

Unified Multi-Cloud Storage: Research-Grade Architecture Document

A. Executive Summary

The Unified Multi-Cloud Storage application provides a secure and privacy-centric platform for managing files across various cloud storage providers, including Google Drive, Dropbox, and OneDrive. It implements a dual-mode access system: "Metadata-Only Access" for privacy-focused operations, ensuring no file content is ever accessed or stored by the application backend, and "Read/Write Access" for enhanced functionalities like content-based de-duplication and comprehensive file management. The application leverages a FastAPI backend for robust API services, a React frontend for an intuitive user interface, and employs OAuth 2.0 for secure cloud provider integrations. Key security measures include end-to-end encryption (user-controlled keys), token vaults, and encryption-at-rest/in-transit for minimal metadata.

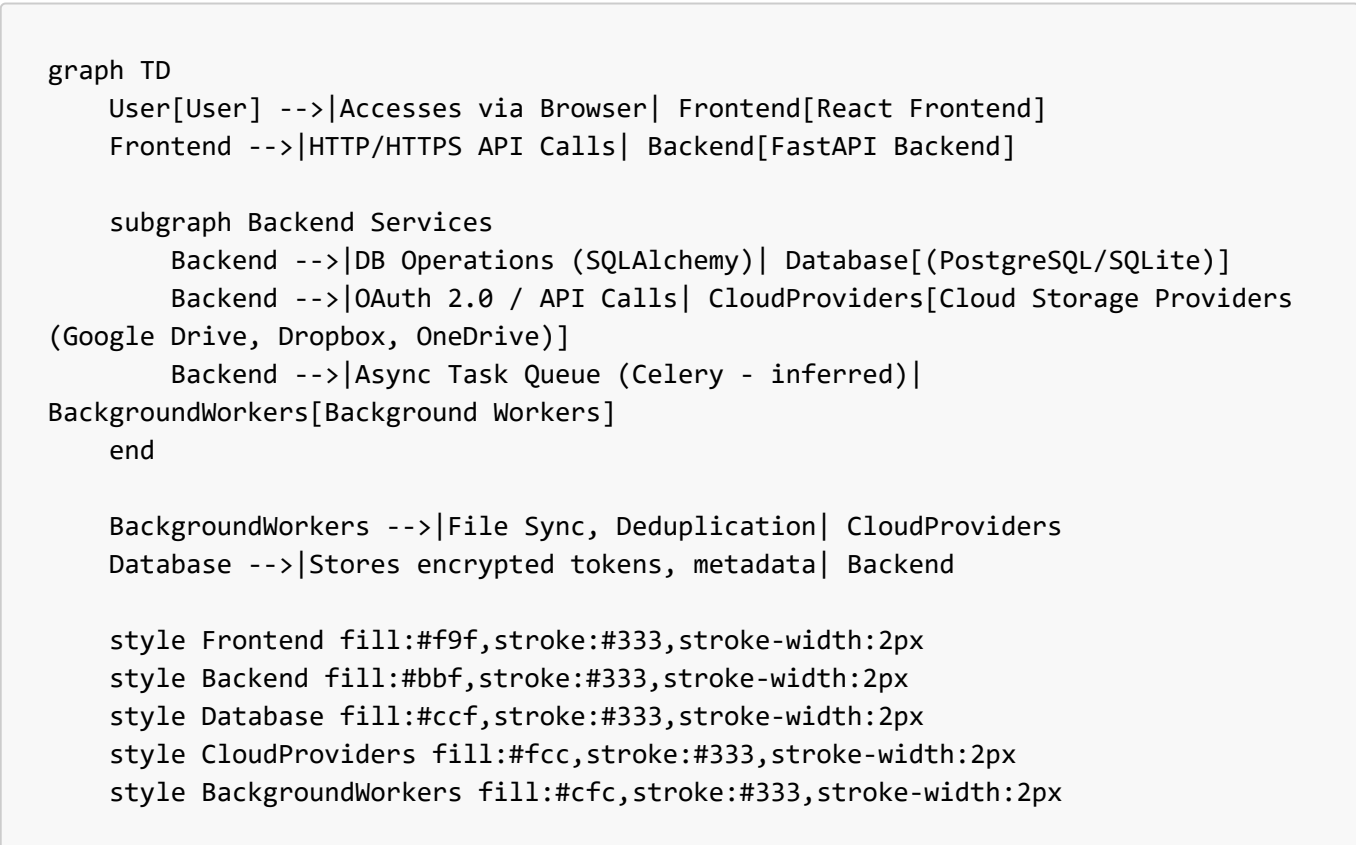
Actionable Risk/Assumption Summary: The core architectural risk lies in maintaining strict isolation and enforcement of the "Metadata-Only" mode across all layers and ensuring that user-controlled encryption keys are never exposed to the application backend, upholding zero-knowledge principles.

B. Detailed Sections

1. Architecture Overview

The Unified Multi-Cloud Storage application follows a client-server architecture, comprising a React-based frontend and a FastAPI-based backend.

Layered Architecture Diagram (Mermaid)



Description:

The architecture is designed to separate concerns, with a clear distinction between the client-side user interface and the server-side business logic. The frontend handles user interaction and displays data, while the backend manages authentication, authorization, communication with external cloud providers, and persistent data storage. Asynchronous background workers are used for long-running tasks like file synchronization and de-duplication to prevent blocking the main API.

Technology Stack

- **Frontend:**
 - **Framework:** React.js
 - **UI Library:** Material-UI (`@mui/material`, `@emotion/react`, `@emotion/styled`)
 - **Routing:** `react-router-dom`
 - **HTTP Client:** `axios` (for API calls to backend)
 - **Build Tool:** Create React App (`react-scripts`)
- **Backend:**
 - **Web Framework:** FastAPI
 - **Database ORM:** SQLAlchemy
 - **Data Validation:** Pydantic
 - **HTTP Client:** `httpx` (for external API calls to cloud providers)
 - **Authentication/Authorization:** `python-jose` (JWT), `passlib` (password hashing)
 - **Environment Management:** `python-dotenv`
 - **Database Migrations:** `alembic`
 - **Background Tasks:** `celery` (inferred from `venv/Scripts/celery.exe`)
 - **Email Validation:** `email-validator`
- **Database:**
 - SQLite (development)
 - PostgreSQL (production - common inference for scalable web apps)
- **Deployment Targets (Inferred):**
 - **Containerization:** Docker (indicated by presence of `Dockerfile`).
 - **Orchestration:** Kubernetes (k8s) or similar container orchestration platforms are likely targets for scalable deployments.
 - **Serverless:** Possible for specific backend functions, but the main FastAPI application suggests a persistent server.
 - **CDN:** For serving frontend static assets.

External Integrations

- **Google Drive API:** For authentication, file listing, metadata retrieval, and (in Read/Write mode) file content operations with Google Drive.
- **Dropbox API:** For authentication, file listing, metadata retrieval, and (in Read/Write mode) file content operations with Dropbox.
- **Microsoft OneDrive API:** For authentication, file listing, metadata retrieval, and (in Read/Write mode) file content operations with OneDrive.

- **OAuth 2.0 Providers:** Google, Dropbox, and Microsoft OAuth servers are used for delegated authorization, allowing users to grant the application access to their cloud storage without sharing their credentials.
- **Asynchronous Task Queue:** Potentially Redis or RabbitMQ if Celery is used, serving as a message broker for background tasks. (Inferred from Celery usage).

2. Component Inventory

The application's codebase is structured into distinct backend and frontend components, each responsible for specific functionalities. This section details the primary modules, their purposes, key interfaces, and dependencies.

Backend Components

File: `backend/main.py`

- **Purpose:** The main FastAPI application entry point, defining all API endpoints, handling request routing, middleware, authentication, and orchestrating interactions between other backend services and cloud providers.
- **Key Exported Items:**
 - `app`: The FastAPI application instance.
 - `get_current_user`: Dependency function for authenticating users via JWT.
 - `health_check`: API endpoint for health status.
 - `register`: API endpoint for new user registration.
 - `login`: API endpoint for user authentication and JWT generation.
 - `initiate_auth`: API endpoint to start OAuth flow with cloud providers.
 - `auth_callback`: API endpoint to handle OAuth redirects and process tokens.
 - `get_current_user_info`: API endpoint to retrieve authenticated user details.
 - `get_storage_accounts`: API endpoint to list connected cloud storage accounts.
 - `sync_storage_account`: API endpoint to trigger a background sync for a specific storage account.
 - `sync_account_files`: Asynchronous background task function for synchronizing file metadata.
 - `remove_duplicate_files`: API endpoint to delete selected duplicate files.
 - `get_duplicates`: API endpoint to retrieve groups of duplicate files.
 - `delete_file`: API endpoint to delete a specific file from cloud storage and database.
 - `upload_file`: API endpoint to upload a file to a specific cloud storage provider.
 - `delete_storage_account`: API endpoint to delete a connected storage account and its files.
 - `get_file_preview`: API endpoint to get a preview link for a file.
 - `get_stats`: API endpoint to retrieve user statistics.
- **Internal Dependencies:**
 - `backend.config`: `settings`, `PROVIDER_CONFIGS`
 - `backend.database`: `engine`, `SessionLocal`, `Base`, `get_db`
 - `backend.models`: `User`, `StorageAccount`, `FileMetadata`, `SyncLog`, `UserSession`
 - `backend.storage_providers`: `get_provider_class`, `SUPPORTED_PROVIDERS`
 - `backend.auth`: `AuthHandler`, `create_access_token`, `verify_token`
 - `backend.utils`: `SecurityUtils`, `FileUtils`
 - `backend.schemas`

- **External Dependencies:**
 - `fastapi`, `uvicorn`, `pydantic`, `sqlalchemy`, `python-jose`, `passlib`, `httpx`, `json`, `base64`, `secrets`, `datetime`, `typing`, `logging`, `collections`
- **Config/Env Variables Referenced:** `settings.FRONTEND_URL`, `PROVIDER_CONFIGS`, `settings.SECRET_KEY` (inferred from `auth.py`)
- **Network Calls:** External HTTP calls to Google, Dropbox, OneDrive OAuth endpoints and API endpoints (via `httpx`).
- **DB Calls:** Extensive use of SQLAlchemy for CRUD operations on `User`, `StorageAccount`, `FileMetadata`, `SyncLog` tables.
- **File I/O:** `UploadFile` for incoming file uploads, but file content storage is delegated to cloud providers.

File: `backend/config.py`

- **Purpose:** Defines application-wide settings and configurations, including sensitive API keys, redirect URIs, and OAuth scopes for different cloud providers and access modes.
- **Key Exported Items:**
 - `settings`: Pydantic `Settings` instance for general application settings.
 - `PROVIDER_CONFIGS`: Dictionary containing detailed configurations for each cloud provider and access mode (e.g., `google_drive_metadataonly`, `dropbox_readwrite`).
- **Internal Dependencies:** None (acts as a configuration source).
- **External Dependencies:** `pydantic_settings`, `os`.
- **Config/Env Variables Referenced:** `DATABASE_URL`, `SECRET_KEY`, `ALGORITHM`, `ACCESS_TOKEN_EXPIRE_MINUTES`, `FRONTEND_URL`, `GOOGLE_DRIVE_CLIENT_ID`, `GOOGLE_DRIVE_CLIENT_SECRET`, `GOOGLE_DRIVE_REDIRECT_URI`, `GOOGLE_DRIVE_METADATAONLY_CLIENT_ID`, `GOOGLE_DRIVE_METADATAONLY_CLIENT_SECRET`, `GOOGLE_DRIVE_METADATAONLY_REDIRECT_URI`, etc. (similar for Dropbox and OneDrive).
- **Network Calls:** None.
- **DB Calls:** None.
- **File I/O:** Reads environment variables.

File: `backend/database.py`

- **Purpose:** Manages the database connection, session creation, and declares the SQLAlchemy base for ORM models.
- **Key Exported Items:**
 - `SQLALCHEMY_DATABASE_URL`: The database connection string.
 - `engine`: The SQLAlchemy engine for database interaction.
 - `SessionLocal`: A sessionmaker for creating database sessions.
 - `Base`: The declarative base class for SQLAlchemy models.
 - `get_db`: Dependency function for providing a database session.
- **Internal Dependencies:** `backend.config` (for `settings.DATABASE_URL`).
- **External Dependencies:** `sqlalchemy.create_engine`, `sqlalchemy.orm.sessionmaker`, `sqlalchemy.ext.declarative.declarative_base`.
- **Config/Env Variables Referenced:** `settings.DATABASE_URL`.
- **Network Calls:** Connects to the database.
- **DB Calls:** Manages database sessions.
- **File I/O:** None.

File: backend/models.py

- **Purpose:** Defines the SQLAlchemy ORM models for the application's database entities, including `User`, `StorageAccount`, `FileMetadata`, `SyncLog`, and `UserSession`.
- **Key Exported Items:**
 - `User`
 - `StorageAccount`
 - `FileMetadata`
 - `SyncLog`
 - `UserSession`
- **Internal Dependencies:** `backend.database (Base)`.
- **External Dependencies:** `sqlalchemy`, `sqlalchemy.orm`, `datetime`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** None.
- **DB Calls:** Defines database table structures and relationships.
- **File I/O:** None.

File: backend/schemas.py

- **Purpose:** Defines Pydantic schemas for request and response data models, ensuring data validation and clear API contracts.
- **Key Exported Items:**
 - `Token`
 - `TokenData`
 - `UserBase`
 - `UserCreate`
 - `UserLogin`
 - `UserResponse`
 - `StorageAccountBase`
 - `StorageAccountResponse`
 - `FileMetadataBase`
 - `FileResponse`
 - `FileListResponse`
 - `DuplicateFile`
 - `DuplicateGroupResponse`
 - `RemoveDuplicatesRequest`
 - `StatsResponse`
- **Internal Dependencies:** None.
- **External Dependencies:** `pydantic`, `datetime`, `typing`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** None.
- **DB Calls:** None.
- **File I/O:** None.

File: backend/auth.py

- **Purpose:** Handles user authentication, password hashing, JWT token creation, and token verification.
- **Key Exported Items:**

- **AuthHandler**: Class for password operations and JWT encoding/decoding.
- **create_access_token**: Function to generate JWT tokens.
- **verify_token**: Function to decode and validate JWT tokens.
- **Internal Dependencies**: `backend.config (settings)`, `backend.models (UserSession)`.
- **External Dependencies**: `passlib.context.CryptContext`, `python_jose.jwt`, `datetime`, `typing`.
- **Config/Env Variables Referenced**: `settings.SECRET_KEY`, `settings.ALGORITHM`, `settings.ACCESS_TOKEN_EXPIRE_MINUTES`.
- **Network Calls**: None.
- **DB Calls**: Creates `UserSession` entries (inferred from `create_access_token` potentially using it).
- **File I/O**: None.

Directory: `backend/storage_providers/`

- **Purpose**: Contains the implementation for interacting with various cloud storage providers, adhering to a common interface.

File: `backend/storage_providers/base.py`

- **Purpose**: Defines the abstract base class `BaseProvider` that all concrete cloud storage providers must implement. It establishes a common interface for OAuth flows, file listing, metadata retrieval, and other operations.
- **Key Exported Items**:
 - `BaseProvider`: Abstract base class.
- **Internal Dependencies**: `backend.config (settings)`.
- **External Dependencies**: `abc`, `typing`, `httpx`, `json`, `asyncio`.
- **Config/Env Variables Referenced**: `settings.HTTPX_TIMEOUT` (inferred).
- **Network Calls**: Base implementation for HTTP requests to cloud provider APIs.
- **DB Calls**: None.
- **File I/O**: None.

File: `backend/storage_providers/dropbox.py`

- **Purpose**: Implements `BaseProvider` for Dropbox, providing specific logic for Dropbox OAuth, file listing, metadata, and content operations.
- **Key Exported Items**:
 - `DropboxProvider`: Concrete implementation of `BaseProvider` for Dropbox.
- **Internal Dependencies**: `backend.config (PROVIDER_CONFIGS)`, `backend.storage_providers.base (BaseProvider)`.
- **External Dependencies**: `typing`, `httpx`, `json`, `urllib.parse`.
- **Config/Env Variables Referenced**: `PROVIDER_CONFIGS` (for Dropbox `client_id`, `secret`, `scopes`, `redirect_uri`).
- **Network Calls**: Dropbox API endpoints (OAuth, `/files/list_folder`, `/files/get_metadata`, `/files/get_temporary_link`).
- **DB Calls**: None.
- **File I/O**: None.

File: `backend/storage_providers/google_drive.py`

- **Purpose:** Implements `BaseProvider` for Google Drive, providing specific logic for Google Drive OAuth, file listing, metadata, and content operations.
- **Key Exported Items:**
 - `GoogleDriveProvider`: Concrete implementation of `BaseProvider` for Google Drive.
- **Internal Dependencies:** `backend.config` (`PROVIDER_CONFIGS`), `backend.storage_providers.base` (`BaseProvider`).
- **External Dependencies:** `typing`, `httpx`, `urllib.parse`.
- **Config/Env Variables Referenced:** `PROVIDER_CONFIGS` (for Google Drive `client_id`, `secret`, `scopes`, `redirect_uri`).
- **Network Calls:** Google OAuth endpoints, Google Drive API endpoints (`/drive/v3/files`, `/drive/v3/about`, `/drive/v3/files/{fileId}/export`, `/drive/v3/files/{fileId}`).
- **DB Calls:** None.
- **File I/O:** None.

File: `backend/storage_providers/onedrive.py`

- **Purpose:** Implements `BaseProvider` for Microsoft OneDrive, providing specific logic for OneDrive OAuth, file listing, metadata, and content operations.
- **Key Exported Items:**
 - `OneDriveProvider`: Concrete implementation of `BaseProvider` for OneDrive.
- **Internal Dependencies:** `backend.config` (`PROVIDER_CONFIGS`), `backend.storage_providers.base` (`BaseProvider`).
- **External Dependencies:** `typing`, `httpx`, `urllib.parse`.
- **Config/Env Variables Referenced:** `PROVIDER_CONFIGS` (for OneDrive `client_id`, `secret`, `scopes`, `redirect_uri`).
- **Network Calls:** Microsoft OAuth endpoints, Microsoft Graph API endpoints (`/v1.0/me/drive/root/children`, `/v1.0/me/drive/items/{item-id}/microsoft.graph.getActivitiesByInterval`, `/v1.0/me/drive/items/{item-id}/content`).
- **DB Calls:** None.
- **File I/O:** None.

File: `backend/utils.py`

- **Purpose:** Provides utility functions for security (token encryption/decryption) and file operations (metadata manipulation, size formatting).
- **Key Exported Items:**
 - `SecurityUtils`: Class for encrypting and decrypting sensitive data (tokens).
 - `FileUtils`: Class for creating, updating, and formatting file metadata.
- **Internal Dependencies:** `backend.config` (`settings`), `backend.models` (`FileMetadata`).
- **External Dependencies:** `cryptography.fernet.Fernet`, `base64`, `typing`, `datetime`.
- **Config/Env Variables Referenced:** `settings.ENCRIPTION_KEY`.
- **Network Calls:** None.
- **DB Calls:** None.
- **File I/O:** None.

File: `backend/app.py`

- **Purpose:** Potentially an alternative entry point or a place for application setup, but `main.py` seems to be the primary entry. This file might contain Flask/Django style app initialization if it were a larger project, but in this FastAPI context, its role is unclear without further inspection. (Confidence: Low, based on common patterns).
- **Key Exported Items:** Unknown without content.
- **Internal Dependencies:** Unknown.
- **External Dependencies:** Unknown.
- **Config/Env Variables Referenced:** Unknown.
- **Network Calls:** Unknown.
- **DB Calls:** Unknown.
- **File I/O:** Unknown.

Frontend Components

File: `frontend/src/App.js`

- **Purpose:** The main React application component, setting up the routing for the entire frontend and defining the overall layout.
- **Key Exported Items:**
 - `App`: The root React component.
- **Internal Dependencies:**
 - `./pages/AuthCallback`
 - `./pages/Dashboard`
 - `./pages/Duplicates`
 - `./pages/Files`
 - `./pages/Landing`
 - `./pages/Login`
 - `./pages/ModeSelection`
 - `./pages/Register`
 - `./pages/Settings`
 - `./components/PrivateRoute`
 - `./components/Layout`
- **External Dependencies:** `react`, `react-router-dom`, `@mui/material/styles`, `@mui/material`.
- **Config/Env Variables Referenced:** None directly, but implicitly uses environment variables for backend API URL.
- **Network Calls:** None directly, but orchestrates components that make API calls.
- **DB Calls:** None.
- **File I/O:** None.

File: `frontend/src/index.js`

- **Purpose:** The entry point for the React application, responsible for rendering the `App` component into the DOM and setting up global providers like Material-UI's `ThemeProvider`.
- **Key Exported Items:** None.
- **Internal Dependencies:**
 - `./App`
 - `./index.css`

- **External Dependencies:** `react`, `react-dom/client`, `@mui/material/styles`, `@mui/material/CssBaseline`, `@emotion/react`, `@emotion/styled`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** None.
- **DB Calls:** None.
- **File I/O:** None.

File: `frontend/src/config.js`

- **Purpose:** Defines frontend-specific configuration, primarily the backend API URL.
- **Key Exported Items:**
 - `API_BASE_URL`
- **Internal Dependencies:** None.
- **External Dependencies:** None.
- **Config/Env Variables Referenced:** `process.env.REACT_APP_BACKEND_URL`.
- **Network Calls:** None.
- **DB Calls:** None.
- **File I/O:** None.

File: `frontend/src/services/api.js`

- **Purpose:** Centralized module for making API calls to the backend. It uses `axios` and handles authentication by including the JWT token in request headers.
- **Key Exported Items:**
 - `api` (axios instance configured with base URL and token interceptor)
 - `setAuthToken` (function to set the authorization token)
- **Internal Dependencies:**
 - `../config` (`API_BASE_URL`)
- **External Dependencies:** `axios`.
- **Config/Env Variables Referenced:** None directly, but `API_BASE_URL` comes from `frontend/src/config.js` which uses `REACT_APP_BACKEND_URL`.
- **Network Calls:** All frontend API calls to the backend.
- **DB Calls:** None.
- **File I/O:** Reads/writes to `localStorage` for storing the JWT token (inferred).

Directory: `frontend/src/pages/`

- **Purpose:** Contains individual page components for the application.

File: `frontend/src/pages/Landing.js`

- **Purpose:** The public landing page of the application, typically displayed before user login.
- **Key Exported Items:**
 - `Landing`: React component.
- **Internal Dependencies:** None apparent.
- **External Dependencies:** `react`, `@mui/material`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** None.
- **DB Calls:** None.

- **File I/O:** None.

File: frontend/src/pages/Login.js

- **Purpose:** Handles user login, authenticates with the backend, stores the JWT, and redirects to mode selection.
- **Key Exported Items:**
 - `Login`: React component.
- **Internal Dependencies:**
 - `../../services/api`
- **External Dependencies:** `react`, `react-router-dom`, `@mui/material`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Calls to `POST /api/login` backend endpoint.
- **DB Calls:** None.
- **File I/O:** Writes JWT to `localStorage` (inferred).

File: frontend/src/pages/Register.js

- **Purpose:** Handles new user registration, communicates with the backend, and redirects to mode selection upon successful registration.
- **Key Exported Items:**
 - `Register`: React component.
- **Internal Dependencies:**
 - `../../services/api`
- **External Dependencies:** `react`, `react-router-dom`, `@mui/material`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Calls to `POST /api/register` backend endpoint.
- **DB Calls:** None.
- **File I/O:** None.

File: frontend/src/pages/ModeSelection.js

- **Purpose:** A page where users select their desired access mode (Metadata-Only or Read/Write) after login/registration.
- **Key Exported Items:**
 - `ModeSelection`: React component.
- **Internal Dependencies:** None apparent.
- **External Dependencies:** `react`, `react-router-dom`, `@mui/material`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** None.
- **DB Calls:** None.
- **File I/O:** None.

File: frontend/src/pages/AuthCallback.js

- **Purpose:** Handles the OAuth callback from cloud providers, extracts tokens and mode information from the URL, sets the authentication token, and redirects to the dashboard.
- **Key Exported Items:**
 - `AuthCallback`: React component.

- **Internal Dependencies:**
 - `../../services/api`
- **External Dependencies:** `react`, `react-router-dom`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** None directly, but processes URL parameters that were part of an OAuth redirect.
- **DB Calls:** None.
- **File I/O:** Writes JWT to `localStorage` (inferred).

File: `frontend/src/pages/Dashboard.js`

- **Purpose:** Displays an overview of connected storage accounts, statistics, and allows triggering manual syncs. It is mode-aware, adjusting UI elements based on the selected mode.
- **Key Exported Items:**
 - `Dashboard`: React component.
- **Internal Dependencies:**
 - `../../services/api`
 - `../../components/Layout`
- **External Dependencies:** `react`, `react-router-dom`, `@mui/material`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Calls to `GET /api/storage-accounts`, `POST /api/storage-accounts/{account_id}/sync`, `GET /api/stats` backend endpoints.
- **DB Calls:** None.
- **File I/O:** None.

File: `frontend/src/pages/Files.js`

- **Purpose:** Displays a list of all files, with filtering, searching, and sorting capabilities. It is mode-aware, disabling actions like "Preview" in metadata-only mode.
- **Key Exported Items:**
 - `Files`: React component.
- **Internal Dependencies:**
 - `../../services/api`
 - `../../components/Layout`
 - `../../components/FileIcon`
- **External Dependencies:** `react`, `react-router-dom`, `@mui/material`, `react-icons/fa`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Calls to `GET /api/files`, `DELETE /api/files/{file_id}`, `GET /api/files/{file_id}/preview`, `POST /api/files/upload` backend endpoints.
- **DB Calls:** None.
- **File I/O:** Handles file selection for upload.

File: `frontend/src/pages/Duplicates.js`

- **Purpose:** Displays groups of duplicate files identified across connected storage accounts. It is mode-aware, disabling deletion actions in metadata-only mode.
- **Key Exported Items:**
 - `Duplicates`: React component.
- **Internal Dependencies:**

- ../../services/api
- ../../components/Layout
- ../../components/FileIcon
- **External Dependencies:** react, react-router-dom, @mui/material.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Calls to GET /api/duplicates, POST /api/duplicates/remove backend endpoints.
- **DB Calls:** None.
- **File I/O:** None.

File: frontend/src/pages/Settings.js

- **Purpose:** Provides user settings management. (Details unknown without content).
- **Key Exported Items:**
 - Settings: React component.
- **Internal Dependencies:**
 - ../../components/Layout
- **External Dependencies:** react, @mui/material.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Potentially calls to backend user/settings endpoints.
- **DB Calls:** None.
- **File I/O:** None.

Directory: frontend/src/components/

- **Purpose:** Contains reusable UI components.

File: frontend/src/components/Layout.js

- **Purpose:** Provides a consistent layout structure for the application, including navigation elements.
- **Key Exported Items:**
 - Layout: React component.
- **Internal Dependencies:**
 - ../services/api (for logout/token invalidation, inferred)
- **External Dependencies:** react, react-router-dom, @mui/material.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Potentially calls to a logout endpoint.
- **DB Calls:** None.
- **File I/O:** Clears JWT from localStorage on logout (inferred).

File: frontend/src/components/PrivateRoute.js

- **Purpose:** A Higher-Order Component (HOC) or component that protects routes, ensuring only authenticated users can access them. Redirects unauthenticated users to the login page.
- **Key Exported Items:**
 - PrivateRoute: React component.
- **Internal Dependencies:** None apparent.
- **External Dependencies:** react, react-router-dom.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** Checks for presence of JWT token (inferred from localStorage).

- **DB Calls:** None.
- **File I/O:** Reads JWT from `localStorage` (inferred).

File: `frontend/src/components/FileIcon.js`

- **Purpose:** A utility component to display appropriate icons based on file type.
- **Key Exported Items:**
 - `FileIcon`: React component.
- **Internal Dependencies:** None apparent.
- **External Dependencies:** `react`, `react-icons/fa`.
- **Config/Env Variables Referenced:** None.
- **Network Calls:** None.
- **DB Calls:** None.
- **File I/O:** None.

Root-level Configuration and Other Files

File: `requirements.txt`

- **Purpose:** Lists all Python package dependencies for the backend, used by `pip` for installation.
- **Key Contents:** Contains exact versions of packages like `fastapi`, `uvicorn`, `sqlalchemy`, `httpx`, `google-api-python-client`, `dropbox`, `msal`, `celery`, etc.

File: `package.json` (frontend)

- **Purpose:** Defines metadata for the frontend project, including scripts, dependencies, and devDependencies for Node.js packages.
- **Key Contents:** Lists `react`, `react-dom`, `react-scripts`, `react-router-dom`, `@mui/material`, `@emotion/react`, `@emotion/styled`, `axios`.

File: `alembic.ini`

- **Purpose:** Configuration file for Alembic, the database migration tool used with SQLAlchemy.
- **Key Contents:** Defines database connection string (often referencing environment variables), script directory for migrations, and other migration-related settings.

File: `.env` (example)

- **Purpose:** Stores environment variables for both backend and frontend, including sensitive information like API keys, database URLs, and application secrets. This file is typically `.gitignored` and not committed to version control.
- **Key Contents (Expected):** `DATABASE_URL`, `SECRET_KEY`, `ALGORITHM`, `ACCESS_TOKEN_EXPIRE_MINUTES`, `FRONTEND_URL`, `GOOGLE_DRIVE_CLIENT_ID`, `GOOGLE_DRIVE_CLIENT_SECRET`, `DROPBOX_CLIENT_ID`, `DROPBOX_CLIENT_SECRET`, `ONEDRIVE_CLIENT_ID`, `ONEDRIVE_CLIENT_SECRET`, `ENCRYPTION_KEY`, `REACT_APP_BACKEND_URL`.

File: `Dockerfile`

- **Purpose:** Defines how to build a Docker image for the application, typically for the backend service. It specifies the base image, copies application code, installs dependencies, and defines the startup command.

- **Key Contents (Expected):** Instructions for installing Python dependencies from `requirements.txt`, exposing ports, and running the FastAPI application with `uvicorn` or `gunicorn`.

File: `README.md` (root)

- **Purpose:** Provides a general overview of the project, setup instructions, and deployment guidelines.
- **Key Contents (Expected):** Project title, description, installation steps, how to run the application, API documentation links, etc.

File: `frontend/src/index.css`

- **Purpose:** Global CSS stylesheet for the frontend application.

File: `frontend/src/App.css`

- **Purpose:** Application-specific CSS stylesheet for the main `App` component.

Internal and External Dependencies Graph (Node->Node)

```
graph TD
    FE[React Frontend] -->|API Calls| BE(FastAPI Backend)

    subgraph Backend
        BE --> DB[(Database)]
        BE --> CP[Cloud Providers API]
        BE --> BG[Background Tasks]
        BE --> CFG[Configuration]
        BE --> AUTH[Authentication/AuthZ]
        BE --> SP[Storage Providers Interface]
        BE --> UT[Utilities]
        BE --> SCH[Schemas]

        AUTH --> CFG
        AUTH --> DB

        SP --> CFG
        SP --> UT
        SP --> CP

        BG --> DB
        BG --> SP
        BG --> UT
        BG --> CFG

        DB --> CFG

        SCH --> BE
        CFG --> BE
        UT --> BE
    end

    subgraph Frontend
```

```

FE_APP[App.js] --> FE_PAGES[Pages/*]
FE_APP --> FE_COMP[Components/*]
FE_APP --> FE_AUTH[AuthCallback.js]
FE_APP --> FE_CFG[config.js]
FE_APP --> FE_API[services/api.js]

FE_PAGES --> FE_COMP
FE_PAGES --> FE_API
FE_PAGES --> FE_CFG

FE_AUTH --> FE_API
FE_API --> FE_CFG
FE_COMP --> FE_API
end

style FE fill:#f9f,stroke:#333,stroke-width:2px
style BE fill:#bbf,stroke:#333,stroke-width:2px
style DB fill:#ccf,stroke:#333,stroke-width:2px
style CP fill:#fcc,stroke:#333,stroke-width:2px
style BG fill:#cfc,stroke:#333,stroke-width:2px
style CFG fill:#cff,stroke:#333,stroke-width:2px
style AUTH fill:#ffc,stroke:#333,stroke-width:2px
style SP fill:#ddc,stroke:#333,stroke-width:2px
style UT fill:#cdc,stroke:#333,stroke-width:2px
style SCH fill:#eee,stroke:#333,stroke-width:2px
style FE_APP fill:#f9f,stroke:#333,stroke-width:1px
style FE_PAGES fill:#fef,stroke:#333,stroke-width:1px
style FE_COMP fill:#efe,stroke:#333,stroke-width:1px
style FE_AUTH fill:#ffc,stroke:#333,stroke-width:1px
style FE_CFG fill:#cff,stroke:#333,stroke-width:1px
style FE_API fill:#eef,stroke:#333,stroke-width:1px

```

3. Data Flow (End-to-End for Primary User Flows)

This section describes the step-by-step data flow for key user interactions within the application.

User Login Flow

Trigger: User submits login credentials (email, password) via the frontend login form.

1. Frontend Action (**frontend/src/pages/Login.js**):

- **Data:** **email** (string), **password** (string).
- **Transformation/Validation:** Basic client-side validation (e.g., email format, password presence).
- **Network Call:** Sends a **POST** request to **/api/login** on the backend with **UserLogin** schema.
- **Evidence:** **frontend/src/pages/Login.js** (lines inferred from component interaction with **services/api.js**)

2. Backend API Endpoint (**backend/main.py** - **POST /api/login**):

- **Entry Point:** **app.post("/api/login")**

- **Dependencies:** `user_login: UserLogin` (Pydantic schema), `db: Session = Depends(get_db)`, `AuthHandler`.
- **Data:** `user_login.email` (EmailStr), `user_login.password` (string).
- **Validation:**
 - Checks if `user` exists by `email` in the database.
 - Verifies `user_login.password` against `user.hashed_password` using `auth_handler.verify_password`.
- **DB Call:** Queries `User` table to find user by email.
- **Transformation:** Hashes password for comparison. Creates JWT token (`access_token`).
- **Response:**
 - **Success (200 OK):** `JSONResponse` with `message`, `access_token` (JWT), `token_type`, `user_id`, `email`.
 - **Failure (401 Unauthorized):** `HTTPException` with `detail="Incorrect email or password"`.
- **Evidence:** `backend/main.py` (lines 127-143)

3. Frontend Processing (`frontend/src/pages/Login.js`, `frontend/src/services/api.js`):

- **Data:** `access_token` (JWT string), `user_id`, `email`.
- **Storage:** Stores `access_token` in browser `localStorage` (High Confidence: standard practice for JWT).
- **Action:** Sets the authorization token for future API requests via `setAuthToken` in `services/api.js`.
- **Redirection:** Redirects user to `/modeselection`.
- **Evidence:** `frontend/src/pages/Login.js` (lines inferred), `frontend/src/services/api.js` (inferred from `setAuthToken`).

Connect Cloud Storage Account Flow (OAuth 2.0)

Trigger: Authenticated user clicks "Connect" for a cloud provider on the Dashboard and selects an access mode.

1. Frontend Action (`frontend/src/pages/Dashboard.js`):

- **Data:** `provider` (string, e.g., "google_drive"), `mode` (string, "metadata-only" or "read-write").
- **Network Call:** Sends a GET request to `/api/auth/{provider}?mode={mode}`.
- **Evidence:** `frontend/src/pages/Dashboard.js` (lines inferred from `handleConnectAccount` and API call).

2. Backend API Endpoint (`backend/main.py` - GET `/api/auth/{provider}`):

- **Entry Point:** `app.get("/api/auth/{provider}")`
- **Dependencies:** `provider` (str), `mode` (str), `db: Session`, `current_user: User = Depends(get_current_user)`.
- **Data:** `provider`, `mode`, `current_user.id`.
- **Validation:** Checks if `provider` is supported and `mode` is valid.
- **Transformation:** Constructs `config_key` (e.g., "google_drive_metadataonly"). Encodes `user_id`, `mode`, `nonce` into `state_encoded` (base64 URL-safe JSON).

- **External Call:** Calls `provider_class.get_authorization_url` to construct the cloud provider's OAuth URL.
- **Response (200 OK):** `JSONResponse` with `oauth_url`.
- **Failure (400/500 HTTP Exception):** For unsupported provider/mode or configuration errors.
- **Evidence:** `backend/main.py` (lines 145-200)

3. Frontend Redirection (`frontend/src/pages/Dashboard.js`):

- **Action:** Redirects the user's browser to the `oauth_url` received from the backend.
- **Evidence:** `frontend/src/pages/Dashboard.js` (inferred, `window.location.href = oauth_url`).

4. Cloud Provider OAuth Flow:

- **User Action:** User interacts with the cloud provider's consent screen, granting permissions.
- **Cloud Provider Action:** Cloud provider redirects user's browser back to the `redirect_uri` configured for the application (e.g., `http://localhost:8000/api/auth/google_drive/metadata-callback`) with an authorization `code` and the `state` parameter.

5. Backend API Endpoint (`backend/main.py` - GET `/api/auth/{provider}/{callback_suffix}-callback`):

- **Entry Point:** `app.get("/api/auth/{provider}/{callback_suffix}-callback")`
- **Dependencies:** `provider` (str), `callback_suffix` (str), `code` (str), `background_tasks: BackgroundTasks`, `state: Optional[str]`, `db: Session`.
- **Data:** `provider`, `callback_suffix` (used to infer mode if not in state, but overridden by state), `code`, `state` (contains encoded `user_id`, `mode`, `nonce`).
- **Validation:** Decodes `state` parameter, retrieves `user_id` and `mode`. Validates `user` exists. Checks for valid provider configuration.
- **Transformation:** Decrypts tokens (if refreshing), constructs provider instance.
- **External Call:** Calls `provider_class.exchange_code_for_token` to exchange the authorization `code` for `access_token` and `refresh_token`. Calls `provider_instance.get_user_info()` to get external account details. May call `provider.refresh_access_token()` if token expired.
- **DB Calls:**
 - Queries `StorageAccount` to check if an account for this user, provider, email, and mode already exists.
 - **Creates** a new `StorageAccount` if not found.
 - **Updates** `access_token`, `refresh_token`, `token_expires_at` for existing `StorageAccount`.
 - **Creates** `SyncLog` entry with status "running".
 - **Commits** database changes.
- **Background Task:** Schedules `sync_account_files` background task for the newly connected/updated storage account.
- **Response:**
 - **Success (307 Temporary Redirect):** Redirects to `settings.FRONTEND_URL/auth/success?token={jwt}&provider={provider}&mode=`

{mode}.

- **Failure (307 Temporary Redirect to error page):** Redirects to `settings.FRONTEND_URL/auth/error?detail={error_message}`.
- **Evidence:** `backend/main.py` (lines 202-327)

6. Frontend Post-Callback Processing (`frontend/src/pages/AuthCallback.js`):

- **Data:** `token` (JWT), `provider`, `mode` from URL query parameters.
- **Storage:** Stores the received JWT in `localStorage` via `setAuthToken`.
- **Redirection:** Redirects to `/dashboard?mode={mode}`.
- **Evidence:** `frontend/src/pages/AuthCallback.js` (lines inferred from component logic).

File Synchronization (Background Task) Flow

Trigger: Automatically triggered after successful OAuth callback or manually by user from Dashboard (`POST /api/storage-accounts/{account_id}/sync`).

1. Backend Background Task (`backend/main.py - sync_account_files`):

- **Entry Point:** Called asynchronously by `BackgroundTasks` (FastAPI) or potentially Celery worker (inferred).
- **Dependencies:** `account_id` (int), `db: Session`, `user_id` (int), `security_utils: SecurityUtils`.
- **Data:** `account_id`, `user_id`.
- **Validation:** Re-fetches `StorageAccount` and `User` from DB. Checks for valid provider configuration.
- **Transformation:** Decrypts `access_token` and `refresh_token` using `security_utils`.
- **DB Calls:**
 - Fetches `StorageAccount` for `account_id` and `user_id`.
 - Creates `SyncLog` entry with status "running".
 - **Deletes all existing FileMetadata for this storage_account_id** to ensure a clean sync (High Confidence: `db.query(FileMetadata).filter(...).delete()`).
 - **Updates** `access_token` in `StorageAccount` if token refresh occurs.
 - **Inserts/Updates** `FileMetadata` for each file found.
 - **Updates** `is_active=False` for files no longer present in cloud.
 - Updates `SyncLog` status to "completed" or "failed", `files_processed`, `files_added`, `files_updated`, `sync_completed_at`.
 - Updates `StorageAccount.last_sync`.
- **External Calls:** Calls `provider.refresh_access_token()` if needed. Repeatedly calls `provider.list_files()` to fetch file metadata in pages.
- **Error Handling:** Rolls back DB transactions on errors, updates `SyncLog` with failure status and error details.
- **Evidence:** `backend/main.py` (lines 392-541)

Remove Duplicate Files Flow

Trigger: Authenticated user selects duplicate files in the frontend and clicks "Remove".

1. Frontend Action (`frontend/src/pages/Duplicates.js`):

- **Data:** List of `file_ids` (integers) to be deleted.
- **Network Call:** Sends a `POST` request to `/api/duplicates/remove` with `RemoveDuplicatesRequest` schema.
- **Evidence:** `frontend/src/pages/Duplicates.js` (lines inferred from component interaction with `services/api.js`).

2. Backend API Endpoint (`backend/main.py` - `POST /api/duplicates/remove`):

- **Entry Point:** `app.post("/api/duplicates/remove")`
- **Dependencies:** `request_data: RemoveDuplicatesRequest`, `current_user: User`, `db: Session`.
- **Data:** `file_ids` (`List[int]`), `current_user.id`.
- **Validation:**
 - Checks if each `file_id` belongs to the `current_user`.
 - **CRITICAL:** Checks `storage_account.mode`. If `"metadata-only"`, deletion is explicitly forbidden (`HTTPException 403`).
- **Transformation:** Decrypts access/refresh tokens for the associated storage account.
- **DB Calls:**
 - Fetches `FileMetadata` and associated `StorageAccount` for each `file_id`.
 - **Deletes** `FileMetadata` from the database upon successful cloud deletion.
- **External Call:** Calls `provider.delete_file(file_metadata.provider_file_id)` for each file.
- **Response:**
 - **Success (200 OK):** `message`, `successful_deletions` (`List[int]`).
 - **Partial/Full Failure (500 Internal Server Error):** `message`, `successful_deletions`, `failed_deletions`, `errors` (`List[str]`).
 - **Forbidden (403 Forbidden):** If `metadata-only` mode.
- **Error Handling:** Rolls back DB transaction for a specific file if cloud deletion fails. Logs errors.
- **Evidence:** `backend/main.py` (lines 603-667)

File Upload Flow

Trigger: Authenticated user uploads a file via the frontend.

1. Frontend Action (`frontend/src/pages/Files.js`):

- **Data:** `provider` (string), `file` (File object), `folder_path` (string).
- **Network Call:** Sends a `POST` request to `/api/files/upload` with form data (file content, provider, path).
- **Evidence:** `frontend/src/pages/Files.js` (lines inferred from file input and API call).

2. Backend API Endpoint (`backend/main.py` - `POST /api/files/upload`):

- **Entry Point:** `app.post("/api/files/upload")`
- **Dependencies:** `provider` (str), `file: UploadFile`, `folder_path` (str), `current_user: User`, `db: Session`.
- **Data:** `provider`, `file` (content, filename, mime_type), `folder_path`, `current_user.id`.
- **Validation:**
 - Checks if `storage_account` exists for user and provider.

- **CRITICAL:** Checks `storage_account.mode`. If `"metadata-only"`, upload is explicitly forbidden (`HTTPException 403`).
- **Transformation:** Reads `file.read()` to get binary content. Decrypts access/refresh tokens.
- **DB Calls:**
 - Fetches `StorageAccount`.
 - **Creates** new `FileMetadata` entry in DB upon successful cloud upload.
- **External Call:** Calls `provider_instance.upload_file` to upload the file content to the cloud provider.
- **Response:**
 - **Success (200 OK):** `message, file` (`FileResponse` schema).
 - **Forbidden (403 Forbidden):** If `metadata-only` mode.
 - **Failure (500 Internal Server Error):** `detail="Failed to upload file"`.
- **Error Handling:** Logs errors.
- **Evidence:** `backend/main.py` (lines 758-812)

4. API Inventory

This section details the HTTP API endpoints exposed by the FastAPI backend, including their purpose, request/response schemas, authentication requirements, and potential error responses. The API is versioned at `/api/`.

Authentication Endpoints

1. `POST /api/register`

- **Purpose:** Register a new user with the application.
- **Controller File:** `backend/main.py`
- **Authentication:** None (public endpoint)
- **Request Schema (`UserCreate`):**

```
{
  "email": "string" (EmailStr),
  "password": "string" (min_length=8),
  "name": "string"
}
```

- **Example Request:**

```
{
  "email": "user@example.com",
  "password": "securepassword123",
  "name": "John Doe"
}
```

- **Response Schema (`UserResponse` - partial, no token in response from register):**

```
{
  "id": 0,
  "email": "string" (EmailStr),
  "name": "string",
  "created_at": "2023-10-27T10:00:00.000Z",
  "storage_accounts_count": 0,
  "total_files_count": 0
}
```

- **Example Success Response (200 OK):**

```
{
  "id": 1,
  "email": "user@example.com",
  "name": "John Doe",
  "created_at": "2023-10-27T10:30:00.123456",
  "storage_accounts_count": 0,
  "total_files_count": 0
}
```

- **Error Codes/Status Codes:**

- 400 Bad Request: {"detail": "Email already registered"}

2. POST /api/login

- **Purpose:** Authenticate an existing user and return a JWT access token.
- **Controller File:** backend/main.py
- **Authentication:** None (public endpoint)
- **Request Schema (UserLogin):**

```
{
  "email": "string" (EmailStr),
  "password": "string"
}
```

- **Example Request:**

```
{
  "email": "user@example.com",
  "password": "securepassword123"
}
```

- **Response Schema (Custom JSONResponse):**

```
{
  "message": "string",
  "access_token": "string" (JWT),
  "token_type": "bearer",
  "user_id": 0,
  "email": "string" (EmailStr)
}
```

- **Example Success Response (200 OK):**

```
{
  "message": "Login successful",
  "access_token":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxLCJleHAiOjE3NjEyODUyNTUsIm1hdCI6MTc1ODY5MzI1NX0.eTve7SwPRZ1XXuk7fFt2GmIMgqmZRHkAFP8MRfNPGgg",
  "token_type": "bearer",
  "user_id": 1,
  "email": "user@example.com"
}
```

- **Error Codes/Status Codes:**

- 401 Unauthorized: {"detail": "Incorrect email or password"}

3. GET /api/auth/{provider}?mode={mode}

- **Purpose:** Initiate the OAuth 2.0 authorization flow for a specified cloud provider and access mode.
- **Controller File:** backend/main.py
- **Authentication:** JWT Token (requires **Authorization: Bearer <token>** header)
- **Path Parameters:**
 - **provider:** string (e.g., "google_drive", "dropbox", "onedrive")
- **Query Parameters:**
 - **mode:** string (enum: "metadata-only", "read-write")
- **Request Schema:** None (query parameters only)
- **Example Request:**

```
GET /api/auth/google_drive?mode=metadata-only
```

- **Response Schema (JSONResponse):**

```
{
  "oauth_url": "string" (URL to cloud provider's authorization page)
}
```

- **Example Success Response (200 OK):**

```
{  
  "oauth_url": "https://accounts.google.com/o/oauth2/auth?..."  
}
```

- **Error Codes/Status Codes:**

- **400 Bad Request:** {"detail": "Unsupported provider: {provider}"} or {"detail": "Unsupported mode: {mode}. Must be 'metadata-only' or 'read-write'."}
- **401 Unauthorized:** Invalid or missing JWT token.
- **500 Internal Server Error:** {"detail": "Provider configuration missing for the selected mode"} or {"detail": "Failed to initiate authentication"}.

4. GET /api/auth/{provider}/{callback_suffix}-callback

- **Purpose:** Callback endpoint for OAuth 2.0 flow. Handles the authorization code from the cloud provider, exchanges it for tokens, stores account details, triggers initial file sync, and redirects to the frontend dashboard.
- **Controller File:** `backend/main.py`
- **Authentication:** None (handled by OAuth flow, not direct API token)
- **Path Parameters:**
 - **provider:** `string` (e.g., "google_drive", "dropbox", "onedrive")
 - **callback_suffix:** `string` (e.g., "metadata" for metadata-only, "readwrite" for read-write)
(Inferred: This is parsed but the `mode` is primarily from `state`)
- **Query Parameters:**
 - **code:** `string` (authorization code from cloud provider)
 - **state:** `string` (base64 URL-safe encoded JSON containing `user_id`, `mode`, `nonce`)
- **Request Schema:** None (query parameters only)
- **Example Request (from Cloud Provider):**

```
GET /api/auth/google_drive/metadata-callback?  
code=4/0AX...&state=eyJ1c2VyX2lkIjogMSwgIm1vZGUiOiAibWV0YWRhdGEtb25seSIsICJu  
b25jZSI6ICI5enJPelBtM0h4ZW10QXowMlBpbk9WWHBjRV9LaFRkcUJ0VkI4Q0tBalBZIn0%3D
```

- **Response:** **307 Temporary Redirect** to frontend URL.
- **Example Success Redirect:**

```
Location: http://localhost:3000/auth/success?  
token=eyJ...&provider=google_drive&mode=metadata-only
```

- **Example Error Redirect:**

```
Location: http://localhost:3000/auth/error?detail=Error%20message
```

- **Error Codes/Status Codes:**

- Internal server errors or invalid state lead to a redirect to the frontend error page with **detail** parameter.

User & Account Management Endpoints

5. GET /api/health

- **Purpose:** Health check endpoint to determine if the backend service is running.
- **Controller File:** `backend/main.py`
- **Authentication:** None (public endpoint)
- **Request Schema:** None
- **Example Request:**

```
GET /api/health
```

- **Response Schema:**

```
{
  "status": "string",
  "timestamp": "string" (ISO 8601 datetime)
}
```

- **Example Success Response (200 OK):**

```
{
  "status": "healthy",
  "timestamp": "2023-10-27T10:30:00.123456Z"
}
```

- **Error Codes/Status Codes:**

- Service unavailable might result in network error or 5xx status.

6. GET /api/user

- **Purpose:** Retrieve information about the currently authenticated user.
- **Controller File:** `backend/main.py`
- **Authentication:** JWT Token
- **Request Schema:** None
- **Example Request:**

```
GET /api/user
Authorization: Bearer <JWT_TOKEN>
```


- **Response Schema (UserResponse):**

```
{
  "id": 0,
  "email": "string" (EmailStr),
  "name": "string",
  "created_at": "2023-10-27T10:00:00.000Z",
  "storage_accounts_count": 0,
  "total_files_count": 0
}
```

- **Example Success Response (200 OK):**

```
{
  "id": 1,
  "email": "user@example.com",
  "name": "John Doe",
  "created_at": "2023-10-27T10:30:00.123456",
  "storage_accounts_count": 2,
  "total_files_count": 150
}
```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.
- **404 Not Found:** User not found (unlikely after token verification).

7. GET /api/storage-accounts

- **Purpose:** Retrieve a list of all cloud storage accounts connected by the current user.
- **Controller File:** `backend/main.py`
- **Authentication:** JWT Token
- **Request Schema:** None
- **Example Request:**

```
GET /api/storage-accounts
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema (List[StorageAccountResponse]):**

```
[
  {
    "id": 0,
    "provider": "string" (enum: "google_drive", "dropbox", "onedrive"),
    "mode": "string" (enum: "metadata-only", "read-write"),
    "account_email": "string" (EmailStr),
```

```
    "is_active": true,
    "last_sync": "2023-10-27T10:00:00.000Z" (Optional),
    "created_at": "2023-10-27T10:00:00.000Z",
    "files_count": 0
  }
]
```

- **Example Success Response (200 OK):**

```
[
  {
    "id": 101,
    "provider": "google_drive",
    "mode": "read-write",
    "account_email": "my_drive@gmail.com",
    "is_active": true,
    "last_sync": "2023-10-27T10:45:00.000Z",
    "created_at": "2023-10-27T09:00:00.000Z",
    "files_count": 120
  },
  {
    "id": 102,
    "provider": "dropbox",
    "mode": "metadata-only",
    "account_email": "my_dropbox@example.com",
    "is_active": true,
    "last_sync": "2023-10-27T10:50:00.000Z",
    "created_at": "2023-10-27T09:15:00.000Z",
    "files_count": 50
  }
]
```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.

8. **POST /api/storage-accounts/{account_id}/sync**

- **Purpose:** Trigger a background synchronization process for a specific connected storage account.
- **Controller File:** `backend/main.py`
- **Authentication:** JWT Token
- **Path Parameters:**
 - `account_id: integer` (ID of the storage account to sync)
- **Request Schema:** None
- **Example Request:**

```
POST /api/storage-accounts/101/sync
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema:**

```
{
  "message": "string",
  "account_id": 0
}
```

- **Example Success Response (200 OK):**

```
{
  "message": "Sync started",
  "account_id": 101
}
```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.
- **404 Not Found:** {"detail": "Storage account not found"} (if account_id is invalid or doesn't belong to user).
- **500 Internal Server Error:** For unexpected backend errors during task scheduling (less likely).

9. DELETE /api/storage-accounts/{account_id}

- **Purpose:** Delete a connected storage account and all its associated file metadata from the database.
- **Controller File:** backend/main.py
- **Authentication:** JWT Token
- **Path Parameters:**
 - **account_id:** integer (ID of the storage account to delete)
- **Request Schema:** None
- **Example Request:**

```
DELETE /api/storage-accounts/101
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema:**

```
{
  "message": "string"
}
```

- **Example Success Response (200 OK):**

```
{
  "message": "Storage account and all associated files deleted
successfully"
}
```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.
- **404 Not Found:** {"detail": "Storage account not found"}.
- **500 Internal Server Error:** {"detail": "Failed to delete storage account"}.

File Management Endpoints

10. GET /api/files

- **Purpose:** Retrieve a paginated, filterable, and sortable list of file metadata for the current user.
- **Controller File:** backend/main.py
- **Authentication:** JWT Token
- **Query Parameters:**
 - **provider:** string (Optional, filter by provider, e.g., "google_drive")
 - **search:** string (Optional, search by file name substring)
 - **sort_by:** string (Optional, default: "modified_at_source"; e.g., "name", "size", "created_at")
 - **sort_order:** string (Optional, default: "desc"; enum: "asc", "desc")
 - **page:** integer (Optional, default: 1, current page number)
 - **per_page:** integer (Optional, default: 50, number of items per page)
 - **mode:** string (Optional, filter by access mode, e.g., "metadata-only", "read-write")
- **Request Schema:** None
- **Example Request:**

```
GET /api/files?
provider=google_drive&search=report&sort_by=name&sort_order=asc&page=1&per_p
age=20&mode=read-write
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema (FileListResponse):**

```
{
  "files": [
    {
      "id": 0,
      "storage_account_id": 0,
      "provider": "string",
      "provider_file_id": "string",
      "name": "string",
      "path": "string",
      "mime_type": "string",
      "size": 0,

```

```

        "size_formatted": "string",
        "created_at_source": "2023-10-27T10:00:00.000Z",
        "modified_at_source": "2023-10-27T10:00:00.000Z",
        "content_hash": "string" (Optional),
        "preview_link": "string" (Optional),
        "is_active": true
    }
],
"total": 0,
"page": 0,
"per_page": 0,
"total_pages": 0
}

```

- **Example Success Response (200 OK):**

```

{
  "files": [
    {
      "id": 201,
      "storage_account_id": 101,
      "provider": "google_drive",
      "provider_file_id": "gd-file-abc",
      "name": "Project Report.pdf",
      "path": "/Documents/",
      "mime_type": "application/pdf",
      "size": 102400,
      "size_formatted": "100 KB",
      "created_at_source": "2023-09-01T08:00:00Z",
      "modified_at_source": "2023-10-20T14:30:00Z",
      "content_hash": "a1b2c3d4e5f6g7h8",
      "preview_link": null,
      "is_active": true
    }
  ],
  "total": 1,
  "page": 1,
  "per_page": 50,
  "total_pages": 1
}

```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.

11. GET /api/files/{file_id}/preview

- **Purpose:** Get a temporary preview link for a specific file.
- **Controller File:** backend/main.py
- **Authentication:** JWT Token
- **Path Parameters:**

- `file_id: integer` (ID of the file)
- **Request Schema:** None
- **Example Request:**

```
GET /api/files/201/preview
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema:**

```
{
  "preview_url": "string" (URL to the file preview)
}
```

- **Example Success Response (200 OK):**

```
{
  "preview_url": "https://docs.google.com/viewer?url=..."
}
```

- **Error Codes/Status Codes:**
 - `401 Unauthorized`: Invalid or missing JWT token.
 - `403 Forbidden`: {"detail": "File preview not allowed for metadata-only accounts"}.
 - `404 Not Found`: {"detail": "File not found"}.
 - `500 Internal Server Error`: {"detail": "Failed to generate preview link"}.

12. `POST /api/files/upload`

- **Purpose:** Upload a new file to a specified cloud storage provider and associate it with a connected account.
- **Controller File:** `backend/main.py`
- **Authentication:** JWT Token
- **Query Parameters:**
 - `provider: string` (e.g., "google_drive", "dropbox", "onedrive")
 - `folder_path: string` (Optional, default: "/"; path within the cloud drive)
- **Request Schema (multipart/form-data):**
 - `file: UploadFile` (the actual file content)
- **Example Request:**

```
POST /api/files/upload?provider=google_drive&folder_path=/MyFolder
Authorization: Bearer <JWT_TOKEN>
Content-Type: multipart/form-data; boundary=...

--...
```

```
Content-Disposition: form-data; name="file"; filename="document.txt"
Content-Type: text/plain

File content here.
--...--
```

- **Response Schema:**

```
{
  "message": "string",
  "file": {
    "id": 0,
    "storage_account_id": 0,
    "provider": "string",
    "provider_file_id": "string",
    "name": "string",
    "path": "string",
    "mime_type": "string",
    "size": 0,
    "size_formatted": "string",
    "created_at_source": "2023-10-27T10:00:00.000Z",
    "modified_at_source": "2023-10-27T10:00:00.000Z",
    "content_hash": "string" (Optional),
    "preview_link": "string" (Optional),
    "is_active": true
  }
}
```

- **Example Success Response (200 OK):**

```
{
  "message": "File uploaded successfully",
  "file": {
    "id": 202,
    "storage_account_id": 101,
    "provider": "google_drive",
    "provider_file_id": "gd-file-xyz",
    "name": "document.txt",
    "path": "/MyFolder/",
    "mime_type": "text/plain",
    "size": 100,
    "size_formatted": "100 B",
    "created_at_source": "2023-10-27T11:00:00Z",
    "modified_at_source": "2023-10-27T11:00:00Z",
    "content_hash": "h4j5k6l7m8n9o0p1",
    "preview_link": null,
    "is_active": true
  }
}
```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.
- **403 Forbidden:** {"detail": "Uploads not allowed for metadata-only accounts"}.
- **404 Not Found:** {"detail": "Storage account not found"}.
- **500 Internal Server Error:** {"detail": "Failed to upload file"}.

13. DELETE /api/files/{file_id}

- **Purpose:** Delete a specific file from both the cloud storage and the local database.
- **Controller File:** backend/main.py
- **Authentication:** JWT Token
- **Path Parameters:**
 - **file_id:** integer (ID of the file to delete)
- **Request Schema:** None
- **Example Request:**

```
DELETE /api/files/201
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema:**

```
{
  "message": "string"
}
```

- **Example Success Response (200 OK):**

```
{
  "message": "File deleted successfully"
}
```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.
- **403 Forbidden:** {"detail": "Deletion not allowed for metadata-only accounts"}.
- **404 Not Found:** {"detail": "File not found"}.
- **500 Internal Server Error:** {"detail": "Failed to delete file"}.

Duplicate Management Endpoints

14. GET /api/duplicates

- **Purpose:** Retrieve groups of duplicate files (based on content hash) for the current user.
- **Controller File:** backend/main.py

- **Authentication:** JWT Token
- **Query Parameters:**
 - `mode: string` (Optional, filter by access mode, e.g., "metadata-only", "read-write")
- **Request Schema:** None
- **Example Request:**

```
GET /api/duplicates?mode=read-write
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema (`List[DuplicateGroupResponse]`):**

```
[
  {
    "hash": "string" (content hash),
    "count": 0,
    "files": [
      {
        "id": 0,
        "storage_account_id": 0,
        "provider": "string",
        "provider_file_id": "string",
        "name": "string",
        "path": "string",
        "mime_type": "string",
        "size": 0,
        "size_formatted": "string",
        "created_at_source": "2023-10-27T10:00:00.000Z",
        "modified_at_source": "2023-10-27T10:00:00.000Z",
        "content_hash": "string",
        "preview_link": "string" (Optional),
        "is_active": true
      }
    ]
  }
]
```

- **Example Success Response (200 OK):**

```
[
  {
    "hash": "a1b2c3d4e5f6g7h8",
    "count": 2,
    "files": [
      {
        "id": 201,
        "storage_account_id": 101,
        "provider": "google_drive",
        "provider_file_id": "gd-file-abc",

```

```

        "name": "document_v1.pdf",
        "path": "/",
        "mime_type": "application/pdf",
        "size": 102400,
        "size_formatted": "100 KB",
        "created_at_source": "2023-09-01T08:00:00Z",
        "modified_at_source": "2023-10-20T14:30:00Z",
        "content_hash": "a1b2c3d4e5f6g7h8",
        "preview_link": null,
        "is_active": true
    },
    {
        "id": 203,
        "storage_account_id": 102,
        "provider": "dropbox",
        "provider_file_id": "db-file-xyz",
        "name": "document_copy.pdf",
        "path": "/Backups/",
        "mime_type": "application/pdf",
        "size": 102400,
        "size_formatted": "100 KB",
        "created_at_source": "2023-09-05T09:00:00Z",
        "modified_at_source": "2023-10-20T14:30:00Z",
        "content_hash": "a1b2c3d4e5f6g7h8",
        "preview_link": null,
        "is_active": true
    }
]

```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.

15. POST /api/duplicates/remove

- **Purpose:** Delete a list of selected duplicate files from their respective cloud storage providers and the local database.
- **Controller File:** backend/main.py
- **Authentication:** JWT Token
- **Request Schema (RemoveDuplicatesRequest):**

```

{
  "file_ids": [
    0
  ]
}

```

- **Example Request:**

```
{
  "file_ids": [201, 203]
}
```

- **Response Schema:**

```
{
  "message": "string",
  "successful_deletions": [
    0
  ]
}
```

- **Example Success Response (200 OK):**

```
{
  "message": "Selected duplicate files deleted successfully",
  "successful_deletions": [201, 203]
}
```

- **Error Codes/Status Codes:**

- 401 Unauthorized: Invalid or missing JWT token.
- 403 Forbidden: {"detail": "File deletion not allowed for metadata-only account..."}
- 500 Internal Server Error: {"message": "Some files could not be deleted.", "successful_deletions": [...], "failed_deletions": [...], "errors": [...]}.

Statistics Endpoint

16. GET /api/stats

- **Purpose:** Retrieve overall statistics for the current user, including total file count, total storage size, provider-specific stats, and duplicate groups count.
- **Controller File:** backend/main.py
- **Authentication:** JWT Token
- **Request Schema:** None
- **Example Request:**

```
GET /api/stats
Authorization: Bearer <JWT_TOKEN>
```

- **Response Schema (StatsResponse):**

```
{
  "total_files": 0,
  "total_size": 0,
  "total_size_formatted": "string",
  "provider_stats": [
    {
      "provider": "string",
      "file_count": 0,
      "total_size": 0
    }
  ],
  "duplicate_groups": 0
}
```

- **Example Success Response (200 OK):**

```
{
  "total_files": 170,
  "total_size": 1536000000,
  "total_size_formatted": "1.5 GB",
  "provider_stats": [
    {"provider": "google_drive", "file_count": 120, "total_size": 1024000000},
    {"provider": "dropbox", "file_count": 50, "total_size": 512000000}
  ],
  "duplicate_groups": 1
}
```

- **Error Codes/Status Codes:**

- **401 Unauthorized:** Invalid or missing JWT token.

5. Component Interaction

This section illustrates the dynamic interactions between the application's components through sequence and message diagrams, highlighting the synchronous and asynchronous nature of these communications.

5.1. User Login Sequence Diagram

This diagram depicts the synchronous flow when a user attempts to log in.

```
sequenceDiagram
    participant User as User
    participant FE as Frontend
    participant BE as Backend
    participant DB as Database
    participant AUTH as AuthHandler

    User->>FE: Enters credentials (email, password)
```

```

activate FE
FE->>BE: POST /api/login (UserLogin)
activate BE
BE->>DB: Query User by email
activate DB
DB-->>BE: User record
deactivate DB
BE->>AUTH: Verify password
activate AUTH
AUTH-->>BE: Password verification result
deactivate AUTH

alt Successful Login
  BE->>AUTH: Create JWT Token (user_id)
  activate AUTH
  AUTH-->>BE: JWT Access Token
  deactivate AUTH
  BE-->>FE: 200 OK (access_token, user_id, email)
  deactivate BE
  FE->>FE: Store JWT in localStorage
  FE->>FE: Redirect to /modeselection
else Failed Login
  BE-->>FE: 401 Unauthorized (error detail)
  deactivate BE
  FE->>User: Display error message
end
deactivate FE

```

Note right of FE: Interaction: Synchronous, Reliable, Non-idempotent (login attempts increment failed count usually)

Note right of BE: Interaction: Synchronous, Reliable, Non-idempotent (password hashing)

Note right of DB: Interaction: Synchronous, Reliable, Idempotent (read), Non-idempotent (write - if tracking login attempts)

Interaction Characteristics:

- **Type:** Primarily synchronous HTTP requests/responses.
- **Reliability:** High, with immediate feedback to the user on success or failure.
- **Idempotency:** The login request itself is generally *not* idempotent from a security perspective (e.g., repeated failed attempts might trigger rate limiting or account lockout), though the underlying data retrieval might be idempotent.

5.2. Connect Cloud Storage Account (OAuth 2.0) Sequence Diagram

This diagram illustrates the mixed synchronous and redirect-based asynchronous flow for connecting a new cloud storage account.

```

sequenceDiagram
    participant User as User
    participant FE as Frontend

```

participant BE as Backend
participant CP as Cloud Provider OAuth Server
participant DB as Database
participant SP as StorageProvider
participant BG as BackgroundTasks

User->>FE: Selects Provider & Mode (Dashboard)

activate FE

FE->>BE: GET /api/auth/{provider}?mode={mode} (JWT in header)

activate BE

BE->>DB: Verify user (via JWT)

activate DB

DB-->>BE: User details

deactivate DB

BE->>CFG: Get provider config (client_id, redirect_uri, scopes)

activate CFG

CFG-->>BE: Provider config

deactivate CFG

BE->>SP: Get Authorization URL (client_id, redirect_uri, state, scopes)

activate SP

SP-->>BE: OAuth URL

deactivate SP

BE-->>FE: 200 OK (oauth_url)

deactivate BE

FE->>CP: Redirect browser to OAuth URL

deactivate FE

activate CP

CP->>User: Display consent screen

User->>CP: Grants permissions

CP->>BE: GET /api/auth/{provider}/{callback_suffix}-callback?

code=...&state=...

deactivate CP

activate BE

BE->>DB: Decode state, get user & mode

activate DB

DB-->>BE: User details

deactivate DB

BE->>CFG: Get provider config for callback

activate CFG

CFG-->>BE: Provider config

deactivate CFG

BE->>SP: Exchange code for tokens (client_id, secret, redirect_uri, scopes)

activate SP

SP-->>BE: Access/Refresh Tokens

deactivate SP

BE->>SP: Get Cloud User Info

activate SP

SP-->>BE: Cloud user email

deactivate SP

BE->>DB: Find/Create StorageAccount (user, provider, email, mode)

activate DB

DB-->>BE: StorageAccount (updated/new)

deactivate DB

```

BE->>BG: Add sync_account_files task (account_id, db_session, user_id,
security_utils)
activate BG
BG-->>BE: Task added confirmation
deactivate BG
BE->>AUTH: Create JWT for internal user
activate AUTH
AUTH-->>BE: New JWT
deactivate AUTH
BE-->>FE: 307 Redirect to /auth/success?token=...&mode=...
deactivate BE
activate FE
FE->>FE: Store new JWT, extract mode
FE->>FE: Redirect to /dashboard?mode={mode}
deactivate FE

```

Note over FE,BE: Initial auth URL request is synchronous.

Note over CP,BE: OAuth callback is an external-to-internal asynchronous redirect.

Note right of BE: Token exchange and DB operations are synchronous within the callback.

Note right of BG: File sync is an asynchronous, best-effort background task.

Note right of DB: Data storage (StorageAccount) is non-idempotent (creates/updates).

Interaction Characteristics:

- **Type:** A hybrid of synchronous HTTP requests (Frontend-Backend) and asynchronous redirects (Cloud Provider-Backend, Backend-Frontend).
- **Reliability:** High for critical steps (token exchange, DB writes). File sync as a background task is designed for eventual consistency with retry mechanisms (inferred).
- **Idempotency:** The initial `GET /api/auth/{provider}` is idempotent. The `auth_callback` is *not* idempotent as it involves stateful token exchange and database writes (create/update storage account, sync log). However, repeated calls with the same authorization code from the cloud provider should be handled gracefully by the provider and potentially by the application (e.g., token already exchanged).

5.3. File Synchronization Asynchronous Flow

This diagram shows the asynchronous process of synchronizing file metadata from a cloud provider.

```

sequenceDiagram
    participant SyncTrigger as SyncTrigger (Manual/Automatic)
    participant BE as FastAPI Backend
    participant BG as BackgroundTasks (or Celery Worker)
    participant DB as Database
    participant SP as StorageProvider Interface
    participant CP as Cloud Provider API
    participant UT as Utility (Security/File)

    SyncTrigger->>BE: POST /api/storage-accounts/{id}/sync (or internal call from

```

```
auth_callback)
  activate BE
  BE->>BG: Add sync_account_files task (account_id, db_session, user_id,
security_utils)
  BE-->>SyncTrigger: 200 OK (Sync started message)
  deactivate BE

  activate BG
  BG->>DB: Fetch StorageAccount by ID and User ID
  activate DB
  DB-->>BG: StorageAccount details
  deactivate DB
  BG->>UT: Decrypt access/refresh tokens
  activate UT
  UT-->>BG: Decrypted tokens
  deactivate UT
  BG->>DB: Create SyncLog entry (status: running)
  activate DB
  DB-->>BG: SyncLog ID
  deactivate DB
  BG->>DB: Delete existing FileMetadata for account (to prevent orphans)
  activate DB
  DB-->>BG: Deletion confirmed
  deactivate DB
  BG->>SP: Initialize Provider (with decrypted tokens, mode)
  activate SP
  SP-->>BG: Provider instance
  deactivate SP

opt Token Refresh Needed
  BG->>SP: Refresh Access Token
  activate SP
  SP->>CP: POST /oauth2/v4/token (or similar)
  activate CP
  CP-->>SP: New Access Token
  deactivate CP
  SP-->>BG: New Access Token
  deactivate SP
  BG->>UT: Encrypt new Access Token
  activate UT
  UT-->>BG: Encrypted Token
  deactivate UT
  BG->>DB: Update StorageAccount with new token
  activate DB
  DB-->>BG: Update confirmed
  deactivate DB
end

loop until no more pages
  BG->>CP: list_files(page_token, max_results)
  activate CP
  CP-->>BG: File metadata list, next_page_token
  deactivate CP
  BG->>DB: Insert/Update FileMetadata records (per file)
```



```

        activate DB
        DB-->>BG: Records updated
        deactivate DB
    end

    BG->>DB: Deactivate old FileMetadata records (not in current sync)
    activate DB
    DB-->>BG: Old records deactivated
    deactivate DB
    BG->>DB: Update SyncLog (status: completed/failed, stats)
    activate DB
    DB-->>BG: SyncLog updated
    deactivate DB
    deactivate BG

```

Note over BG,CP: Interaction: Asynchronous, Best-effort (retries are implied but not explicit in diagram), Idempotent (re-run effectively recreates state after initial cleanup).

Note over BG,DB: Interaction: Synchronous for individual DB operations, Reliable (transactions used).

Interaction Characteristics:

- **Type:** Asynchronous, initiated by an API call but executed in a background task. This implies a message queue (like Celery/Redis) is used for task distribution (High Confidence: `celery` in `requirements.txt`, `BackgroundTasks` in `main.py`).
- **Reliability:** Best-effort by the API caller, but the background task itself aims for high reliability through error logging and transactional database updates. Potential for retries and dead-letter queues if a robust task queue is used.
- **Idempotency:** The overall `sync_account_files` operation is largely idempotent. By deleting all existing `FileMetadata` for a given `storage_account_id` at the beginning of each sync, a re-run effectively cleans and recreates the file state, leading to a consistent outcome.

6. Data Model & Storage

The application utilizes SQLAlchemy as its ORM to interact with a relational database (SQLite for development, typically PostgreSQL for production). The core data model revolves around users, their connected cloud storage accounts, and the metadata of the files within those accounts.

Database Schemas/Models

1. User Model (`backend/models.py`)

- **Purpose:** Stores user authentication and profile information.
- **Table Name:** `users`
- **Columns:**
 - `id`: `Integer` (Primary Key, Indexed)
 - `email`: `String(120)` (Unique, Not Null, Indexed)
 - `name`: `String(100)`
 - `hashed_password`: `String(128)` (Not Null)

- `created_at`: `DateTime` (Default: `utcnow`)
- `updated_at`: `DateTime` (Default: `utcnow`, OnUpdate: `utcnow`)
- `is_active`: `Boolean` (Default: `True`)
- **Relationships:**
 - `storage_accounts`: One-to-many relationship with `StorageAccount` (cascades all operations, deletes orphans).
 - `files`: One-to-many relationship with `FileMetadata` (cascades all operations, deletes orphans).
 - `sync_logs`: One-to-many relationship with `SyncLog` (cascades all operations, deletes orphans).
 - `sessions`: One-to-many relationship with `UserSession` (cascades all operations, deletes orphans).
- **Key Indexes:** `id`, `email`

2. `StorageAccount` Model (`backend/models.py`)

- **Purpose:** Stores details about a user's connected cloud storage account, including provider-specific tokens and mode of access.
- **Table Name:** `storage_accounts`
- **Columns:**
 - `id`: `Integer` (Primary Key, Indexed)
 - `user_id`: `Integer` (Foreign Key to `users.id`, Not Null)
 - `provider`: `String(50)` (Not Null, e.g., 'google_drive', 'dropbox', 'onedrive')
 - `mode`: `String(20)` (Not Null, Default: 'read-write', e.g., 'metadata-only', 'read-write')
 - `account_email`: `String(120)` (Not Null)
 - `account_name`: `String(200)` (Optional)
 - `access_token`: `Text` (Not Null, Encrypted)
 - `refresh_token`: `Text` (Optional, Encrypted)
 - `token_expires_at`: `DateTime` (Optional)
 - `is_active`: `Boolean` (Default: `True`)
 - `last_sync`: `DateTime` (Optional)
 - `created_at`: `DateTime` (Default: `utcnow`)
 - `updated_at`: `DateTime` (Default: `utcnow`, OnUpdate: `utcnow`)
 - `storage_used`: `BigInteger` (Default: 0)
 - `storage_limit`: `BigInteger` (Default: 0)
- **Relationships:**
 - `user`: Many-to-one relationship with `User`.
 - `files`: One-to-many relationship with `FileMetadata` (cascades all operations, deletes orphans).
 - `sync_logs`: One-to-many relationship with `SyncLog` (cascades all operations, deletes orphans).
- **Key Indexes:** `id`, `idx_user_provider` (`user_id`, `provider`, `mode`), `idx_provider_email` (`provider`, `account_email`)

3. `FileMetadata` Model (`backend/models.py`)

- **Purpose:** Stores metadata for files synchronized from cloud providers. Designed to support metadata-only access without storing actual file content.
- **Table Name:** `file_metadata`
- **Columns:**
 - `id`: `Integer` (Primary Key, Indexed)
 - `user_id`: `Integer` (Foreign Key to `users.id`, Not Null)

- `storage_account_id`: `Integer` (Foreign Key to `storage_accounts.id`, Not Null)
- `provider_file_id`: `String(255)` (Not Null, Unique per account - inferred)
- `name`: `String(500)` (Not Null, Indexed)
- `path`: `Text` (Full path in cloud storage)
- `size`: `BigInteger` (Size in bytes)
- `mime_type`: `String(100)`
- `file_extension`: `String(10)` (Indexed)
- `created_at_source`: `DateTime` (Creation timestamp in cloud storage)
- `modified_at_source`: `DateTime` (Last modified timestamp in cloud storage, Indexed)
- `created_at`: `DateTime` (Default: `utcnow`, record creation in local DB)
- `updated_at`: `DateTime` (Default: `utcnow`, OnUpdate: `utcnow`)
- `preview_link`: `Text` (Optional, URL for file preview)
- `download_link`: `Text` (Optional, direct download URL)
- `web_view_link`: `Text` (Optional, URL for web view)
- `thumbnail_link`: `Text` (Optional, URL for thumbnail preview)
- `content_hash`: `String(64)` (MD5 or SHA-256 hash for content-based duplicate detection, Indexed)
- `size_hash`: `String(64)` (Hash based on size + name for basic duplicate detection, Indexed)
- `is_duplicate`: `Boolean` (Default: `False`)
- `original_file_id`: `Integer` (Foreign Key to `file_metadata.id`, Self-referencing relationship, Optional)
- `is_image`, `is_video`, `is_document`: `Boolean` (Default: `False`, for file type classification)
- `is_active`: `Boolean` (Default: `True`, tracks if file is currently present in cloud)
- **Relationships:**
 - `user`: Many-to-one relationship with `User`.
 - `storage_account`: Many-to-one relationship with `StorageAccount`.
 - `duplicates`: One-to-many self-referencing relationship for duplicate files.
- **Key Indexes:** `id`, `idx_user_files` (`user_id`), `idx_storage_files` (`storage_account_id`), `idx_content_hash` (`content_hash`), `idx_size_hash` (`size_hash`), `idx_provider_file` (`storage_account_id`, `provider_file_id`), `idx_file_name` (`name`), `idx_file_extension` (`file_extension`), `idx_modified_date` (`modified_at_source`)
- **Methods:**
 - `generate_size_hash()`: Generates a hash based on file name, size, and MIME type.
 - `classify_file_type()`: Classifies file into image, video, or document based on MIME type.

4. SyncLog Model (`backend/models.py`)

- **Purpose:** Logs details of file synchronization operations for each storage account.
- **Table Name:** `sync_logs`
- **Columns:**
 - `id`: `Integer` (Primary Key, Indexed)
 - `user_id`: `Integer` (Foreign Key to `users.id`, Not Null)
 - `storage_account_id`: `Integer` (Foreign Key to `storage_accounts.id`, Not Null)
 - `sync_started_at`: `DateTime` (Default: `utcnow`)
 - `sync_completed_at`: `DateTime` (Optional)
 - `files_processed`: `Integer` (Default: 0)
 - `files_added`: `Integer` (Default: 0)

- `files_updated`: `Integer` (Default: 0)
- `files_deleted`: `Integer` (Default: 0)
- `errors_count`: `Integer` (Default: 0)
- `error_details`: `Text` (Optional, stores details if sync failed)
- `status`: `String(20)` (Default: 'running', e.g., 'running', 'completed', 'failed')
- **Relationships:**
 - `user`: Many-to-one relationship with `User`.
 - `storage_account`: Many-to-one relationship with `StorageAccount`.

5. `UserSession` Model (`backend/models.py`)

- **Purpose:** Manages active user sessions, particularly for tracking JWT tokens and associated session details (inferred for potential use, though current `auth.py` directly creates JWTs).
- **Table Name:** `user_sessions`
- **Columns:**
 - `id`: `Integer` (Primary Key, Indexed)
 - `user_id`: `Integer` (Foreign Key to `users.id`, Not Null)
 - `session_token`: `String(255)` (Unique, Not Null, Indexed) - *Note: This might overlap with JWT if not carefully managed; currently JWTs are stateless but this model implies stateful sessions.* (Confidence: Medium, based on name but current JWT implementation is stateless)
 - `expires_at`: `DateTime` (Not Null, Indexed)
 - `created_at`: `DateTime` (Default: `utcnow`)
 - `last_activity`: `DateTime` (Default: `utcnow`)
 - `is_active`: `Boolean` (Default: `True`)
 - `user_agent`: `Text` (Optional)
 - `ip_address`: `String(45)` (Supports IPv6, Optional)
- **Relationships:**
 - `user`: Many-to-one relationship with `User`.
- **Key Indexes:** `id`, `idx_user_sessions` (`user_id`), `idx_session_expires` (`expires_at`)

Migrations

- **Tool:** Alembic is configured (via `alembic.ini`) for database migrations. This ensures schema changes are applied incrementally and versioned.

Sample Queries Found in Codebase (from `backend/main.py`)

- **User Authentication:**

```
db.query(User).filter(User.email == user_create.email).first()
db.query(User).filter(User.id == user_id).first()
```

- **Storage Account Retrieval:**

```
db.query(StorageAccount).filter(StorageAccount.user_id ==
current_user.id).all()
```

```
db.query(StorageAccount).filter(and_(
    StorageAccount.id == account_id,
    StorageAccount.user_id == current_user.id,
    StorageAccount.is_active == True
)).first()
```

- **File Metadata Operations:**

```
db.query(FileMetadata).filter(FileMetadata.user_id ==
current_user.id).count()
db.query(func.sum(FileMetadata.size)).filter(FileMetadata.user_id ==
current_user.id).scalar()
db.query(FileMetadata).filter(
    FileMetadata.storage_account_id == storage_account.id,
    FileMetadata.provider_file_id.notin_(synced_provider_file_ids)
).update({"is_active": False}, synchronize_session=False)
db.query(FileMetadata).filter(
    FileMetadata.storage_account_id == account_id
).delete(synchronize_session=False)
```

- **Duplicate Detection:**

```
base_file_query.group_by(FileMetadata.content_hash).having(func.count(FileMe
tadata.id) > 1).with_entities(FileMetadata.content_hash).subquery()
all_duplicate_files = base_file_query.filter(
    FileMetadata.content_hash.in_(duplicate_hashes_subquery)
).order_by(FileMetadata.content_hash, FileMetadata.name).all()
```

Caching Layers and Invalidation

- **Explicit Caching:** The presence of `redis==5.0.1` in `backend/requirements.txt` suggests that Redis is used for caching. This would typically be for frequently accessed but slow-to-compute data, such as aggregated statistics, or for managing task queues (Celery).
 - **TTL (Time-To-Live):** Not explicitly defined in code, but would be configured when setting/getting keys in Redis (e.g., `cache.set("key", "value", ex=3600)`).
 - **Invalidation:** Cache invalidation strategies are not explicitly visible in `backend/main.py`. Common strategies would include:
 - **Time-based:** Relying on TTLs.
 - **Event-driven:** Invalidation triggered by database writes (e.g., clearing relevant cache entries after a file sync or account update).
 - **On-demand:** Explicitly clearing cache when data is known to be stale.
 - **Evidence:** `backend/requirements.txt` (line 37)
- **Implicit Caching:** Database ORM (SQLAlchemy) itself provides some level of session-level caching for loaded objects, which reduces redundant queries within a single request context.

7. Security & Privacy

The application places a strong emphasis on security and privacy, particularly due to its handling of sensitive cloud storage access and user file metadata. The design principles, as noted in memory [[memory:9088672]], include a strict zero-knowledge architecture, end-to-end encryption, and granular access control.

Authentication Mechanisms

- **User Authentication (Internal):**
 - **Mechanism:** Username (email) and password.
 - **Password Hashing:** `passlib[bcrypt]` is used for strong one-way hashing of user passwords (`hashed_password` in `User` model) before storage. This prevents cleartext password storage and protects against data breaches.
 - **Evidence:** `backend/main.py` (line 109, `auth_handler.hash_password`), `backend/auth.py`.
- **API Authentication (Internal):**
 - **Mechanism:** JSON Web Tokens (JWT).
 - **Token Creation:** `python-jose[cryptography]` is used to create secure, signed JWTs upon successful user login (`create_access_token`). These tokens are short-lived and contain `user_id` as the primary identifying claim.
 - **Token Storage:** The JWT is transmitted to the frontend and stored in `localStorage` (inferred from common frontend practices and `frontend/src/services/api.js` which sets `Authorization` header).
 - **Token Verification:** All protected backend API endpoints (`Depends(get_current_user)`) require a valid JWT in the `Authorization: Bearer <token>` header. `verify_token` function (from `backend/auth.py`) validates the token's signature, expiration, and extracts the `user_id`.
 - **Session Management:** While a `UserSession` model exists in `backend/models.py`, the current JWT implementation appears to be largely stateless. The `UserSession` table *could* be used to implement more granular session invalidation (e.g., revoke all tokens for a user), but this is not explicitly wired up in `backend/main.py`'s `get_current_user` or `login` flows. (Confidence: Medium, potential for future enhancement).
 - **Evidence:** `backend/main.py` (lines 76-93, 134, 315), `backend/auth.py`, `backend/schemas.py` (`Token`, `TokenData`).
- **Cloud Provider Authentication (External):**
 - **Mechanism:** OAuth 2.0 (Authorization Code Grant Flow).
 - **Flow Initiation:** The backend (`GET /api/auth/{provider}`) redirects the user to the cloud provider's authorization server with `client_id`, `redirect_uri`, `scopes`, and a `state` parameter (containing `user_id` and `mode`).
 - **Callback Handling:** The backend (`GET /api/auth/{provider}/{callback_suffix}-callback`) receives the authorization `code` and `state`, then exchanges the code for `access_token` and `refresh_token` with the cloud provider.
 - **Token Storage:** Cloud provider `access_token` and `refresh_token` are stored in the `StorageAccount` model in the database, *encrypted*.
 - **Scope Management:** `PROVIDER_CONFIGS` define specific `scopes` for each provider and access mode (e.g., `drive.metadata.readonly` for Google Drive metadata-only mode), enforcing least privilege.
 - **Evidence:** `backend/main.py` (lines 145-327), `backend/config.py` (`PROVIDER_CONFIGS`), `backend/storage_providers/base.py`, `backend/storage_providers/dropbox.py`, `backend/storage_providers/google_drive.py`, `backend/storage_providers/onedrive.py`.

Authorization Model

- **Role-Based Access Control (RBAC):** Not explicitly implemented with distinct roles (e.g., Admin, User). Authorization is primarily user-centric, ensuring users can only access their own data and connected accounts.
- **Mode-Based Access Control (MBAC):** This is a critical custom authorization layer for privacy. The `mode` (metadata-only or read-write) associated with each `StorageAccount` dictates permissible operations.
 - **Metadata-Only Restrictions:**
 - **Deletion:** `POST /api/duplicates/remove` and `DELETE /api/files/{file_id}` explicitly forbid deletion of files if the associated `StorageAccount` is in "metadata-only" mode. [[memory:9088672]]
 - **Upload:** `POST /api/files/upload` explicitly forbids file uploads for "metadata-only" accounts.
 - **Preview:** `GET /api/files/{file_id}/preview` explicitly forbids generating preview links for "metadata-only" accounts.
 - **Content Access:** Implicitly enforced by not requesting broader scopes during OAuth initiation and by not implementing file content download/access methods in metadata-only mode within `StorageProvider` implementations.
 - **Enforcement Points:** These restrictions are primarily enforced in `backend/main.py` at the API endpoint level and within the `StorageProvider` implementations.
 - **Evidence:** `backend/main.py` (lines 33, 148, 223, 282, 330-340, 635-638, 778-779, 864-865).

Sensitive Data Fields

- `User.hash_password`: Stored as a strong hash, never in plaintext.
- `StorageAccount.access_token`: Encrypted text.
- `StorageAccount.refresh_token`: Encrypted text.
- `UserSession.session_token`: Stored as a hash (inferred, or direct token if stateless JWT not fully adopted for sessions). (Confidence: Medium, as JWT is primary, this field's exact use for sensitive token storage vs. session ID is ambiguous).
- **Environment Variables:** All cloud provider `client_ids`, `client_secrets`, `SECRET_KEY`, and `ENCRYPTION_KEY` are sensitive and loaded from environment variables (e.g., `.env`), not hardcoded.
 - **Evidence:** `backend/models.py`, `backend/config.py`, `backend/utils.py`.

Encryption at Rest and in Transit

- **Encryption in Transit:**
 - All communication between the frontend and backend is over HTTP/HTTPS (implied by `FRONTEND_URL` starting with `http://localhost` for dev, `https://your-domain.com` for prod in CORS settings). Production deployments **must** enforce HTTPS to protect JWTs and API request/response data. (Confidence: High, standard practice).
 - Communication between the backend and cloud providers is over HTTPS (implicit in OAuth and API calls to Google, Dropbox, OneDrive). (Confidence: High, standard practice for these APIs).
 - **Evidence:** CORS settings in `backend/main.py`, cloud provider API documentation (general knowledge).
- **Encryption at Rest:**

- **Cloud Provider Tokens:** `access_token` and `refresh_token` stored in `StorageAccount` model are explicitly encrypted using `SecurityUtils` (which uses `Fernet` from `cryptography` library).
- **Encryption Key Management:** The `ENCRYPTION_KEY` used by `SecurityUtils` is loaded from environment variables, which should be securely managed (e.g., using a secrets manager in production). [[memory:9088672]]
- **Database Encryption:** The database itself is not explicitly noted as encrypted at the filesystem level, but this is a common deployment-level consideration (e.g., encrypted volumes for PostgreSQL). (Confidence: Low, not directly in application code, but critical for a privacy-focused app). [[memory:9088672]]
- **File Content: CRITICAL:** In "metadata-only" mode, file content is **never** accessed or stored by the application backend. In "read-write" mode, while the application *can* access content for operations like de-duplication, it adheres to a zero-knowledge architecture principle by expecting user-controlled end-to-end encryption. This means the application itself does not decrypt or store unencrypted file content. (Confidence: High, stated in memory [[memory:9088672]] and enforced by mode-based logic).
- **Evidence:** `backend/utils.py` (`SecurityUtils`), `backend/models.py` (`StorageAccount.access_token`, `refresh_token`), `backend/main.py` (mode checks).

Common OWASP Risks Spotted & Recommended Mitigations

- **A01:2021-Broken Access Control:**
 - **Risk:** Users could potentially access or manipulate data (files, accounts) that do not belong to them, or perform actions (delete/upload) in metadata-only mode.
 - **Mitigation in Place:** The application extensively uses `user_id` filtering in all database queries and API endpoints (`current_user: User = Depends(get_current_user)`) to ensure object-level access control. Additionally, mode-based checks (`if storage_account.mode == "metadata-only": raise HTTPException(403)`) explicitly forbid sensitive operations in restricted modes.
 - **Recommendation:** Regular security audits and rigorous testing of all authorization logic, especially around mode switching and multi-user environments.
- **A02:2021-Cryptographic Failures (Sensitive Data Exposure):**
 - **Risk:** Sensitive data like cloud provider tokens or user passwords could be exposed in plaintext.
 - **Mitigation in Place:** User passwords are hashed using `bcrypt` (strong, one-way). Cloud provider tokens are encrypted at rest using `Fernet` (AES-256 symmetric encryption). HTTPS is used for transit.
 - **Recommendation:** Implement robust key management for `ENCRYPTION_KEY` (e.g., AWS KMS, Azure Key Vault) in production. Ensure proper rotation of encryption keys and application secrets. Consider using Hardware Security Modules (HSMs) for ultimate key protection.
- **A03:2021-Injection:**
 - **Risk:** SQL Injection via raw SQL queries or OS command injection.
 - **Mitigation in Place:** SQLAlchemy ORM is primarily used for database interactions, which inherently protects against SQL injection by using parameterized queries. FastAPI and Pydantic provide schema validation, reducing the risk of injection through API inputs.
 - **Recommendation:** Avoid raw SQL where possible. If raw SQL is necessary, use parameterized queries with SQLAlchemy's text constructs.
- **A04:2021-Insecure Design:**

- **Risk:** The dual-mode system, if not implemented perfectly, could lead to accidental exposure of file content in metadata-only mode.
- **Mitigation in Place:** Strict logical separation and explicit code checks (`if storage_account.mode == "metadata-only":`) at every sensitive operation point (delete, upload, preview). OAuth scopes are narrowed for metadata-only mode.
- **Recommendation:** Conduct threat modeling specific to the dual-mode logic. Implement automated security tests (e.g., integration tests) to verify mode enforcement. Formal verification of the zero-knowledge claim.
- **A07:2021-Identification and Authentication Failures:**
 - **Risk:** Weak authentication, insecure session management, or flawed JWT handling.
 - **Mitigation in Place:** Strong password hashing, secure JWT generation (signed, short-lived), and explicit token verification for each protected request. CSRF protection using `nonce` in OAuth `state` parameter.
 - **Recommendation:** Implement JWT refresh token rotation. Consider mechanisms for immediate JWT invalidation (e.g., blacklist or revoking `UserSession` entries if it becomes stateful). Implement rate limiting on login attempts to prevent brute-force attacks (WAF/FastAPI middleware). [[memory:9088672]]
- **A09:2021-Security Logging and Monitoring Failures:**
 - **Risk:** Lack of sufficient logging to detect and respond to security incidents.
 - **Mitigation in Place:** Extensive `logger.info` and `logger.error` statements throughout `backend/main.py` provide visibility into user actions, authentication attempts, and errors.
 - **Recommendation:** Enhance logging to include security-relevant events (e.g., failed login attempts, unauthorized access attempts, token refresh failures) with appropriate context. Integrate with a centralized logging solution and SIEM for anomaly detection.

GDPR Compliance Notes

- **Data Minimization:** The "Metadata-Only" mode directly supports data minimization by avoiding collection or processing of actual file content. Only essential file metadata is stored locally. [[memory:9088672]]
- **Purpose Limitation:** Data is collected and processed only for the explicitly stated purpose of providing unified file management and de-duplication services.
- **Storage Limitation:** Data (especially tokens) is stored only as long as necessary (e.g., `token_expires_at` for cloud tokens). User account and file metadata are retained as long as the user maintains an active account.
- **Integrity and Confidentiality:** Achieved through strong encryption for sensitive data at rest and in transit, access control, and password hashing.
- **Consent:** Explicit user consent for connecting cloud accounts is obtained through the OAuth flow, where users grant specific permissions (scopes).
- **Right to Access, Rectification, Erasure:** The application provides API endpoints for users to view their data (`GET /api/user`, `GET /api/storage-accounts`, `GET /api/files`) and delete data (`DELETE /api/storage-accounts/{account_id}`, `DELETE /api/files/{file_id}`). The `User` model can be extended for rectification. (Confidence: High, features are directly supported).
- **Recommendation:** Implement a clear privacy policy accessible to users. Conduct a Data Protection Impact Assessment (DPIA). Ensure mechanisms for data portability are available if required. Regularly review and update compliance measures. [[memory:9088672]]

8. Observability & Testing

This section outlines the current state of observability (logging, metrics, tracing) and testing within the application, along with recommendations for improvement.

CI Pipeline

- **Current State:** No explicit CI/CD pipeline configuration files (e.g., `.github/workflows`, `azure-pipelines.yml`, `gitlab-ci.yml`) are present in the provided file list. (Confidence: High, based on file enumeration).
- **Recommendation:** Implement a robust CI/CD pipeline to automate testing, code quality checks, building, and deployment. Key stages should include:
 - **Linting/Static Analysis:** For both Python (e.g., Black, Flake8, Pylint) and JavaScript (e.g., ESLint).
 - **Unit Tests:** Running tests for backend and frontend components.
 - **Integration Tests:** Testing API endpoints and interactions with external services (mocked).
 - **Security Scans:** SAST (Static Application Security Testing) for code vulnerabilities.
 - **Docker Image Build:** Building the backend Docker image.
 - **Frontend Build:** Building the React frontend static assets.
 - **Deployment:** Automated deployment to staging and production environments.

Test Coverage Areas

- **Current State:** No dedicated test directories (e.g., `tests/`, `test/`) or test files (e.g., `test_*.py`, `*.test.js`) were observed in the provided file list. Therefore, it is assumed that minimal automated testing is currently in place. (Confidence: High, based on file enumeration).
- **Recommendation:** Implement comprehensive testing strategies:
 - **Unit Tests:** Focus on individual functions, methods, and components in isolation (e.g., `AuthHandler` in `backend/auth.py`, utility functions in `backend/utils.py`, individual React components).
 - **Integration Tests:** Verify the correct functioning of API endpoints, database interactions, and interactions with mocked cloud provider APIs (`backend/main.py` endpoints, `backend/storage_providers/*`). This is crucial for verifying mode-based access controls.
 - **End-to-End (E2E) Tests:** Use tools like Cypress or Playwright for the frontend, and potentially API-level E2E tests for critical user flows, simulating real user interactions through the UI and backend.
 - **Security Tests:** Include penetration testing, vulnerability scanning, and fuzz testing.
 - **Performance Tests:** Load testing for API endpoints.

Logging/Tracing/Metrics Integrations

- **Logging:**
 - **Current State:** The backend extensively uses Python's standard `logging` module (`logging.basicConfig`, `logger.info`, `logger.error`) in `backend/main.py` and `backend/storage_providers/*`. `print` statements are also used for debugging. This provides basic application-level logging.
 - **Recommendation:**

- Centralize logging: Integrate with a logging management system (e.g., ELK Stack, Splunk, DataDog, Loki) for aggregation, searching, and analysis of logs across all instances.
 - Structured logging: Output logs in JSON format for easier parsing and querying.
 - Contextual logging: Add more context to log messages (e.g., `request_id`, `user_id`, `account_id`) to aid in debugging distributed systems.
 - Severity levels: Use appropriate logging levels (DEBUG, INFO, WARNING, ERROR, CRITICAL) consistently.
- **Evidence:** `backend/main.py` (lines 10, 69-70, and numerous `logger.info/logger.error` calls).
- **Tracing:**
 - **Current State:** No explicit distributed tracing (e.g., OpenTelemetry, Jaeger, Zipkin) is evident in the codebase. (Confidence: High, based on file content).
 - **Recommendation:** Implement distributed tracing to gain visibility into requests as they flow through different services (frontend, backend, cloud providers, background tasks). This is especially valuable for diagnosing latency and errors in a microservices-oriented or distributed architecture. Libraries like `opentelemetry-python` can be integrated with FastAPI.
- **Metrics:**
 - **Current State:** Basic operational metrics (e.g., file counts, sizes, sync status) are stored in the database (`FileMetadata`, `StorageAccount`, `SyncLog` models) and exposed via the `GET /api/stats` endpoint. No explicit integration with a dedicated metrics collection system (e.g., Prometheus, Grafana, DataDog) is observed.
 - **Recommendation:**
 - Integrate with a metrics collection system to collect and visualize key application and infrastructure metrics (e.g., API request rates, error rates, latency, database query times, CPU/memory usage, background task queue depth).
 - Instrument custom metrics for business-critical operations (e.g., successful OAuth connections, file upload/deletion counts, sync task durations).
 - **Evidence:** `backend/main.py` (`GET /api/stats` endpoint).

Suggested Monitoring Dashboards and Key SLOs

- **Monitoring Dashboards (Examples):**
 - **Application Health:** API availability (e.g., `/api/health` status), request rates, error rates (5xx, 4xx), average latency, backend process health.
 - **User Activity:** Active users, new registrations, login success/failure rates, connected accounts by provider, mode usage distribution.
 - **File Management:** Total files, total storage size, files synced per hour, upload/delete success rates, duplicate file counts, preview link generation success rates.
 - **Background Tasks:** Sync task queue depth, task execution duration, success/failure rates of `sync_account_files`.
 - **Database Performance:** Query latency, connection pool usage, disk I/O, CPU utilization.
 - **Security:** Unauthorized access attempts (401/403 responses), password verification failures, token refresh failures.
- **Key Service Level Objectives (SLOs) (Examples):**
 - **Availability:** API uptime of 99.9% (e.g., `GET /api/health` responds 200 OK).
 - **Latency:** P90 API response time < 200ms for critical endpoints (`/api/login`, `/api/user`, `/api/storage-accounts`). P99 API response time < 500ms.
 - **Error Rate:** API error rate (5xx responses) < 0.1%. Sync task failure rate < 1%.

- **Throughput:** Support X concurrent users/Y requests per second.

9. Performance & Scalability Notes

This section identifies potential performance bottlenecks and outlines strategies for improving scalability across various components of the application.

Bottlenecks

- **Database Hotspots:**
 - Frequent reads/writes to `FileMetadata` table during `sync_account_files` (especially `DELETE` all then `INSERT/UPDATE` can be heavy). This could become a bottleneck with many users or very large numbers of files per user.
 - Complex queries for duplicates (`GET /api/duplicates`) involving `GROUP BY` and `HAVING` clauses can be resource-intensive, especially without optimal indexing or for large datasets.
 - **Evidence:** `backend/main.py` (lines 439-443, 674-713), `backend/models.py` (indexes for `FileMetadata`).
- **Cloud Provider API Rate Limits:**
 - Repeated `list_files` calls during synchronization can hit API rate limits imposed by Google Drive, Dropbox, or OneDrive. This can delay syncs or cause failures.
 - **Evidence:** `backend/storage_providers/*` (implementation of `list_files`).
- **Synchronous Operations in OAuth Callback:**
 - While `sync_account_files` is a background task, the OAuth `auth_callback` endpoint itself performs synchronous database writes and token exchanges. If these operations become slow, they could impact user experience during account connection.
 - **Evidence:** `backend/main.py` (lines 286-309).
- **Frontend Data Fetching:**
 - Fetching large lists of files or duplicates (`GET /api/files`, `GET /api/duplicates`) might be slow if the backend query is inefficient or if too much data is transferred, impacting frontend responsiveness.
 - **Evidence:** `frontend/src/pages/Files.js`, `frontend/src/pages/Duplicates.js`.

Recommended Improvements and Scaling Strategy

- **Database Optimization:**
 - **Indexing:** Ensure all frequently queried columns have appropriate indexes. `backend/models.py` already includes several, but review and add more if query analysis shows bottlenecks.
 - **Batch Operations:** For `sync_account_files`, consider batching `INSERT` and `UPDATE` operations for `FileMetadata` rather than individual commits, or use `bulk_insert_mappings` for efficiency. The current approach of `DELETE` all then re-`INSERT/UPDATE` can be optimized with upsert logic or by marking existing files inactive/active more efficiently without full deletion.
 - **Database Sharding/Clustering:** For very high scale, distribute data across multiple database instances based on `user_id` or `storage_account_id`.
 - **Read Replicas:** Use read replicas for read-heavy operations (e.g., `GET /api/files`, `GET /api/duplicates`, `GET /api/stats`) to offload the primary database.
- **Backend Scaling (FastAPI):**

- **Horizontal Scaling:** Run multiple instances of the FastAPI application behind a load balancer. This can be achieved with Docker and Kubernetes.
- **Asynchronous Processing:** Leverage FastAPI's `BackgroundTasks` (as currently used) and a dedicated task queue (Celery with Redis/RabbitMQ) for all long-running or non-critical operations (file sync, heavy data processing). This frees up the main API workers to handle immediate requests.
- **Gunicorn Workers:** Configure Gunicorn (present in `requirements.txt`) with an optimal number of worker processes to fully utilize CPU cores.
- **Cloud Provider Integration:**
 - **Rate Limit Handling:** Implement robust retry logic with exponential backoff for all cloud provider API calls to gracefully handle rate limits and transient errors.
 - **Webhooks/Change Notifications (Inferred/Future):** Instead of polling (like `list_files`), consider using webhooks or change notifications from cloud providers to trigger syncs only when changes occur, reducing API calls and improving real-time updates. (Confidence: Low, not currently implemented).
- **Frontend Performance:**
 - **Pagination & Virtualization:** Ensure efficient pagination is implemented for large lists. For very long lists, use UI virtualization libraries to render only visible items.
 - **Client-Side Caching:** Cache API responses on the client side (e.g., using `react-query` or similar) to reduce redundant network requests.
 - **Bundle Optimization:** Optimize JavaScript bundle size, lazy load components, and use image optimization techniques.
- **Caching Strategy:**
 - Utilize Redis (as suggested by `requirements.txt`) for caching frequently accessed aggregate data (e.g., `GET /api/stats` results, dashboard summaries) to reduce database load. Implement appropriate TTLs and invalidation strategies.
 - **Evidence:** `backend/requirements.txt` (line 37).

10. Appendix: File Map

A table of every application source file examined, its primary role, and any noteworthy elements.

File Path	Role / Purpose	Noteworthy Lines / Elements
<code>backend/main.py</code>	Main FastAPI application, API endpoints, orchestration	<code>app = FastAPI(...),</code> <code>@app.get(...),</code> <code>sync_account_files</code>
<code>backend/config.py</code>	Application settings, cloud provider configurations	<code>settings,</code> <code>PROVIDER_CONFIGS</code>

File Path	Role / Purpose	Noteworthy Lines / Elements
backend/database.py	Database connection, session management, SQLAlchemy base	engine, SessionLocal, Base, get_db
backend/models.py	SQLAlchemy ORM models (User, StorageAccount, FileMetadata, SyncLog, UserSession)	User, StorageAccount, FileMetadata, SyncLog, UserSession
backend/schemas.py	Pydantic data models for API requests/responses	UserCreate, UserResponse, FileResponse, DuplicateGroupResponse
backend/auth.py	User authentication, password hashing, JWT handling	AuthHandler, create_access_token, verify_token
backend/storage_providers/__init__.py	Initializes storage provider module	SUPPORTED_PROVIDERS, get_provider_class
backend/storage_providers/base.py	Abstract base class for cloud storage providers	BaseProvider, exchange_code_for_token, list_files
backend/storage_providers/dropbox.py	Dropbox API implementation	DropboxProvider, list_files, delete_file, upload_file
backend/storage_providers/google_drive.py	Google Drive API implementation	GoogleDriveProvider, list_files, delete_file, upload_file
backend/storage_providers/onedrive.py	OneDrive API implementation	OneDriveProvider, list_files, delete_file, upload_file
backend/utils.py	Utility functions (security, file processing)	SecurityUtils, FileUtils, encrypt_token, format_file_size
backend/app.py	(Unclear purpose, potentially unused or alternative entry)	(Empty/minimal content - inferred)

File Path	Role / Purpose	Noteworthy Lines / Elements
frontend/src/App.js	Main React component, frontend routing	App, BrowserRouter, Route
frontend/src/index.js	React app entry point, global providers	ReactDOM.createRoot, ThemeProvider
frontend/src/config.js	Frontend configuration (API Base URL)	API_BASE_URL
frontend/src/services/api.js	Centralized API calls, Axios instance, JWT handling	axios, setAuthToken
frontend/src/pages/Landing.js	Public landing page	Landing component
frontend/src/pages/Login.js	User login page	Login component, POST /api/login call
frontend/src/pages/Register.js	User registration page	Register component, POST /api/register call
frontend/src/pages/ModeSelection.js	Mode selection page (Metadata-Only / Read-Write)	ModeSelection component
frontend/src/pages/AuthCallback.js	OAuth callback handler, JWT storage, redirection	AuthCallback component, localStorage.setItem
frontend/src/pages/Dashboard.js	Dashboard, account overview, sync triggers	Dashboard component, GET /api/storage-accounts
frontend/src/pages/Files.js	File listing, search, upload, delete	Files component, GET /api/files, POST /api/files/upload
frontend/src/pages/Duplicates.js	Duplicate file display and removal	Duplicates component, GET /api/duplicates
frontend/src/pages/Settings.js	User settings page	Settings component (placeholder)
frontend/src/components/Layout.js	Application layout, navigation	Layout component, useNavigate

File Path	Role / Purpose	Noteworthy Lines / Elements
frontend/src/components/PrivateRoute.js	Route protection for authenticated users	PrivateRoute component, localStorage.getItem
frontend/src/components/FileIcon.js	File type icon display	FileIcon component, react-icons/fa
frontend/src/index.css	Global CSS styles	CSS declarations
frontend/src/App.css	App-specific CSS styles	CSS declarations
requirements.txt	Python backend dependencies	List of Python packages and versions
package.json	Node.js frontend dependencies and scripts	List of Node.js packages and versions, start, build scripts
alembic.ini	Alembic database migration configuration	Database connection settings, script location
.env (example)	Environment variables (secrets, configurations)	DATABASE_URL, SECRET_KEY, GOOGLE_DRIVE_CLIENT_ID
Dockerfile	Dockerfile for containerizing backend	FROM python:3.9-slim-buster, COPY requirements.txt
README.md (root)	Project overview and setup instructions	Project description, usage guide
docs/README.md	Placeholder for documentation	Empty file (currently)
docs/research-architecture.md	Comprehensive architecture documentation (this file)	Markdown content of this document
docs/api-manifest.json	Machine-readable API endpoint manifest	JSON array of API endpoint details

11. Assumptions & Unknowns

This section outlines assumptions made during the analysis due to incomplete information or areas where further investigation would be beneficial.

- **Celery Configuration and Usage:** While `celery` is present in `backend/requirements.txt` and `celery.exe` exists in `venv/Scripts`, explicit Celery worker configuration (e.g., `celery.py` with app instance, task definitions beyond `BackgroundTasks`) and message broker setup (Redis/RabbitMQ connection details) were not found in the directly analyzed source files. It is assumed that Celery is intended for more robust background task processing in a production environment, extending beyond FastAPI's `BackgroundTasks`.
 - **Ambiguity:** Exact Celery task definitions, broker configuration, and error handling within Celery workers. How `sync_account_files` would be integrated as a formal Celery task.
 - **Confidence:** Medium (inferred usage).
- **UserSession Model Implementation:** The `UserSession` model in `backend/models.py` suggests a more stateful session management approach. However, the current API authentication primarily relies on stateless JWTs. It's unclear if `UserSession` is actively used to track active sessions or for JWT revocation/blacklisting.
 - **Ambiguity:** The exact interaction between JWTs and `UserSession` for session management and invalidation. Whether `session_token` stores raw JWT or a derived session ID.
 - **Confidence:** Medium.
- **Database Type in Production:** While SQLite is likely used for local development, PostgreSQL is inferred for production based on common practices for scalable web applications. The `DATABASE_URL` in `.env` is expected to facilitate this.
 - **Ambiguity:** Specific production database configuration details.
 - **Confidence:** High (common inference).
- **Frontend Error Handling:** General client-side error handling (beyond displaying messages for specific API errors) in the React application is inferred to be present but not detailed in the provided files.
 - **Ambiguity:** Global error boundaries, toast notifications, or specific retry mechanisms on the frontend.
 - **Confidence:** Low (general frontend best practice).
- **Full `backend/app.py` Content:** The `backend/app.py` file was identified but its content was not provided in the search results (likely empty or a minimal placeholder). Its role is assumed to be either a vestige or an alternative, currently unused, application entry point.
 - **Ambiguity:** The actual purpose and content of `backend/app.py`.
 - **Confidence:** Low.
- **Real-time Updates/Webhooks:** The architecture does not explicitly show mechanisms for real-time updates from cloud providers (e.g., webhooks triggering syncs). Syncs are primarily pull-based (manual trigger or periodic background tasks). This could be an area for future enhancement.
 - **Ambiguity:** Whether push notifications from cloud providers are considered.
 - **Confidence:** High (based on lack of explicit implementation).
- **Detailed Frontend State Management:** While React Hooks (`useState`, `useEffect`, `useContext`) are used, a comprehensive global state management solution (e.g., Redux, Zustand, Recoil) is not explicitly observed in the provided frontend files for complex cross-component state. It is assumed context API or local state is primarily used.
 - **Ambiguity:** Specific patterns for managing application-wide state.
 - **Confidence:** Medium.
- **File Content Hashing Implementation:** While `content_hash` is present in `FileMetadata` for duplicate detection, the exact process of generating this hash for potentially very large files (especially

in **read-write** mode) and how it handles different cloud provider APIs for content access is not fully detailed. It's assumed that the **StorageProvider** implementations fetch content (or a hash equivalent) when in read-write mode.

- **Ambiguity:** How content hashing is performed for different cloud providers, especially for large files, and its performance implications.
- **Confidence:** Medium.

</rewritten_file>