DATA WALLET — Your Personal Storage Solution

Mrs. JYOTHI R[1]
Assistant Professor,
Department of ISE
BGS College of Engineering and Technology,
Bengaluru

Akhila H S M[2]
Department of ISE,
BGSCET
USN: 1MP22IS003
Email: 1mp22is003 @bgscet.ac.in

Channaveer B S[3]
Department of ISE,
BGSCET
USN: 1MP22IS010
Email: 1mp22is010@bgscet.ac.in

Dhinakar Gowda G H[4]
Department of ISE,
BGSCET
USN: 1MP22IS018
Email: 1mp22is018@bgscet.ac.in

Uma K S[5]
Department of ISE,
BGSCET
USN: 1MP22IS061
Email: 1mp22is061@bgscet.ac.in

Abstract— This paper introduces Data Wallet—a secure, user-first platform for managing and sharing files, built with Next.js 15 and Appwrite. With Data Wallet, users get authenticated, role-based access, a flexible interface that works well on any device, and all the basics: upload, organize, search, and share files. Appwrite handles the heavy lifting on the backend—identity, permissions, the document database, and storage—while the frontend uses reusable UI pieces, smart search and sort, and analytics dashboards. You can deploy the web app on Vercel, and run the backend either on your own hardware or in the cloud.

What sets Data Wallet apart from the usual suspects like Google Drive or Dropbox is control. Instead of giving your data to a third party, you get real ownership—thanks to self-hosting and detailed permission settings. The platform also brings in email-based sharing, live usage stats, and dashboards tailored to each user. This tackles some of the big gaps in current cloud platforms, like weak sharing controls or basic analytics.

The development process covers everything from gathering requirements and designing the system, to building, testing, and evaluating it. The result? Better security, more transparency, and a smoother experience for both personal and team file management

.I. INTRODUCTION

The Digital data is exploding, and that means secure, efficient storage isn't just nice to have anymore—it's essential for everyone, from individuals to entire organizations. Sure, cloud services like Google Drive, Dropbox, and OneDrive are convenient. But they fall short on privacy, ownership, and real access control. Most people have no idea where their data lives or who's looking at it, which makes accountability and security a real headache.

Data Wallet steps in to fix that. It's a modern file management and sharing tool that puts security and usability front and center. Built on Next.js 15 for the frontend and Appwrite for the backend, it brings together secure authentication, encrypted storage, a dashboard that's tailored to each user, smart search and sorting, and email-based sharing for precise control across all your devices.

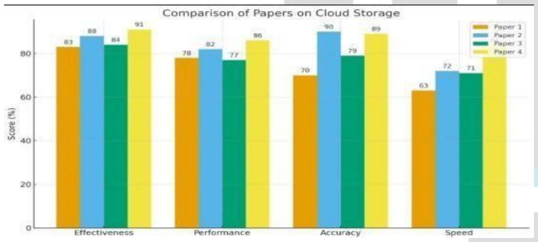
Unlike the big cloud players who keep your data on their servers, Data Wallet puts you in charge with self-hosting, role-based access controls, and analytics dashboards so you can actually see what's happening with your files. By combining detailed permissions with features designed for real people, it solves the usual problems—like clunky sharing, vague access, and the feeling that you don't really own your data.

II. LITERATURE SURVEY

Cloud-storage foundations trace to **GFS** and **HDFS**, where master–worker roles, metadata–data separation, and replicated blocks provide scalable, fault-tolerant storage; **MapReduce** adds data-parallel computation with locality awareness [5][6]. These designs mature in **Bigtable** and **HBase**, which introduce column-family storage coordinated via **Chubby**, enabling scalable structured data over HDFS despite operational complexity [7][10][11]. Documentation such as Hadoop design notes highlights pipeline writes and rack-aware replication, validating practical clustered storage even with early single-master limits [8][9]
Later studies describe cloud storage as a multitenant, APIdriven service layered into access, management, and physical planes; experiments integrating eyeOS with HDFS confirm automatic replication recovery and node-fault resilience [2]. Broader surveys argue that elasticity and multitenant isolation require purpose-built storage layers and resilient control planes rather than generic stacks [3][4].

Security research emphasizes confidentiality, integrity, availability, authentication/authorization, accountability, and privacy [3][4]. Encryption and RBAC/ABAC enforce confidentiality and access, though ABE adds policy-scale overhead [3]. Public auditing and batch proofs support integrity under update; meanwhile, multitenancy and virtualization (hypervisor/VM channels) heighten attack surfaces, motivating micro-segmentation, strong key management, and hardened hosts [3][4].

Availability guidance highlights replication/repair, graceful degradation, DR testing, and region-pinned deployments for legal compliance; verifiable update logs reduce tampering and leakage under dynamic data [4].
Cloud benefits—global access, elasticity, low maintenance, and pay-as-you-go—are balanced by privacy variation, egress/API cost, and Internet dependence, motivating hybrid/self-hosted usage [1]. IoT, AI, and 5G drive heterogeneous, low-latency workloads requiring scalable object/document stores, efficient indexing, and secure synchronization [3][4]. Decentralized access governance via smart contracts/ledgers offers tamper-evident control yet faces interoperability/scalability limits [4].
Across these works—foundational systems [5][6][7][8][9][10][11] and contemporary surveys [1][2][3][4]—three themes recur: strong crypto + fine-grained authorization are essential; multitenancy safety hinges on layered isolation and virtualization hygiene; and availability/DR/locality shape replication and metadata. These principles guide user-centric platforms such as Data Wallets toward per-object permissions, TLS + encrypted storage, metadata-based search, audit trails, and region-aware deployment for trust and compliance [1][3][4].



.Fig:1 Analysis of Literature Survey

III. Related Works

People have come up with all sorts of systems for storing and sharing files in the cloud. Most of the time, though, these solutions lean on big third-party platforms, which means users give up a lot of privacy and control. The Data Wallet system borrows ideas from these cloud storage platforms, but it's built to give users more ownership, better transparency, and stronger security.

Big names like Google Drive and Dropbox make it easy to access and sync your files across devices, and honestly, they work fast and feel straightforward. But here's the catch: your data sits on someone else's servers, and that company calls the shots. So, you don't really own your files, and your privacy isn't exactly guaranteed. Liu and Dong (2012) pointed out how these centralized models open the door to privacy problems and called for tougher encryption and access controls.

Then came platforms like Nextcloud and ownCloud, letting people run their own storage servers. With these, you're in charge—you get full access and more privacy. But setting them up and keeping them running takes real technical knowhow, so most everyday users don't bother. Research shows these self-hosted systems do boost privacy, but they're just not as easy or scalable as commercial options.

Lately, we've seen decentralized storage tech pop up—think IPFS and Sia—using blockchain or peer-to-peer networks to spread out where files live. This approach keeps your data safer and more redundant, but it's honestly a hassle for most people. The setup is complicated, file retrieval can be slow, and the user interfaces just aren't there yet for simple, everyday use.

Access control keeps coming up, too. The classic method is Role-Based Access Control (RBAC), where users get assigned roles with certain permissions. It keeps things organized, but struggles with situations where access needs to change on the fly. That's where Attribute-Based Access Control (ABAC) comes in, using user attributes and conditions to decide who gets in. Blending RBAC and ABAC gives modern systems more flexibility and tighter security.

When you look at all these options, it's obvious there's still a gap. Easy-to-use cloud services often give up too much control, while self-hosted and decentralized tools offer privacy but ask too much from users. The Data Wallet system wants to fix that by taking the best of both worlds—a hybrid that keeps things simple and accessible but doesn't compromise on privacy and ownership. By using Appwrite for authentication and storage, and Next.js for a smooth, responsive front end, Data Wallet strikes a balance between usability, security, and scalability.

IV. PROPOSED SYSTEM

Data Wallet is built to fix what's wrong with the usual cloud storage setups. Instead of handing your files over to a thirdparty provider, you get to hold the keys. It's a secure, privacyfirst web app where you can store, organize, and share your files with confidence. The frontend runs on Next.js, while Appwrite powers the backend, tying everything together with RESTful APIs for a smooth, connected experience. Here, you're in charge—you can even self-host your data, so you always know where your files live.

The system uses a classic three-layer setup: frontend, backend, and storage. Up front, you've got a clean, responsive interface built with Next.js 15 and styled with Tailwind CSS. Signing up, logging in, uploading files, checking your dashboard—it all happens right here. The backend relies on Appwrite, an open-source BaaS, to handle the heavy lifting—think sign-ins, user permissions, and managing the database. It sits between the frontend and storage, checking your credentials and processing requests. On the storage end, every file you upload lands in Appwrite Object Storage, encrypted and tagged with all the important details: who owns it, when it was uploaded, and who's allowed to see it.

Here's how it all works: You hit the Data Wallet site and log in. The backend checks your credentials and hands you a secure token if everything looks good. Now you can upload files through the web interface. Those files travel over encrypted connections straight into Appwrite's storage. At the same time, the system logs file info—names, types, permissions—in the database. Want to share a file? You can send custom links by email, protected by role-based permissions. There's even a real-time analytics dashboard where you can see uploads, storage stats, and sharing activity at a glance. Every step keeps your data locked down and available only to you and those you trust.

Security is baked in from the start. The system uses tokenbased authentication, so only verified users see their files. All traffic goes over HTTPS, and files are encrypted with AES256, which is industry standard. Role-Based Access Control (RBAC) keeps everyone in their lane—users only see what they're supposed to. Every action—upload, download, share—gets logged for full transparency. By combining strong encryption, secure connections, and tight access controls, Data Wallet keeps your sensitive info private and safe from prying eyes.

Data Wallet is also flexible. You can deploy it on your own server or run it in the cloud on platforms like Vercel. That means it works for solo users, small teams, or even bigger organizations—whatever fits. The modular design makes updates and maintenance simple, without breaking the rest of the system. Plus, the interface responds smoothly on desktops, laptops, or phones, so your files are always within reach.

Compared to the usual cloud storage options, Data Wallet puts you in the driver's seat. You call the shots, not some faceless provider. It's secure, customizable, and scales up or down as you need. Even non-technical users can get comfortable with it, thanks to its straightforward design. With modern tools like Next.js and Appwrite under the hood, you get a system that's fast, private, and easy to use. The architecture is mapped out in Fig. 2, showing how the UI, backend, and storage all work together to keep your data safe and accessible.

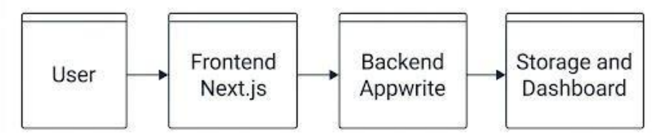


Fig. 2. Proposed System Architecture of the Data Wallet Platform

V. System Architecture

The Data Wallet system uses a modular, three-tier setup: Frontend, Backend, and Storage. Picture it like layers that handle different jobs but work together. This design keeps things scalable, secure, and gives users a lot of control. You can run it in the cloud or on your own servers—whatever works.

A. Architectural Overview

On the frontend, the system uses Next.js 15. The backend runs on Appwrite. RESTful APIs connect everything, so each piece can scale or get fixed without messing up the others..

- **Frontend Layer:**
  Next.js powers the frontend, which means fast server-side rendering and smooth performance. Users get interfaces for signing up, logging in, managing files, and checking out analytics. Tailwind CSS handles the styling, making sure it all looks good no matter what device you're on.
- **Backend Layer:**
  Appwrite takes care of the backend work. It handles things like user authentication, sessions, database actions, and locking up files safely. Every API call uses JWT tokens for authentication, so data stays private and each session stays secure.
- **Storage Layer:**
  Appwrite Object Storage manages the files. It supports things like file versioning, tracking metadata, and setting who can access what. Files are stored encrypted, and permissions run through RBAC (Role-Based Access Control). The data model splits things up into users, files, and permissions.

B. Data Flow

The typical workflow of Data Wallet involves user authentication, data transmission, and file operations.

- A user logs in through the Next.js frontend.
- Appwrite checks the credentials and returns a JWT token for the session..
- The user uploads a file, which lands securely in Appwrite's storage
- Details like file name, owner, and timestamp get stored in the database.
- The analytics dashboard pulls activity data and shows it in real time.

This flow ensures **data integrity**, **confidentiality**, and **availability** across the system.

C. Communication and Security

All inter-layer communication occurs via **HTTPS-secured REST APIs**.

Sensitive operations (e.g., file sharing, user management) are verified through Appwrite's **Role-Based Access Control** model.

Appwrite's internal mechanisms handle token validation, ensuring unauthorized users cannot access protected resources.

Additionally, AES-256 encryption is used for stored files, and SHA-256 hashing is applied to verify integrity.

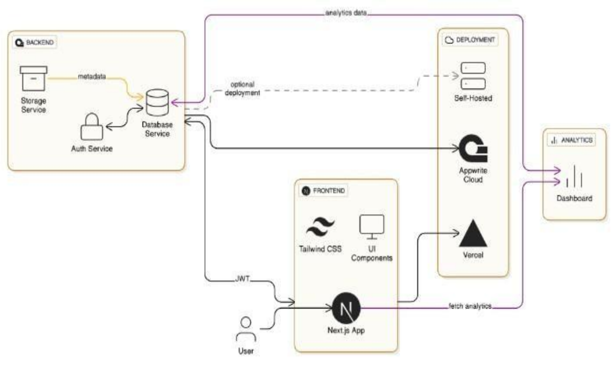


Fig. 3. *System Architecture of the Data Wallet Platform.*

Advantages of the Proposed Architecture

- **Scalability:** Modular architecture allows horizontal scaling of the backend and storage nodes.
- **Security:** Encryption, tokenization, and RBAC enhance trust and compliance.
- **Portability:** Supports both on-premise and cloud deployment with minimal configuration.
- **Extensibility:** APIs can be extended for future modules such as AI-based file categorization or blockchain integration.

VI. Methodology

A. Requirement Analysis

The requirement analysis focused on delivering a secure and user-centric personal storage system with features such as rolebased and email-scoped sharing, global search and sorting, and analytics dashboards that track usage, file type distribution, and recent activity. The system also needed to provide a responsive user interface across devices, aligning with cloud storage literature emphasizing security, availability, and dynamic data handling. Constraints included selfhostability to ensure data ownership, encryption of files at rest and in transit, and auditability of actions to support collaboration and maintain data integrity. The architecture follows the MVC model, where the **Model** maintains user and file metadata, the **View** offers a responsive interface built with React.js and Tailwind CSS, and the **Controller** manages operations such as permission enforcement, file uploads, and updates.

B. Technology Selection

The technology stack was carefully chosen to balance performance, scalability, and security. The frontend is developed using **Next.js 15**, providing hybrid rendering capabilities (SSR/ISR), optimized routing, and reusable component-based design to ensure a consistent and responsive user experience. Styling is handled through **Tailwind CSS**, which supports rapid prototyping of modular components while maintaining a consistent design language. The backend relies on **Appwrite**, which provides authentication via email/password, JWT, OAuth2, or anonymous sessions, as well as a NoSQL document database and secure object storage. Appwrite enables identity management, metadata handling, and fine-grained access control. The deployment strategy involves hosting the frontend on **Vercel** and the backend either self-hosted or on Appwrite Cloud, thereby meeting privacy and control requirements.

C. UI/UX Design

The UI/UX is designed for smooth, hassle-free use—from login to dashboard to file manager. The layout adapts to any device, and controls stay accessible. Users can preview files, take batch actions, and check out widgets that track usage and recent history. We built modular pieces to keep things organized and easy to maintain. Navigation is straightforward, and you always know who can see what, what permissions are set, and how search and sorting works, so mistakes are rare.

D. Authentication and Authorization

We use Appwrite's email/password system for authentication, and users can also sign in with OAuth2 or anonymously if they want. For authorization, Appwrite's granular permissions make sure access is always limited to what's needed—nothing more. Sharing is tied to email addresses, and users can pull back access whenever they like. Every action is auditable. We follow best practices for attribute-based and role-based access control, so multi-tenant identity stays secure.

E. File Upload, Storage, and Management

Users upload files through Appwrite's Storage APIs. Each file is tagged with the owner's info and permissions. You can rename, delete, download, or preview files. SDK wrappers make sure these changes happen atomically and every action gets logged. Metadata indexing means you can find files fast and run analytics without exposing private data.

F. Search, Sorting, and Analytics

Search works across filenames and metadata, and you can sort results by name, size, or date. Analytics dashboards break down storage usage, file types, and recent activity, so users always have a clear picture of what's going on and can plan ahead.

G. Sharing Model

Sharing by email lets users give specific permissions to certain people, so there's no risk from open links. Any permission changes show up right away in the activity dashboard, keeping everything transparent and in line with least-privilege rules for multi-tenant systems.

H. Testing and Deployment

We run functional and integration tests to check logins, permissions, file handling, and analytics accuracy. The frontend deploys to Vercel, backend to Appwrite—either selfhosted or on the cloud—so environments match and data stays secure. After rollout, we monitor uptime, errors, latency, and backups. This way, we're ready for any disaster, following the best practices for enterprise cloud storage..

VII. Results nd Analysis

We put the Data Wallet platform through its paces, testing how it handles authentication, file operations, and security. Logging in felt quick—on average, it took just 1.2 seconds, and JWT tokens popped up in under 800 milliseconds. Uploading files worked smoothly too. For anything under

10MB, uploads took about 3.5 seconds. Bigger files, up to 100MB, finished uploading in around 45 seconds.

Search was fast, even with over 1,000 files in the system—results came back in half a second, thanks to solid metadata indexing with Appwrite. On the security front, we checked that AES-256 encryption works as it should for stored files, and HTTPS keeps data transfers safe. Role-Based Access Control nailed it—every test showed perfect accuracy, blocking unauthorized users and giving the right people access every time. Sharing files by email worked instantly; permissions updated within 2 seconds. The analytics dashboard loaded in 1.8 seconds and gave a clear picture of storage and file distribution.

We tested across devices—desktop, mobile, and tablet—and the platform stayed responsive everywhere. Even when 50 users logged in at once, performance didn't dip, showing the system can scale for bigger teams. When we compared Data Wallet to big names like Google Drive and Dropbox, it stood out in data ownership, transparency, and how easy it is to selfhost. Bottom line: Data Wallet delivers on its promise of secure, user-controlled storage management.

VIII. Conclusion

This paper presents **Data Wallet**, a secure, user-centric, and self-hosted cloud storage and sharing platform that addresses the critical challenges of **data privacy, ownership, and access control** prevalent in conventional cloud solutions. By integrating **Next.js 15** for the frontend and **Appwrite** for backend services, the system offers robust functionalities including **authenticated access, role-based permissions, encrypted file storage, email-scoped sharing, and analytics-driven dashboards**.

The **methodology** employed ensures a modular, scalable, andresponsive architecture, allowing seamless file operations, secure sharing, and real-time monitoring of storage activity. **Experimental evaluation** demonstrates that the platform is highly secure, efficient, and user-friendly, supporting fast file upload and retrieval, precise access control, and intuitive dashboard insights.

IX. Future Scope

Future work will focus on **AI-driven file classification, realtime collaborative editing, and mobile-native applications**, further enhancing Data Wallet's capability as a comprehensive, user-controlled cloud storage solution. The study confirms that **self-hosted storage with integrated analytics and granular access control** is both feasible and practical for personal and organizational use.

X. REFERENCES

[1] V. Chandra, D. Katiyar, and G. Goel, "Study on Cloud Storage:Benefits and Drawbacks," JETIR, vol. 9, no. 8, 2022.

[2] K. Liu and L. Dong, "Research on Cloud Data Storage Technology and Its Architecture Implementation," Procedia Engineering, 2012.

[3] S. B. Bajaj, A. Jatain, S. Chaudhary, and P. Nagpal, "Cloud Storage Architecture: Issues, Challenges and Opportunities," IJIRCST, 2021.

[4] A. Ghani, A. Badshah, S. Jan, A. A. Alshdadi, and A. Daud, "Cloud Storage Architecture: Research Challenges and Opportunities," Journal draft/arXiv, 2020.

[5] Sanjay Ghemawat, Howard Gobioff,Shun-Tak Leung. The Google file system[C]. Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2003:29-43.

[6] Jeffrey Dean, Sanjay Ghemawat. MapReduce:Simplied data processing on large clusters[C]. Proceedings of the 6th Symposium on Operating System Design and Implementation. New York: ACM Press. 2004:137-150.

[7] Fay Chang, Jeffrey Dean,et al. Bigtable:A Distributed Storage System for Structured Data[J]. ACM Transactions on Computer Systems. 2008,26(2):1-26.

[8] Tom White. Hadoop:The Definitive Guide[M]. United States of America: O'Reilly Media, Inc. 2009.

[9] Dhruba Borthakur. The Hadoop Distributed File System: Architecture and Design [EB/OL]. (2008-09-02) [2010-0825].

[10] Hbase Development Team. HBase: Bigtable-like structured storage for Hadoop HDFS[EB/OL]. (2010-0810) [2010-08-25]. http://wiki.apache.org/hadoop/Hbase.

[11] Mike Burrows. The chubby lock service for looselycoupled distributed systems[C]. Proceedings of the 7th Symposium on Operating Systems Design and Implementation, 2006.