cada snapshot, o caminho lógico do arquivo, seu tamanho, hash, data de modificação e um indicador se foi marcado como excluído naquela versão. Essa camada representa a visão lógica dos arquivos sob proteção.

A deduplicação em nível de chunk é modelada por snapshot_file_chunks, que relaciona cada arquivo a uma sequência ordenada de hashes de chunk, incluindo informações de deslocamento e tamanho dentro do contêiner. Dessa forma, é possível reconstruir um arquivo a partir de seus chunks, sem necessidade de armazenar o mesmo bloco de dados repetidas vezes em múltiplos snapshots. Essa estrutura conecta diretamente o domínio lógico (arquivos e versões) ao domínio físico de armazenamento em contêineres compactados.

Domínio de Dispositivos Lógicos e Visões para o Frontend

Embora o agente trabalhe diretamente com agents, o frontend pode expor uma visão mais amigável de dispositivos por meio de uma entidade lógica devices, que agrupa informações de hostname, sistema operacional, status e último contato associados ao usuário. Sobre esse conjunto de tabelas operacionais são construídas consultas e visões derivadas (views) utilizadas pelo painel, como uma visão de arquivos de backup (unindo snapshots e snapshot_files para exibir a lista de arquivos restauráveis por usuário e dispositivo) e uma visão de métricas agregadas, que consolida quantidade de backups, tamanho total armazenado, número de dispositivos únicos e distribuição de uploads por dia.

Essa organização do domínio de banco de dados permite:

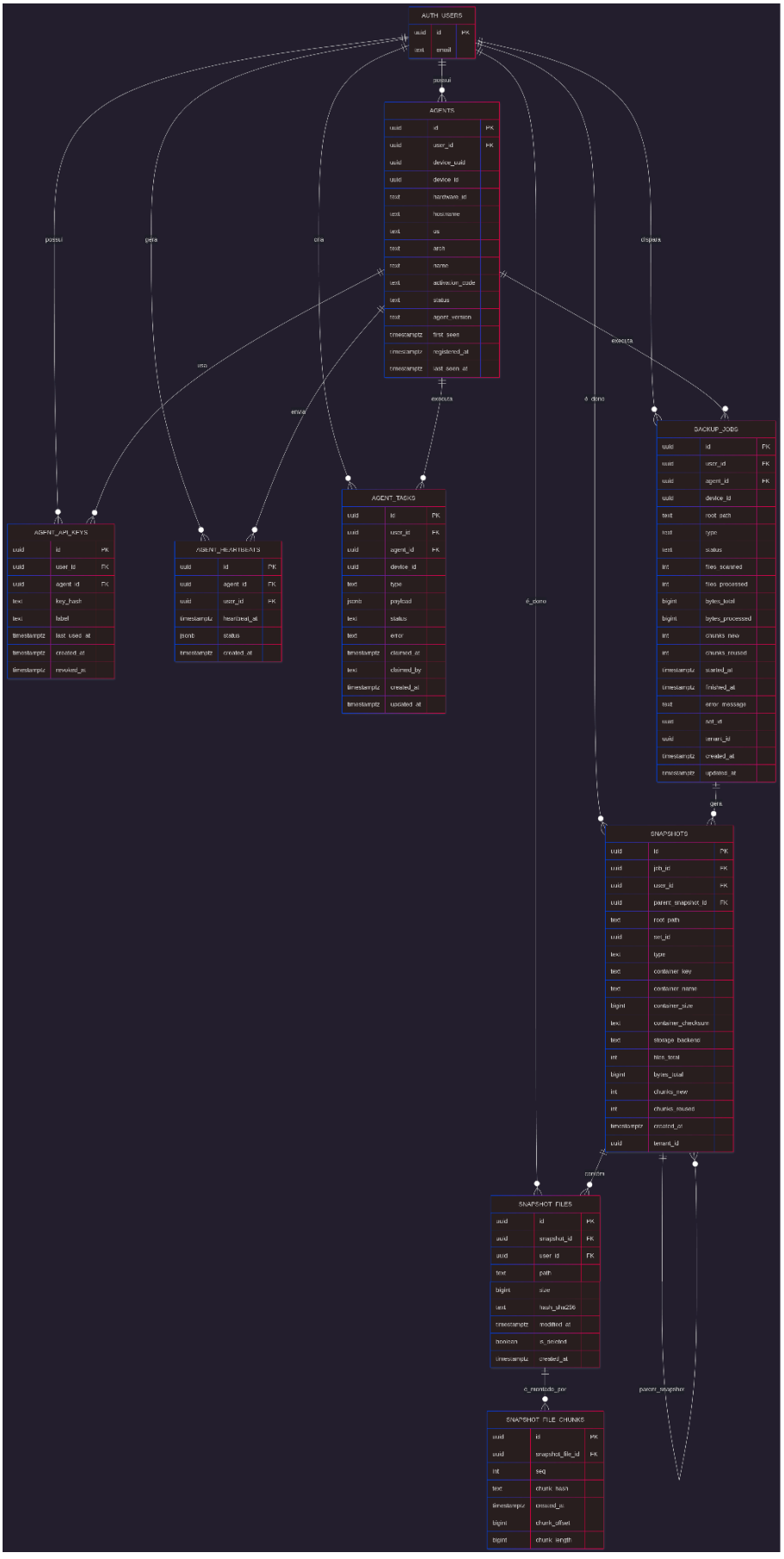
- **Rastreabilidade completa**: de um usuário para seus dispositivos, destes para jobs de backup, destes para snapshots e, por fim, para arquivos e chunks físicos;

- **Suporte à deduplicação global**: reaproveitando chunks entre múltiplas execuções, sem perda de visibilidade lógica;

- **Integração natural com o frontend**: o painel web apenas consulta e filtra dados já estruturados pelo agente e gravados no banco, sem necessidade de acoplamento direto ao filesystem ou ao storage S3;

- **Aplicação de políticas de segurança**: com uso de chaves estrangeiras por usuário, é possível aplicar RLS e garantir que cada conta visualize apenas seus próprios jobs, snapshots, arquivos e métricas.

Dessa forma, o domínio de banco de dados do Keeply atua como ponto de convergência entre o agente local e a interface web, garantindo consistência das informações, possibilidade de auditoria e base sólida para futuras extensões, como multi-tenant completo, papéis diferenciados (RBAC) e relatórios mais avançados de uso e desempenho.


AUTH_USERS
id
email
possui
AGENTS
user_id
device_uuid
device_id
hardware_id
hostname
os
arch
name
activation_code
status
agent_version
first_seen
registered_at
last_seen_at
possui
pela
cria
dispara
executa
usa
envia
executa
é dono
é dono
AGENT_API_KEYS
user_id
agent_id
key_hash
label
last_used_at
created_at
revoked_at
AGENT_HEARTBEATS
agent_id
user_id
heartbeat_at
status
created_at
AGENT_TASKS
user_id
agent_id
device_id
type
payload
status
error
claimed_at
claimed_by
created_at
updated_at
BACKUP_JOBS
user_id
agent_id
device_id
root_path
type
status
files_scanned
files_processed
bytes_total
bytes_processed
chunks_new
chunks_reused
started_at
finished_at
error_message
ext_id
tenant_id
gera
SNAPSHOTS
job_id
user_id
parent_snapshot_id
root_path
os_id
type
container_key
container_name
container_size
container_checksum
storage_backend
file_total
bytes_total
chunks_new
chunks_reused
created_at
tenant_id
contém
SNAPSHOT_FILES
snapshot_id
user_id
path
size
hash_sha256
modified_at
is_deleted
created_at
é_montado_por
parent_snapshot
SNAPSHOT_FILE_CHUNKS
snapshot_file_id
seq
chunk_hash
created_at
chunk_offset
chunk_length

7- AMBIENTE DE PRODUÇÃO

7.1 Cliente (Máquina com Agente)

- Sistema operacional: Linux ou Windows suportado pelo JDK 21+.
- Requisitos mínimos:
    - 2 vCPUs e 4 GB de RAM recomendados para cenários de backup mais intensivos;
    - Espaço em disco suficiente para staging de contêineres e logs;
    - Permissões de leitura nos diretórios protegidos e de escrita no diretório de staging.
- Conectividade:
    - Acesso HTTPS ao storage remoto (quando usado);
    - Acesso ao banco relacional (quando não local).

7.2 Servidor/Infraestrutura de Backend

- Banco de dados relacional (PostgreSQL/Supabase ou equivalente) acessível via TLS.
- Storage de objetos compatível com S3 ou serviço de storage integrado (por exemplo, Supabase Storage).
- Configuração de variáveis de ambiente para credenciais, URLs, chaves e parâmetros de chunk/compressão.
- Serviços de monitoramento e coleta de logs conforme necessidade.

7.3 Servidor (Frontend Web)

- Ambiente Node.js 18+ para build e SSR, ou plataforma serverless compatível.
- Requisitos mínimos de recursos (em ambiente dedicado): 1 vCPU e 1 GB de RAM.
- CDN configurada para servir assets estáticos.
- Variáveis de ambiente como NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_KEY, SUPABASE_SERVICE_ROLE_KEY ou equivalentes devidamente definidas.
- Acesso HTTPS ao BaaS (Supabase) e às APIs internas.

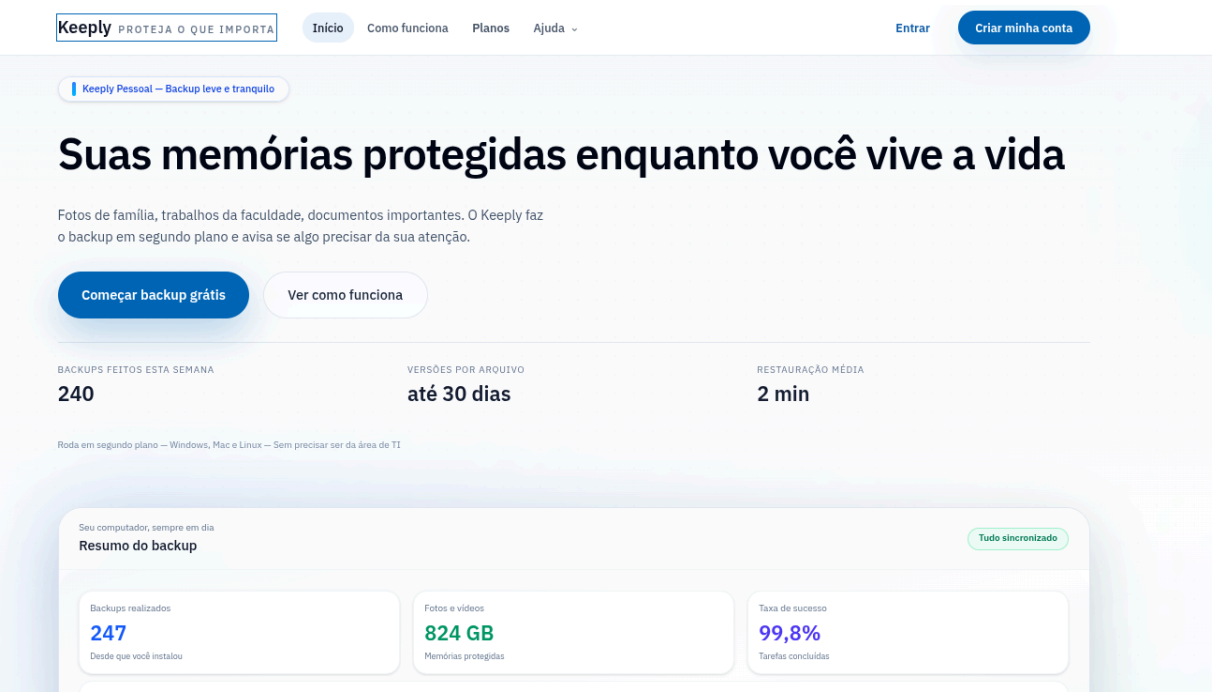
8 -ESPECIFICAÇÃO DE INTERFACE

8.1 Requisitos de Interface e Usabilidade

- Navegação clara, com distinção entre áreas públicas (landing, FAQ, pricing) e área autenticada (dashboard).
- Feedback visual de carregamento, sucesso e erro em operações críticas (login, download, exclusão).
- Textos claros, evitando jargão excessivamente técnico em telas voltadas ao usuário final.

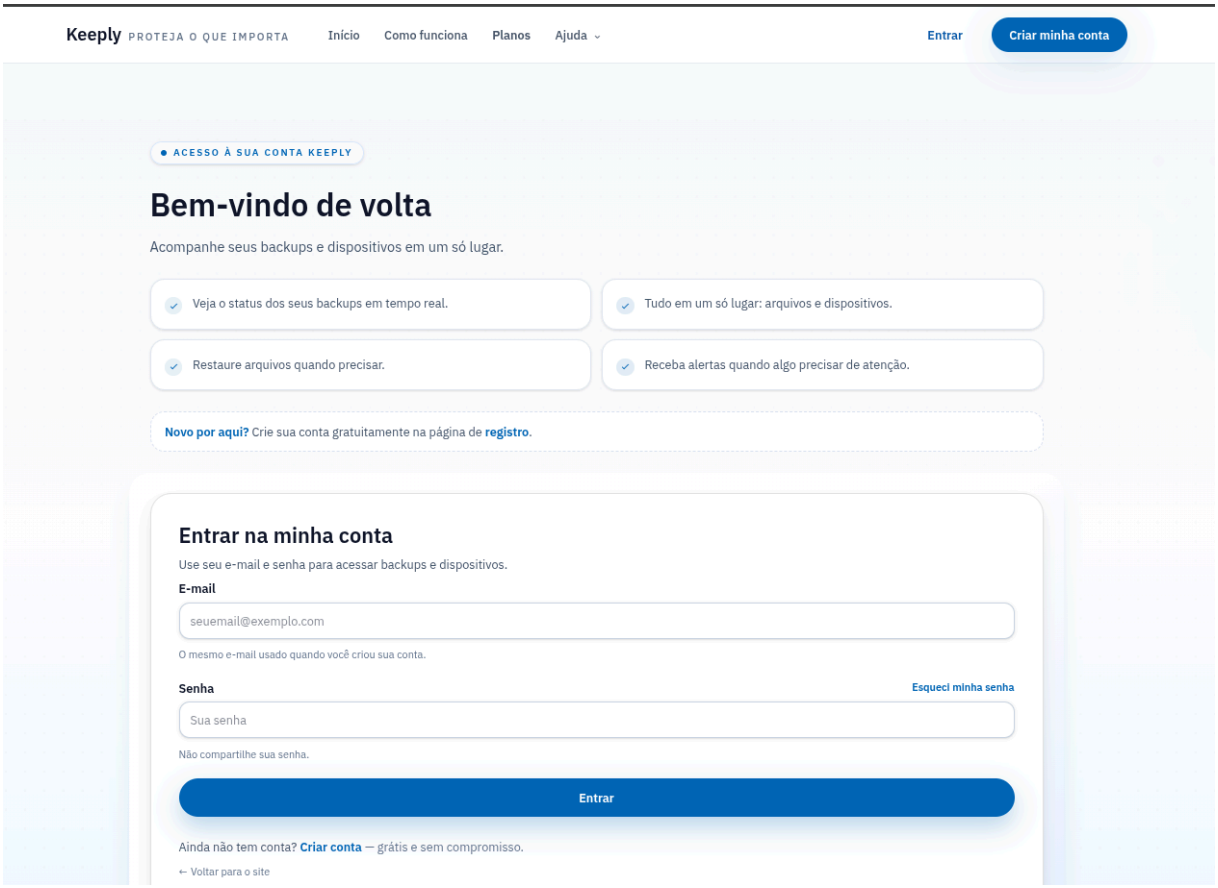
8.2 Protótipos

- **Tela de Landing:**
    - Cabeçalho com logotipo e menu superior (Produto, Preços, FAQ, Entrar).
    - Seção principal (hero) com título e subtítulo explicando a proposta do Keeply, botão “Começar agora”.
    - Seções com benefícios (backup automático, restauração rápida, segurança), cards com ícones e texto curto.
    - Seção de planos/preços e FAQ ao final, seguida de rodapé com contatos



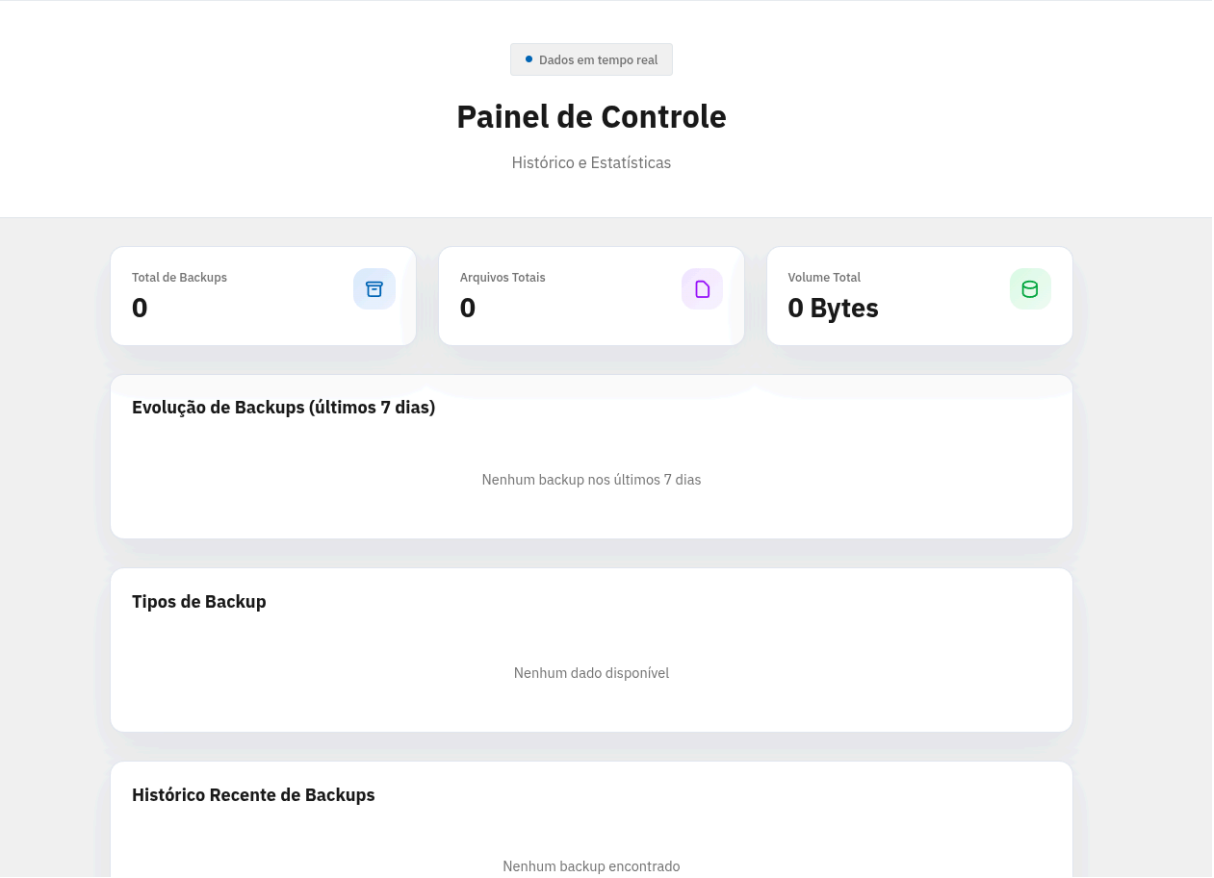
- **Tela de Login/Registro:**

- Formulário centralizado com campos de e-mail e senha.
- Links para registro e “esqueci minha senha”.
- Botão de ação destacado (Entrar/Criar conta).
- Mensagens de erro e validação de forma clara.



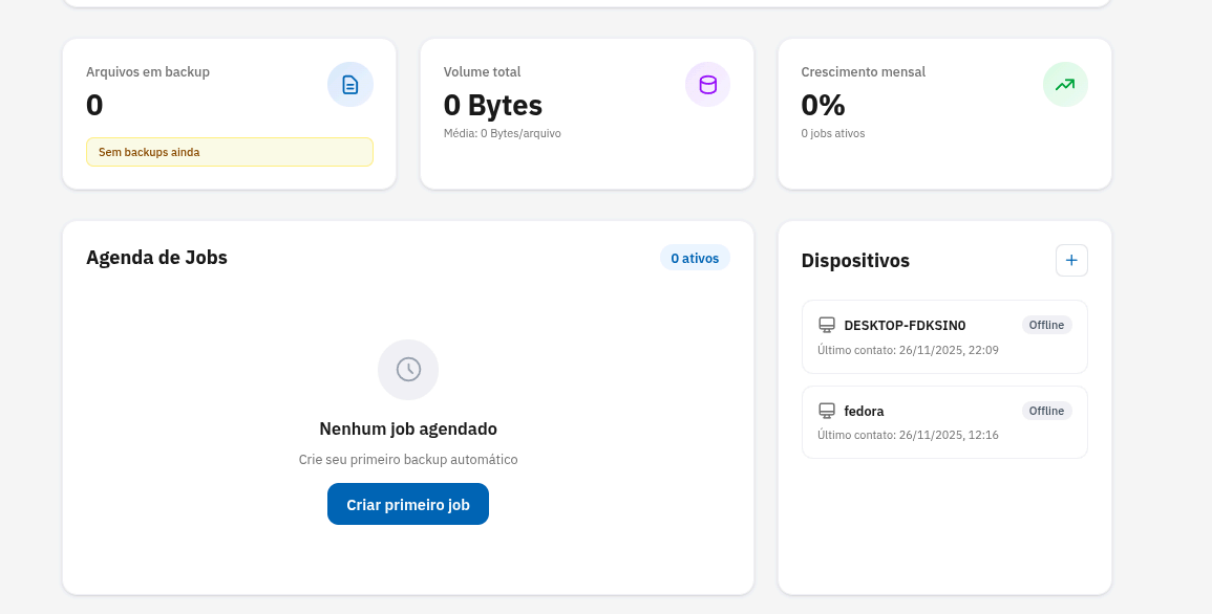
- **Dashboard Autenticado:**

- Barra superior com logo, nome do usuário e botão de sair.
- Cards com métricas agregadas (total de backups, espaço utilizado, dispositivos).
- Gráfico com histórico de uploads por dia.
- Tabela de backups com colunas como nome do snapshot, dispositivo, data, tamanho, status e ações (baixar, gerar URL, remover).
- Aba ou painel lateral com lista de dispositivos e seus status.

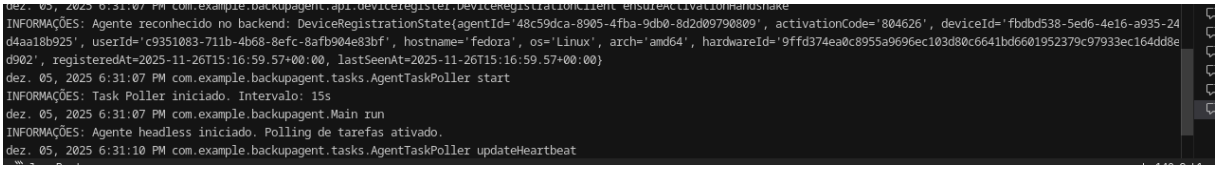


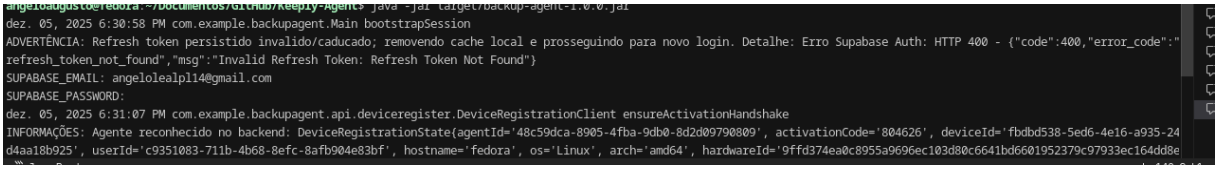
- **Tela/Modal de Detalhe de Backup:**

- Informações detalhadas sobre o snapshot (data, dispositivo, tamanho, status).
- Lista de arquivos ou resumo dos diretórios protegidos.
- Botões para gerar URL assinada e remover backup, com confirmação.



- **Interfaces do Agente (CLI / JavaFX):**
  - CLI com comandos como backup start, backup list, backup restore.
  - UI JavaFX (quando aplicável) com botões para iniciar backup, listar execuções, visualizar logs e status de contêineres.





9- DESCRIÇÃO DO DESENVOLVIMENTO DO SISTEMA PROPOSTO

O desenvolvimento do Sistema Keeply foi conduzido em duas frentes integradas: **agente local** e **frontend web**, sempre orientado pela arquitetura distribuída planejada.

Na frente do **agente**, iniciou-se pela implementação do **scanner**, responsável por percorrer diretórios configurados e detectar alterações, permitindo a geração de planos FULL e INCREMENTAL. Em seguida, foi criado o **packager**, capaz de dividir arquivos em chunks, calcular hashes, aplicar compressão ZSTD e produzir contêineres .tar.zst com cabeçalho KBC, registrando offsets e manifestos para leitura aleatória. Paralelamente, definiu-se o **modelo relacional** de metadados (backups, versões de arquivos, chunks, contêineres), garantindo integridade referencial e rastreabilidade. Foram então implementados os **provedores de storage** (local e S3), com suporte a leitura por faixa e retomada de upload, além do **serviço de restauração**, que reconstrói arquivos e valida integridade. Por fim, adicionou-se a camada de interface (CLI e/ou JavaFX) e o **scheduler**, possibilitando execução automática de backups e registro de histórico.

Na frente do **frontend web**, o trabalho iniciou pela definição da arquitetura em **Next.js App Router**, permitindo combinar páginas estáticas, rotas dinâmicas e APIs internas. Optou-se por Tailwind para padronizar estilos e agilizar prototipagem. Em seguida, foi criado um **AuthContext** para encapsular a integração com Supabase Auth, armazenando sessão e tokens de forma centralizada. Os hooks de domínio (useBackups, useJobs, useDevices) foram responsáveis por consumir as APIs internas, normalizar dados e assinar canais Realtime. As rotas /api/backups, /api/jobs e /api/download foram implementadas com validação de JWT, aplicando regras de propriedade e limites de paginação. Na camada de UI, foram desenvolvidos componentes de **cards de métricas**, **tabelas** e **gráficos** para exibir dados de backup, além de telas de landing, login, registro e recuperação. Ao longo do processo, priorizou-se **observabilidade**, com endpoints de health check e métricas, além de mensagens de erro amigáveis para cenários de falha em Supabase ou no backend.

A integração entre agente e frontend se dá via **banco de dados e storage compartilhados**: o agente escreve metadados e contêineres, enquanto o frontend consulta e apresenta essas informações ao usuário, mediado por autenticação e

políticas de acesso. Essa abordagem consolida o sistema como um **todo arquiteturalmente coeso**.

10 - CONSIDERAÇÕES FINAIS

O Sistema Keeply, tal como apresentado neste trabalho, constitui uma **arquitetura de backup distribuído** que integra um agente robusto de deduplicação e compressão com um painel web moderno de observabilidade e controle. O agente aborda o problema de custo e redundância de armazenamento por meio de chunks, manifestos e contêineres comprimidos, enquanto o frontend aproxima essa complexidade da realidade do usuário, traduzindo execuções de backup em métricas, listas, estados e ações simples.

Os benefícios esperados incluem:

- Redução do volume de dados armazenados devido à deduplicação;
- Diminuição da janela de recuperação em incidentes;
- Maior visibilidade do ciclo de vida de backups;
- Melhoria da experiência de operação, tanto para usuários finais quanto para equipes de TI.

As principais limitações concentram-se na ausência, no escopo atual, de **multi-tenant completo**, **RBAC avançado** e **políticas automáticas de retenção**. Além disso, a dependência de serviços externos implica aderência aos SLAs de terceiros.

Como propostas de trabalhos futuros, destacam-se:

- Implementação de multi-tenant, com isolamento lógico de clientes e papéis diferenciados;
- Ampliação de políticas de retenção automatizadas, vinculadas a custo de storage e janelas de recuperação;
- Criptografia fim a fim dos payloads em repouso, com gerenciamento de chaves separado;
- Dashboards adicionais de SLA (por dispositivo, por usuário, por período);
- Integração com ferramentas externas de observabilidade e alerta.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 14724: informação e documentação – trabalhos acadêmicos – apresentação. Rio de Janeiro: ABNT, 2011.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 6023: informação e documentação – referências – elaboração. Rio de Janeiro: ABNT, 2018.

AWS. *AWS SDK for Java v2 Developer Guide*. Disponível em: https://docs.aws.amazon.com/sdk-for-java/v2/developer-guide/. Acesso em: 07 jan. 2025.

FASTERXML. *Jackson Project*. Disponível em: https://github.com/FasterXML/jackson. Acesso em: 07 jan. 2025.

APACHE SOFTWARE FOUNDATION. *Maven Project – Guides*. Disponível em: https://maven.apache.org/guides/. Acesso em: 07 jan. 2025.

SUPABASE. *Supabase Documentation*. Disponível em: https://supabase.com/docs. Acesso em: 27 fev. 2025.

VERCEL. *Next.js Documentation*. Disponível em: https://nextjs.org/docs. Acesso em: 27 fev. 2025.

APÊNDICE A – TÍTULO DO APÊNDICE (OPCIONAL)

- **Chunk**: bloco de dados resultante da divisão de arquivos para deduplicação.
- **Manifesto de chunk**: metadado que relaciona um conjunto de chunks a uma versão de arquivo e a contêineres específicos.
- **Cabeçalho KBC**: estrutura presente no contêiner que guarda offsets, hashes e metadados de navegação para leitura eficiente.
- **BaaS (Backend as a Service)**: serviço de backend gerenciado que oferece autenticação, banco de dados, storage e outras funcionalidades via APIs.
- **Realtime**: mecanismo de atualização em tempo quase real, permitindo que alterações no backend sejam refletidas instantaneamente no frontend.

ANEXO A – TÍTULO DO ANEXO (OPCIONAL)

- KEEP_REPO_PATH: caminho do repositório local monitorado pelo agente.
- KEEP_DB_URL: URL do banco relacional utilizado pelo agente.
- KEEP_DB_USER / KEEP_DB_PASSWORD: credenciais de acesso ao banco.
- KEEP_S3_BUCKET: nome do bucket de storage.
- KEEP_S3_REGION: região do bucket.
- KEEP_CHUNK_SIZE_MB: tamanho de chunk em megabytes.
- KEEP_COMPRESSION_LEVEL: nível de compressão ZSTD.
- NEXT_PUBLIC_SUPABASE_URL: URL pública do projeto Supabase.
- NEXT_PUBLIC_SUPABASE_ANON_KEY: chave anônima para uso no frontend.
- SUPABASE_SERVICE_ROLE_KEY: chave de serviço para uso em APIs internas (mantida apenas no backend).