Introduction

The exponential growth of digital information has significantly increased the need for reliable and scalable cloud storage solutions. Users—ranging from students to professionals and small organizations require secure, accessible, and user-friendly platforms to store and share files efficiently. Traditional storage services can often present limitations such as restricted customization, complex integration with other systems, or high operational costs for small-scale users.

To address these challenges, this project proposes the design of a cloud-based file storage platform that enables users to upload, organize, and manage digital resources through an intuitive web interface. The system integrates authentication, access control, and metadata management to ensure information security and efficient retrieval. Moreover, it leverages cloud technologies such as object storage services and RESTful APIs to provide scalability and reliability.

This document focuses on the refinement and technical expansion of the design elements, together, these components establish the foundation for the database and system architecture design.

1 Business Model Canvas

1.1 Desing Business Model

The following model provides a structured framework to define the key aspects of the proposed file storage platform. Identifies the value proposition, customer segments, partners, resources, and revenue streams, providing a clear overview of how the system creates and delivers value to its users.

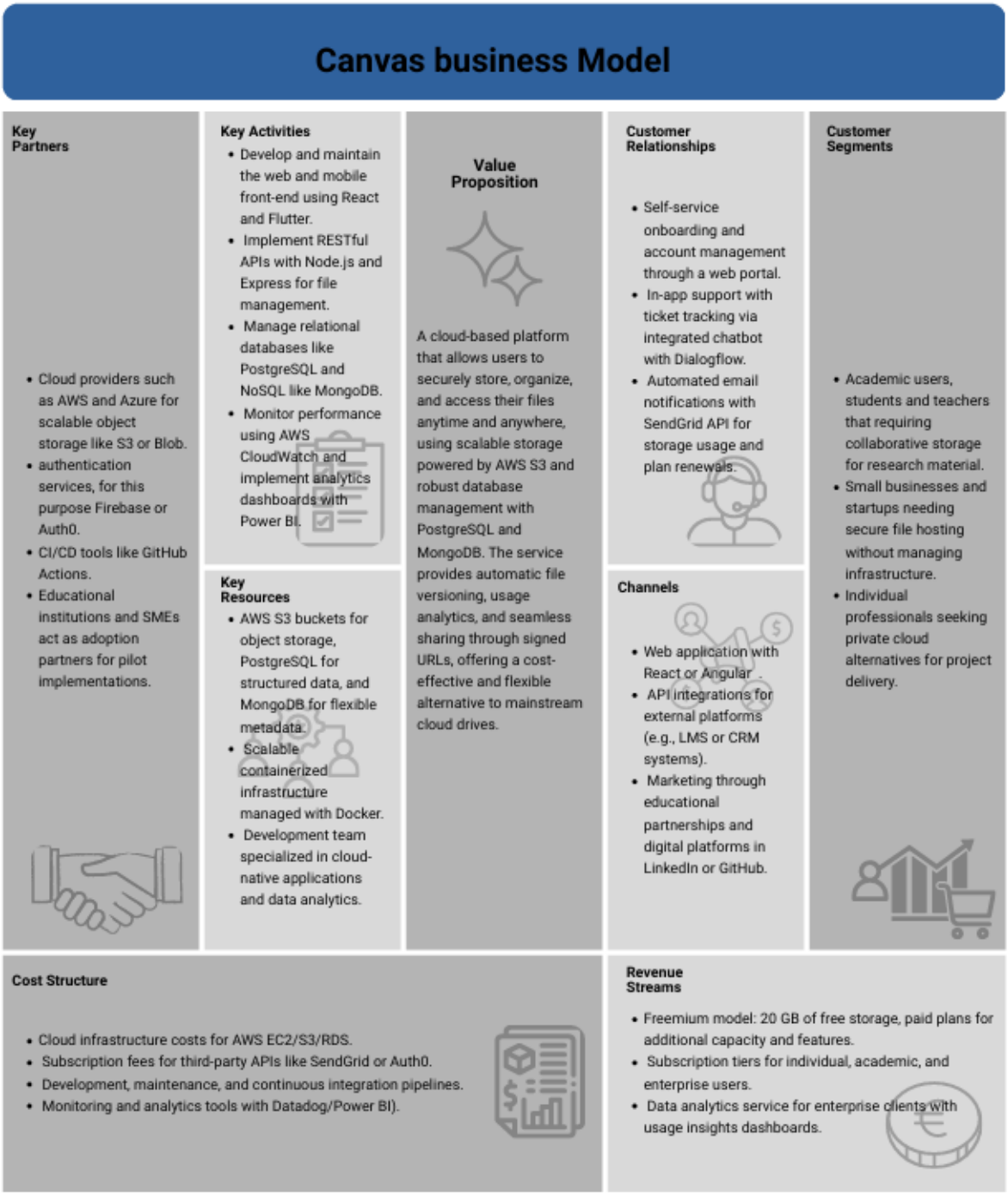

Figure 1: Business Model Canvas for the File Storage Platform.

2 Requirements Documentation

2.1 Functional Requirements

- FR1. Allow users to register with their details and relevant information (email address, personal details, and other details to be agreed upon).
- FR2. Allow already registered users to log in, taking into account their log-in credentials (email and password).
- FR3. Allow users to manage their storage space using folders, which must have specific options (view, organize, delete, move). Similarly, it must be possible to navigate between folders by hierarchical level according to the user's organization, listing the file elements of various permitted extensions, described specifically in 'RF4', which are classified in each of the folder layers.
- FR4. FR4. Users must be able to upload files to their workspace, organizing and separating items as they wish into folders. Possible events during file upload must be guaranteed, indicating whether the files have been uploaded correctly or whether errors have occurred during upload. Valid file types for user upload are (.docx, .pdf, .xls, .mp4, .mp3, .png, .jpg, .gif, .pptx) and the maximum upload size per file is 20 MB.
- FR5. Users must be able to view the current capacity of their storage space, detailing how much space is currently occupied and how much space remains available for use. It should be noted that this storage space depends directly on the user's plan.
- FR6. Allow the user to view important data about the files they have in their storage space, including file name, size, root or file type, and upload date. In addition, users should also be allowed to download files available in their storage space. However, Allow users to manage their files (delete, move, hide, and organize).
- FR7. Users shall have access only to files belonging to them and shall not have access to or be able to view items from other existing accounts.
- FR8. Users must have the ability to securely recover passwords (email with temporary token). This allows users to reset their passwords securely and reliably.
- FR9. Users may access a premium plan by making a fixed payment to purchase membership, which will have a predetermined value and will be valid for only six months. Users may have only one active premium membership, which allows them to expand the storage available on normal free user accounts

2.2 Non-Functional Requirements

Non-functional requirements establish the quality attributes and operational constraints of the file storage platform, each requirement has been refined to ensure technical feasibility and measurable implementation within the project scope.

- **NFR1. Security.** The system must ensure user authentication and data protection. Passwords must be stored securely using hash-based encryption, e.g., SHA-256, and all data transfers like login, upload, download must occur over HTTPS. Access control mechanisms will restrict unauthorized access to user content.

- **NFR2. Scalability.** The platform should be designed to support a moderate increase in users and files without significant performance loss. The implementation will rely on cloud storage services e.g., AWS S3 that allow horizontal scaling when necessary.

- **NFR3. Availability.** The prototype must maintain stable operation during normal usage and testing sessions. Full 24/7 availability is not required at this stage, but the system should be resilient to basic interruptions and restart gracefully.

- **NFR4. Performance.** Core functions such as login, file upload, and download should respond within acceptable limits, thinkink at least in seconds for standard files. Database queries and file retrieval will be optimized for concurrent user requests.

- **NFR5. Usability.** The user interface must be simple, intuitive, and accessible on desktop and mobile browsers. Navigation and file management tasks should be easily performed without user training.

- **NFR6. Maintainability.** The system architecture must follow a modular design, allowing future updates and debugging with minimal effort. Code documentation and naming conventions will ensure ease of maintenance.

- **NFR7. Compatibility.** The web application must operate correctly across major browsers like Chrome, Firefox or Edge. Responsive design must ensure usability on mobile devices.

- **NFR8. Reliability.** The system should handle minor failures without losing user data. Uploads and downloads must include confirmation and error-handling mechanisms. Advanced redundancy is beyond the current scope but will be considered for future iterations.

- **NFR9. Backup and Recovery.** Basic backup mechanisms must be implemented using cloud storage versioning or scheduled export of metadata. Recovery from file deletion or corruption should be possible during testing.

- **NFR10. Logging and Auditability.** The platform must record relevant user operations, login, upload, download and delete in an event log to support monitoring and debugging. Logs will be stored securely and reviewed during system evaluation.

- **NFR11. Interoperability.** The platform must provide RESTful APIs that enable future integration with external applications, such as institutional systems or educational platforms. Initial endpoints will focus on authentication and file access.

3 User Stories

Title	Priority	Estimate
UH-01: User Registration	High	8 SP
As a new user, I want to register for an account by providing my email, password, and personal details, so that I can have my own private storage space on the platform.		
Given a new user on the registration page, when the user enters a valid email, a secure password, and requires personal details and submit the form, then the account is created, and the user receive an email confirmation.		

Title	Priority	Estimate
UH-02: User Login	High	3 SP
As a registered user, I want to log in using my email and password, so that I can securely access my files and storage space.		
Given a registered user on the login page, when the user enter a correct email and password and click "Login", then the user is authenticated and redirected to its personal dashboard.		

Title	Priority	Estimate
UH-03: Secure Password Recovery	Normal	5 SP
As a user who forgot my password, I want to request a password reset via email with a temporary token, so that I can regain access to my account without compromising security.		
Given the user is on the login page and click "Forgot Password", when the user enter its email address and submit the request, then the user receive an email with a secure, temporary link to create a new password.		

Title	Priority	Estimate
UH-04: Upload a File	Must-Have	8 SP
As a logged-in user, I want to upload a file (e.g., .pdf, .jpg, .mp3) up to 100MB to a specific folder, so that I can store and organize my content in the cloud.		
Given the user is in its workspace and have navigated to the desired folder, when the user drag-and-drop or select a valid file for upload, then the system uploads the file and displays a success message; if there is an error, it clearly informs to the user		

Title	Priority	Estimate
UH-05: Create and Navigate Folders	Normal	5 SP
As a user, I want to create new folders and navigate through my folder hierarchy, so that I can organize my files logically and find them easily.		
Given the user is viewing the contents of its current folder, when the user click "New Folder", provide a name, and confirm, then the new folder appears in the current view, and the user can click on it to navigate inside.		

Title	Priority	Estimate
UH-06: View File List and Details	Normal	5 SP
As a user, I want to see a list of my files with their name, size, type, and upload date, so that I can get an overview of my stored content and its properties.		
Given the user is logged in and viewing a folder, when the page loads, then the user see a list of all items in that folder, displayed in a table or grid with the relevant columns.		

Title	Priority	Estimate
UH-07: Download a File	Normal	3 SP
As a user, I want to download a file from my storage space, so that I can access a local copy on my device.		
Given the user is viewing the list of its files, when the user click the "Download" button next to a file, then the file download begins immediately via a secure, temporary URL.		

Title	Priority	Estimate
UH-08: Delete a File or Folder	Normal	5 SP
As a user, I want to delete files or folders I no longer need, so that I can free up storage space and keep my workspace tidy.		
Given the user have selthe user click the "Delete" button and confirm the action, then the items are moved to a "Trash" (or permanently deleted with a warning), and the storage space is updated.		

Title	Priority	Estimate
UH-09: View Storage Capacity	High	8 SP
As a user, I want to see a visual indicator of my used and available storage space, so that I can manage my uploads and avoid running out of space.		
Given the user is logged into its dashboard, when the page loads, then the user see a clear display (e.g., a bar chart or text) showing "X MB used of Y MB available".		

Title	**Priority**	**Estimate**
UH-10: Move Files/Folders	Normal	8 SP
As a user, I want to move files and folders from one location to another within my storage, so that I can reorganize my content without having to re-upload it.		
Given the user have selected a file or folder, when the user select the "Move" action and choose a destination folder from its hierarchy, then the item is moved to the new location and removed from the original one.		

Title	**Priority**	**Estimate**
UH-11: Session Control and Security	Very High	5 SP
As a system security manager, I want all file accesses to be validated through session control, so that unauthorized users cannot access my files.		
Given a user is not logged in or their session has expired, when they try to access a direct file URL or an API endpoint, then the system denies the request and redirects them to the login page.		

Title	**Priority**	**Estimate**
UH-12: File Activity History	Normal	13 SP
As a user, I want to view a history of my recent actions (uploads, deletions, moves), so that I can track changes and recover from mistakes.		
Given the user is on its dashboard, when the user navigate to an "Activity" or "History" section, then user see a chronological list of its file operations with timestamps.		

Title	**Priority**	**Estimate**
UH-13: API for Third-Party Integration	Very High	13 SP
As a developer or power user, I want to use a well-documented API to interact with my files, so that I can integrate the storage service with other applications (e.g., an LMS).		
Given the user have a valid API token, when user send a GET request to the '/api/files' endpoint, then user receive a JSON list of the files in its root directory.		

4 Initial Database Architecture

4.1 High-Level Architecture Proposal

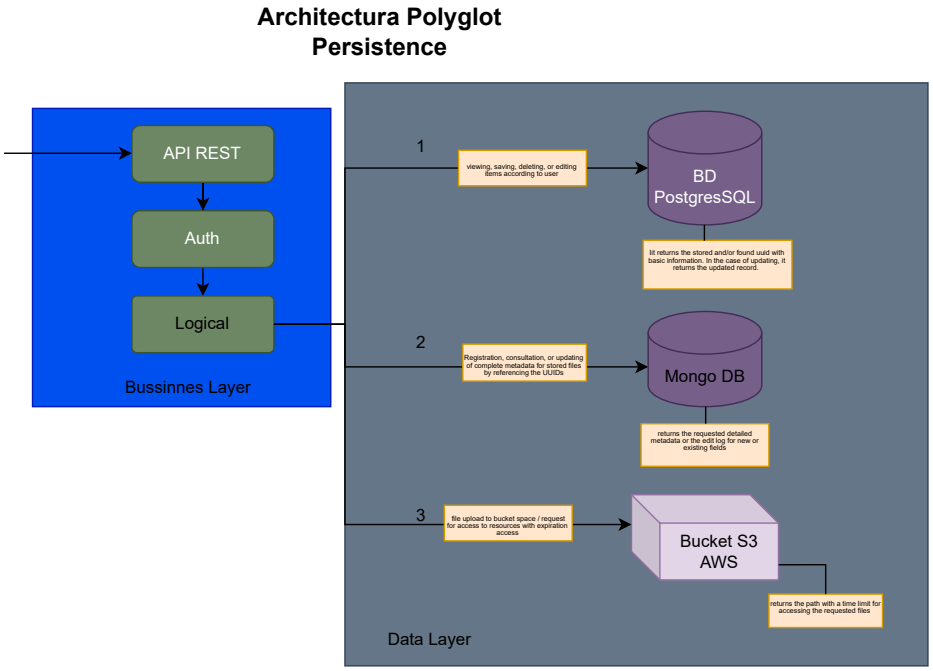

Figure 2: Architecture High Level Model for the File Storage Platform.

4.2 Entity Relationship Diagram - Second Version

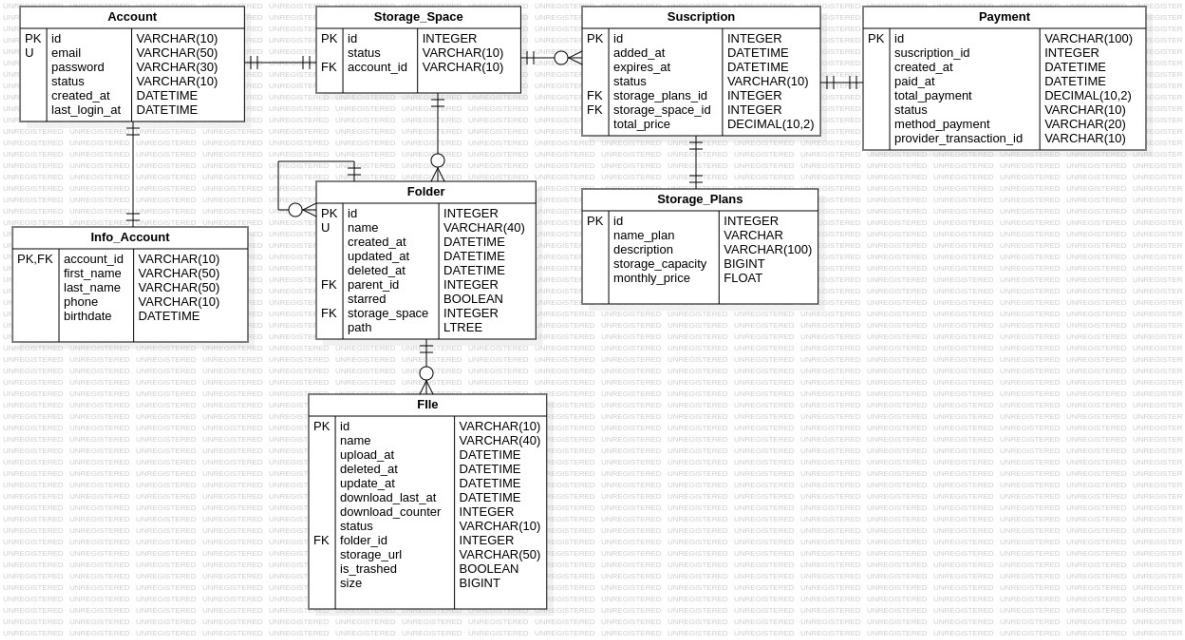

Figure 3: First version of the entity relationship diagram for the file storage platform.

NOTE: Considering the feedback on the need for deeper analysis and improved data storage design, it was decided to remove the metadata information from the relational model. All metadata variations will instead be managed within an unstructured data model. Therefore, the ER diagram includes only a superficial reference to the files, while detailed metadata will be stored at scale in the non-relational database, "MongoDB." Furthermore, the use of the *account* → *info_account* relationship is justified by defining a mandatory one-to-one association, where each account must have exactly one corresponding account information record. Consequently, creating an account information record requires the presence of an existing account ID, as its primary key is a composite key that includes the account ID.

4.2.1 Description of entities

- The **Account** entity represents the user account within the system. It contains main fields such as *id*, *email*, *password*, *status*, *created_at*, and *last_access*. Its role is to be the starting point for all user information, since both personal settings and storage space are derived from it.
- **Info_Account** complements the account with personal data: *first_name*, *last_name*, *phone*, and *birthdate*. It is directly associated with Account and serves to store user identification information.
- The **Storage_Space** entity manages the storage space allocated to each account. It includes *id*, *status*, and a foreign key *account_id*. Its role is to serve as a link between the user and their storage subscriptions.

- **Subscription** stores the data for an active subscription: *id*, *added_at*, *expires_at*, *status*, *total_price*, along with references to the contracted plan and storage space. Its function is to record the conditions of use of the service for a given period.
- The **Payment** entity represents the payments made for a subscription. It contains *id*, *created_at*, *paid_at*, *total_payment*, and *method_payment*. It allows you to keep track of the charges associated with each service contract.
- The **Folder** entity organizes files into hierarchies. Its most important fields are *id*, *name*, *created_at*, *updated_at*, *deleted_at*, *parent_id*, and *starred*. It represents directories that can contain files and subfolders.
- **File** stores user file information. It includes *id*, *name*, *created_at*, *deleted_at*, *download_counter*, *size*, and *folder_id*. It is responsible for representing the digital content within the folders.
- The **Format_File** entity describes the file format. It contains *name*, *description*, and *extension*. Its function is to identify the file type and its basic characteristics.
- Finally, **Metadata** complements the file with specific information such as *mime_type*, *title*, *author*, *storage_backend*, and *complete_metadata*. It serves to expand the technical and descriptive details of each file.

4.2.2 Description of Relationships between Entities

- Account is related to Info_Account, as each account has associated personal information.
- Account is linked to Storage_Space, indicating the storage space allocated to each user.
- Storage_Space is connected to Subscription, which defines the terms of the contracted service.
- Subscription is associated with Payment, which allows the payments for each subscription to be recorded.
- Folder is organized hierarchically by its parent_id, and in turn contains multiple Files.
- File is related to Format_File, to define the file type, and to Metadata, which expands its descriptive information.

4.3 Data Flow and Storage Solutions

The data architecture of the proposed file storage system focuses on how information is generated, transferred, and stored throughout the platform. The goal is to ensure secure handling of user data and efficient management of file storage while maintaining scalability and reliability. Figure 4 illustrates the overall information flow.

4.3.1 Data Flow

When a user performs an operation such as registration, login, or file upload, the system validates the request and processes the corresponding data transaction. During authentication, user credentials are verified against the database, and secure session identifiers are generated. In file upload operations, the file's binary content and descriptive metadata are separated:

- The **binary content** is stored directly in a cloud-based object storage service.
- The **metadata** name, size, type, upload date, and owner is recorded in the relational database.

For retrieval or download requests, metadata is used to locate the file in storage, and a temporary access token or signed URL ensures that only authorized users can retrieve it. All user interactions—such as upload, deletion, or access—are recorded in activity logs that support traceability and auditing.

This structure ensures a clear distinction between operational data, transactions and sessions and persistent data, files and metadata, simplifying future scalability and maintenance.

4.3.2 Storage Solutions

The system uses a hybrid data storage model that separates structured and unstructured data for efficiency and flexibility.

- **Relational Database PostgreSQL.** Stores user information, permissions, and file metadata. The relational model ensures consistency, supports indexing, and facilitates query optimization.
- **Object Storage S3-compatible.** Handles unstructured binary data, storing files using unique identifiers linked to metadata entries. This provides scalable and cost-effective storage for large files.
- **Cache Redis, optional.** Temporarily stores frequently accessed metadata to improve query response times and reduce database load.
- **Logging and Backup.** The system maintains logs for user actions and performs periodic backups of metadata and configuration data to support recoverability and auditing.

This architecture allows for secure, modular, and efficient data handling while remaining lightweight and scalable for a prototype implementation.

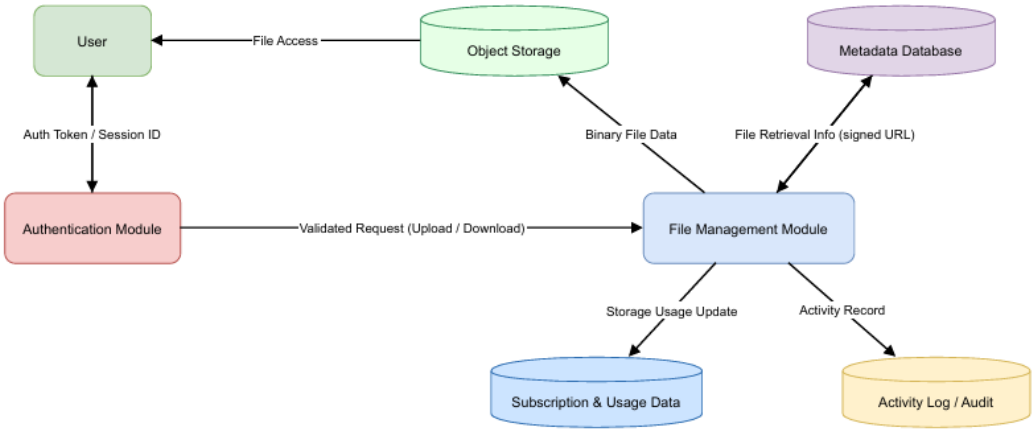

Figure 4: Data Flow and Storage Logic of the file storage platform.

The proposed data flow and storage logic establish a clear separation between user interaction data, operational transactions, and persistent storage. By isolating metadata management from binary file handling, the system ensures scalability, integrity, and efficient retrieval of information. The hybrid storage model—combining relational databases and object storage—provides flexibility for future growth while maintaining a lightweight implementation suitable for a prototype environment.

5 Information Requirements

This section identifies and describes the main types of information for the file storage platform that must retrieve and manage to support its business processes and user interactions.

5.1 Main Information Types

- **User Account Information.** Includes essential user details such as email, password, registration date, and status. This information supports authentication and user management processes. It relates to user stories *UH-01 User Registration*, *UH-02 User Login*, and *UH-03 Secure Password Recovery*.

- **Personal Profile Data.** Contains user-identifying information. Name, last name, birthdate, contact data. This data improves personalization and communication features, aligning with the *Customer Relationship* block in the Business Model.

- **File Metadata.** Describes the properties of each stored file: name, size, format, upload date, and folder location. It enables browsing, searching, and organizing functionalities. Directly related to user stories *UH-04 Upload a File*, *UH-05 Create and Navigate Folders*, and *UH-06 View File List and Details*.

- **Storage Usage Data.** Includes information about total and available space per user and plan type. This supports the system's scalability control and provides feedback to users about their consumption levels, as seen in *UH-09 View Storage Capacity*.

- **Access Control and Permissions.** Defines which user has access to each file or folder for view, edit or share. Ensures security and privacy according to user roles and authentication tokens. This type of data supports *UH-07 Download a File*, *UH-08 Delete a File or Folder*, and *UH-11 Session Control and Security*.

- **Subscription and Payment Records.** Stores plan details, payment methods, and subscription validity periods. It enables the implementation of the freemium model defined in the *Revenue Streams* block and supports future billing automation. It relates to user story *UH-09 View Storage Capacity* and the business requirement of premium upgrades.

- **Activity Logs.** Logs all relevant system events, including uploads, downloads, deletions, and failed access attempts. This information supports auditing, reliability, and fault diagnosis, as defined in non-functional requirements *NFR10 Logging and Auditability* and user story *UH-12 File Activity History*.

- **API Integration Data.** Contains the tokens, permissions, and endpoints associated with third-party system access. It supports the integration described in the *Interoperability* requirement and relates to user story *UH-13 API for Third-Party Integration*.

Each type of information is strategically connected to the business model and user experience:

- **Customer Segments and Relationships:** User and profile information support personalized and secure access for different user categories, for an initial case: students, professionals, and small businesses.
- **Value Proposition:** File metadata, ACLs, and activity logs guarantee reliable and organized file management, reinforcing the core service value.
- **Revenue Streams:** Subscription and payment data ensure the sustainability of the freemium and premium plans.
- **Key Activities:** Monitoring user behavior and storage usage enables continuous system optimization and scalability planning.

Whith these information requirements that define the data backbone of the system, guiding the subsequent database query design and ensuring consistency between storage, performance, and business objectives.

6 Data System Architecture

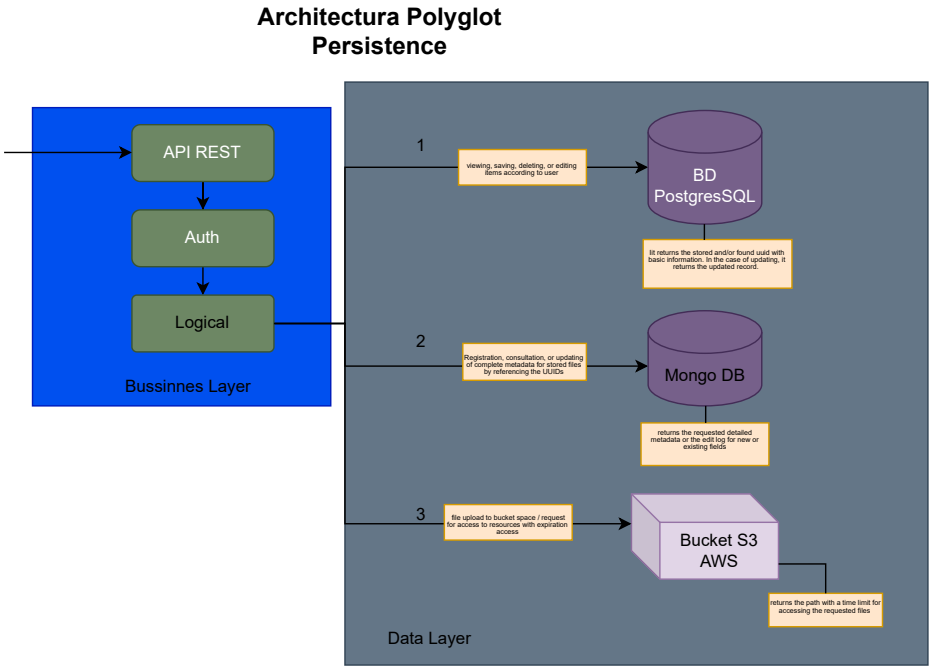

Figure 5: Architecture High Level Model for the File Storage Platform.

The system implements a Polyglot Persistence Architecture designed for a cloud-based file storage system (Dropbox-like), optimizing data management through specialized database technologies. The architecture follows a three-tier model with clear separation of concerns between the Business Layer and Data Layer.

6.1 Component Roles in the Flow

6.2 Logical Orchestrator

Provides advanced workflow management across several data stores and acts as the focal point for all multi-step procedures. File type whitelist validation, file size restriction enforcement, and user storage quota verification are among the business rules enforced by this component. The orchestrator coordinates sequential operations across PostgreSQL, MongoDB, and S3 while preserving compensating transaction logic for rollback circumstances by implementing the SAGA pattern for distributed transaction management. In order to provide end-to-end request tracing across all system components, this component creates distinct correlation identifiers for every action.

6.3 PostgreSQL Database

plays two roles in the sequence of data flow. As the authoritative source of records in Step 1, PostgreSQL creates first file records with status='PENDING' and generates unique file identifiers (UUID v4). This first entry creates the file's presence in the

system and stops it from showing up in user interfaces until the upload is finished. By using database-level constraints to ensure referential integrity, the database preserves important relational structures, such as foreign key relationships between users, folders, and files. Step 4: PostgreSQL completes the last state transition by changing the file status from 'PENDING' to 'COMPLETED' and storing the S3 storage key. This essentially commits the distributed transaction and makes the file accessible to users.

6.4 MongoDB Database

Serves as the adaptable metadata repository, holding variable-structure metadata documents that vary greatly depending on the type of file. Without the need for foreign key constraints, each MongoDB object makes reference to the file-id produced by PostgreSQL, creating a logical relationship. Nested, hierarchical metadata structures, such as EXIF data for photos, codec information for films, and extracted text for documents, are all expertly stored by MongoDB. Rapid feature development is made possible by the schema-less design, which only involves inserting documents with more metadata fields rather than requiring schema changes. Full-text search across extracted material, tag-based filtering, and sophisticated metadata searches are all supported by MongoDB's extensive query language.

6.5 AWS S3 Bucket

It employs a key-value storage model in which each file is represented by a unique object key, enabling scalable and efficient management of binary data. AWS S3 automatically handles geographic replication, high availability (99.99% SLA), and exceptional data durability (99.999999999%), ensuring continuous access and data resilience. The key naming encodes hierarchical organization while guaranteeing global uniqueness through the `file_id` component. Moreover, S3's pre-signed URL mechanism allows secure, direct client-to-storage data transfers, removing the application server from the critical download path and improving system throughput. Its built-in versioning functionality preserves historical file states, facilitating rollback operations and ensuring compliance with data retention policies.

6.6 Justification of Technologies

PostgreSQL was selected as the primary relational database management system due to its robustness, reliability, and advanced support for structured data. It provides strict data integrity through ACID compliance, which is essential for managing critical entities such as user accounts, authentication, access control, and transactional operations. The relational schema allows the definition of complex relationships and constraints, ensuring data consistency and enforcing referential integrity across users, permissions, and file ownership. Additionally, PostgreSQL's support for stored procedures, indexing, and JSON fields offers the flexibility to handle semi-structured data without compromising performance or normalization principles.

MongoDB complements the relational model by managing unstructured and highly variable data, particularly the metadata associated with stored files. Since each file may contain distinct attributes depending on its type (e.g., images, videos, documents),

MongoDB's document-oriented architecture enables dynamic schema evolution without the need for rigid table structures. This flexibility allows for rapid scaling and efficient querying of metadata at high volumes, while maintaining strong integration with the application layer through native JSON compatibility. Furthermore, MongoDB's sharding and replication capabilities ensure horizontal scalability and data redundancy, which are crucial for a cloud-based storage system.

Amazon S3 (Simple Storage Service) was chosen as the core object storage solution to handle the actual file data independently of the databases. S3 provides virtually unlimited, durable, and cost-effective storage with built-in redundancy across multiple availability zones. By decoupling file storage from metadata management, the architecture optimizes both performance and scalability. Files are stored in S3 buckets, while their metadata and relational references are maintained in MongoDB and PostgreSQL, respectively. This separation of concerns enhances security, simplifies data retrieval, and supports advanced features such as access versioning, encryption, and temporary file-sharing links — all essential for a professional-grade file storage platform.

7 Query Proposal

7.1 User Account Information

This query retrieves the user's stored credentials and account status to verify their identity and check if the account is active.

```
SELECT id, email, password, status
FROM account
WHERE email = 'user@example.com';
```

- Purpose: Authenticate a user during login.
- Insight: Verifies user credentials and retrieves basic account status.

7.2 Personal Profile Data

This query joins the account and info account tables to get a complete view of the user's profile.

```
SELECT a.email, ia.first_name, ia.last_name, ia.phone, ia.birthdate
FROM account a
JOIN info_account ia ON a.id = ia.account_id
WHERE a.id = 'USER123';
```

- Purpose: To display a user's profile information on their account settings page.

7.3 File Metadata

This query retrieves non-trashed files from a specific folder, including their format details, sorted by upload date.

```
SELECT f.id, f.name, f.upload_at, f.size, ff.extension
FROM file f
JOIN format_file ff ON f.format_file_name = ff.name
WHERE f.folder_id = 555 AND f.is_trashed = FALSE
ORDER BY f.upload_at DESC;
```

- Purpose: To list all files within a specific folder for navigation..

7.4 Storage Usage Data

This analytical query aggregates (SUM) the size of all files a user owns and compares it to their plan's capacity.

```
SELECT sp.storage_capacity AS plan_limit,
    COALESCE(SUM(f.size), 0) AS used_space
FROM storage_space ss
JOIN storage_plans sp ON ss.id = sp.id
JOIN file f ON f.storage_space_id = ss.id
WHERE ss.account_id = 'USER123'
GROUP BY sp.storage_capacity;
```

- Purpose: To check a user's total used storage against their plan's limit.

7.5 Access Control and Permissions

This query checks complex ownership and sharing rules. The relational model is strong for enforcing these multi-conditional access controls.

```
SELECT f.id
FROM file f
LEFT JOIN folder fo ON f.folder_id = fo.id
WHERE f.id = 'FILE789'
AND (f.owner_id = 'USER123' OR fo.shared_with @> ARRAY['USER123']);
```

- Purpose: To verify if a user has permission to download a specific file.

7.6 Subscription and Payment Records

This query involve multiple joins across transactional tables (Payment, Subscription, Account) to generate a business report.

```
SELECT a.email, s.id AS subscription_id, p.created_at, p.total_payment
FROM payment p
JOIN subscription s ON p.subscription_id = s.id
JOIN account a ON s.account_id = a.id
WHERE p.status = 'FAILED'
AND p.created_at > CURRENT_DATE - INTERVAL '7 days';
```

- Purpose: To find all users with a failed payment for a follow-up process.

7.7 API Integration Data

Similar to the SQL approach, but the flexible document model allows for easily adding new types of permission scopes or metadata to the token without schema changes.

```
db.apiCredentials.findOne(
    { api_key: "tok_xyz123", is_active: true },
    { projection: { user_id: 1, permissions: 1 } }
)
```

- Purpose: To validate an API token and retrieve its associated permissions.