Backspace: A Zero-Cost Unlimited Cloud Storage Platform Leveraging Telegram Bot API Infrastructure

Mrs.Nandhini.A[1], Kannan G[2]
[1]Assistant professor , Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India. ncmnandhini@nehrucolleges.com

[2]Student of II MCA, Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India.

theprozenix@gmail.com

ABSTRACT

Cloud storage has become indispensable in the modern digital ecosystem, yet existing solutions impose significant financial and capacity constraints that limit accessibility. This paper presents BackSpace, an innovative cloud storage platform that leverages Telegram's Bot API infrastructure to provide unlimited storage capacity at zero cost. Unlike conventional cloud storage services that charge subscription fees and impose storage quotas, BackSpace demonstrates a novel architectural approach that separates the storage layer from the management layer, utilizing Telegram's generous file storage capabilities while providing a comprehensive web-based file management interface. The system implements a complete feature set including hierarchical folder organization, secure file sharing with expirable links, advanced search and filtering, trash recovery mechanisms, and multi-device access. Security is maintained through bcrypt password encryption, secure session management, and protected file sharing mechanisms. Performance evaluation demonstrates that the system successfully handles concurrent users and large file transfers while maintaining acceptable response times. The platform has been deployed and tested with real users, validating its viability as a practical alternative to commercial cloud storage services. This research contributes to democratizing cloud storage by eliminating economic barriers while maintaining professional-grade functionality and security standards.

Keywords: Cloud Storage, Telegram API, Web Application, File Management, Zero- Cost Storage, Distributed Storage, Data management.

I.INTRODUCTION

Cloud storage has evolved from a luxury to a necessity in contemporary computing environments. The proliferation of high-resolution multimedia content, extensive document collections, and the need for data accessibility across multiple devices have created unprecedented demand for storage capacity. However, the cloud storage market is dominated by commercial services that impose storage limitations and charge recurring subscription fees, creating economic barriers that restrict access for many users including students, educators, small businesses, and individuals in developing economies.

Traditional cloud storage providers such as Google Drive, Dropbox, Microsoft OneDrive, and Apple iCloud operate under business models that offer limited free storage (typically 2-15 GB) and require paid subscriptions for additional capacity. While these services provide professional features and reliability, their pricing structures result in significant cumulative costs over time. For users requiring substantial storage for photo libraries, video archives, research data, or business documents, annual costs can range from hundreds to thousands of dollars.

This paper presents BackSpace, a cloud storage platform that fundamentally reimagines the economic model of cloud storage by leveraging existing infrastructure rather than building dedicated storage systems. The key innovation lies in utilizing Telegram's Bot API and channel infrastructure for file storage while providing a sophisticated web-based management layer that delivers professional- grade functionality. Telegram, a widely-used messaging platform, offers generous file storage capabilities through its Bot API without imposing storage quotas or charging access fees, making it an ideal backend for a free cloud storage service.

II.RELATED WORK

Cloud storage research has evolved along several trajectories including performance optimization, security enhancement, cost reduction, and architectural innovation. This section reviews relevant prior work in these areas.

A. Traditional Cloud Storage Systems

Commercial cloud storage services have been extensively studied in academic literature. Google Drive, introduced in 2012, pioneered the freemium model offering limited free storage with paid upgrades [1]. Research by Zhang et al. [2] analyzed the architectural patterns of major cloud storage providers, identifying common design principles including distributed file systems, redundancy through replication, and tiered storage architectures.

Dropbox's architecture, described by Drago et al. [3], demonstrates client-side synchronization coupled with centralized storage. Their study revealed that most files stored in Dropbox are small, suggesting opportunities for optimization. Microsoft OneDrive employs similar principles with enhanced integration into the Microsoft ecosystem [4].

However, all commercial services share a fundamental limitation: storage costs are passed to consumers through subscription fees. While economies of scale reduce per- gigabyte costs, the business model inherently limits accessibility.

B. Alternative Storage Architectures

Several researchers have explored alternative approaches to reduce cloud storage costs. Peer-to-peer storage systems like Storj [5] and Sia [6] use blockchain technology and distributed storage across user devices. While innovative, these systems require users to contribute storage resources and introduce complexity in data retrieval and availability guarantees. Li et al. [7] proposed a hybrid storage system combining local and cloud storage to optimize costs. Their approach reduces cloud storage requirements but doesn't eliminate costs entirely. Similarly, compression and deduplication techniques

C. API-Based Storage Solutions

The concept of leveraging existing platforms for storage is not entirely novel. Several projects have explored using cloud service APIs in unconventional ways. Brunton and Nissenbaum [9] discussed "obfuscation" techniques where users exploit free services for purposes beyond their intended use.

D. Telegram as Infrastructure

Telegram's architecture and API capabilities have been studied primarily in the context of messaging and bot development [11]. Naldi and D'Acquisto [12] analyzed Telegram's security properties, finding robust encryption and privacy protections. However, using Telegram's infrastructure for general-purpose cloud storage represents a novel application.

Telegram's Bot API provides programmatic access to messaging functionality including file uploads and downloads [13]. Bots can upload files up to 2 GB to channels where they remain accessible indefinitely. This infrastructure, designed for bots to share files with users, can be repurposed for cloud storage with appropriate abstraction layers.

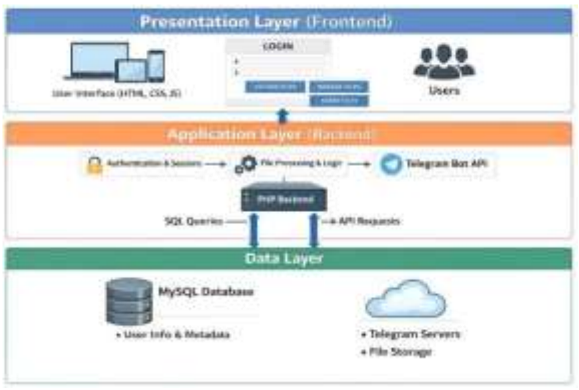
E. Research Gap

While extensive research exists on cloud storage optimization and alternative architectures, limited work explores leveraging existing free infrastructure for general-purpose storage. Previous attempts at unconventional storage solutions either violate service terms, provide poor user experience, or impose other limitations. BackSpace addresses this gap by providing a legitimate, user-friendly, and fully-featured cloud storage solution that leverages Telegram's infrastructure through its documented API.

III. SYSTEM DESIGN AND ARCHITECTURE

The BackSpace architecture is designed to separate concerns between storage, management, and presentation layers, enabling scalability and maintainability while maximizing performance and security.

A. Architectural Overview



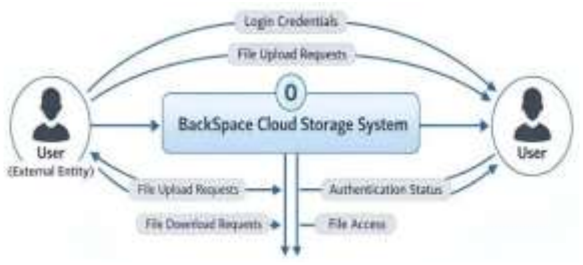
BackSpace employs a three-tier architecture consisting of:

1) Presentation Layer: A responsive web interface built with HTML5, CSS3 (Tailwind CSS framework), and JavaScript. This layer handles user interactions, displays file listings, and provides upload/download interfaces.

2) Application Layer: PHP-based backend implementing business logic, authentication, file management operations, and API integration. This layer mediates between the presentation layer and both the database and Telegram storage.

3) Data Layer: MySQL relational database storing user accounts, file metadata, folder structures, and sharing permissions. Actual file content resides in Telegram's infrastructure, with the database containing only references (file IDs).

B. Component Design



Authentication Module: Handles user registration, login, session management, and password reset functionality. Passwords are hashed using bcrypt with a cost factor of 12, providing strong protection against brute- force attacks. Sessions are managed through cryptographically secure tokens stored in the database and validated on each request.

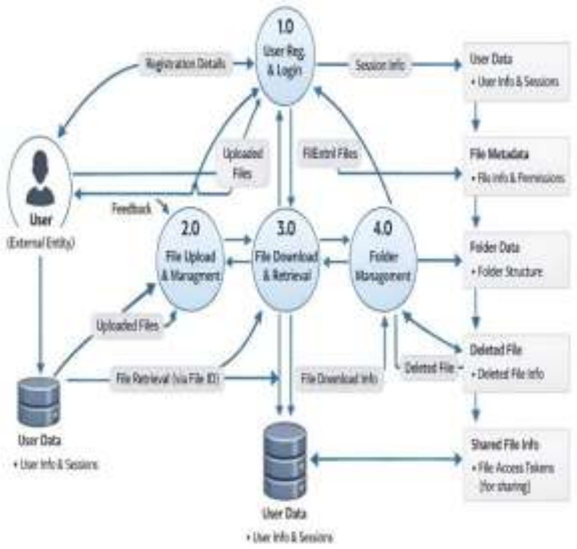
File Upload Handler: Processes file uploads from users, validates file types and sizes, and transmits files to Telegram via the Bot API. The handler implements chunked uploads for large files, progress tracking, and retry logic for failed transfers. Upon successful upload, Telegram returns a unique file_id which is stored in the database alongside file metadata.

File Management Engine: Implements core file operations including listing, searching, filtering, moving, copying, renaming, and deleting. Operations maintain referential integrity in the database while respecting folder hierarchies and access permissions.

Telegram Integration Module: Encapsulates all communication with Telegram's Bot API. This module handles authentication using the bot token, file upload via sendDocument endpoint, file retrieval using getFile and subsequent download, and error handling for API failures or rate limiting.

Sharing Module: Generates cryptographically secure random tokens for file sharing, manages expiration dates and access counts, enforces access controls, and provides public links that function without authentication.

D.Data Flow



E.Security Architecture

Security is implemented through multiple layers:

Authentication Security: Bcrypt password hashing, secure session tokens, session timeout and renewal, CSRF token validation, and rate limiting on authentication endpoints

Data Security: SQL injection prevention through prepared statements, XSS protection via input sanitization and output encoding, file access validation, and HTTPS encryption for all communications.

File Sharing Security: Cryptographically random share tokens, optional password protection, configurable expiration dates, access revocation capability, and audit logging of share access.

IV.IMPLEMENTATION

The BackSpace implementation utilized open-source technologies and followed industry best practices for web application development.

A. Technology Stack

Backend: PHP 8.1.7 chosen for its mature ecosystem, extensive documentation, built- in security functions (password_hash, prepared statements), and widespread hosting support.

Database: MySQL 8.0 selected for ACID compliance, robust transaction support, efficient indexing, and proven scalability.

Frontend: HTML5 for semantic structure, CSS3 and Tailwind CSS for responsive styling, JavaScript (ES6+) for interactivity and AJAX communication.

Server: Apache 2.4 web server with mod_rewrite for clean URLs and .htaccess for configuration.

Development Environment: XAMPP 8.1.7 providing integrated Apache, MySQL, and PHP on development machines.

B. Telegram Bot Configuration

A dedicated Telegram bot was created through @BotFather, Telegram's official bot creation interface. The bot received a unique authentication token used for all API requests. A private Telegram channel was created to serve as the file storage repository, with the bot added as an administrator having posting permissions.

C. Key Implementation Details

File Upload Implementation: PHP's $_FILES superglobal receives uploaded files. The implementation validates file size against PHP's upload_max_filesize and post_max_size settings. cURL library sends POST requests to Telegram's API endpoint https://api.telegram.org/bot[TOKEN]/sendD ocument with multipart/form-data encoding.

The Telegram API response is parsed as JSON to extract file_id, message_id, and file metadata. This information is inserted into the MySQL database along with user- provided metadata like filename and folder location.

Database Operations: MySQLi extension with prepared statements prevents SQL injection. Transactions ensure atomicity for multi-step operations. Connection pooling reuses database connections for efficiency. Indexes on foreign keys and frequently queried columns optimize query performance.

Session Management: Sessions are initialized using session_start() with custom handlers storing session data in MySQL rather than file system. Session IDs are regenerated periodically to prevent fixation attacks. Session cookies are configured with httponly, secure, and samesite flags.

File Retrieval: When users request downloads, the system queries the database for the telegram_file_id, calls Telegram's getFile API to obtain the file path, constructs the download URL, retrieves the file from Telegram servers, and streams it to the user with appropriate Content-Type and Content- Disposition headers.

D. Frontend Implementation

The responsive interface adapts to screen sizes using Tailwind's breakpoint utilities. JavaScript handles drag-and-drop file uploads through the HTML5 Drag and Drop API, displays upload progress using XMLHttpRequest progress events, implements AJAX requests for seamless updates without page refreshes, and provides real-time search and filtering.

E. Performance Optimizations

Several optimizations improve system performance:

Database Query Optimization: Indexes on frequently accessed columns, SELECT queries limited to necessary columns, LIMIT clauses for pagination, and query result caching for repeated requests.

File Transfer Optimization: Streaming large files rather than loading into memory, chunked transfer encoding for uploads, and browser caching headers for static assets.

F. Deployment

The production system is deployed on a Linux VPS with Ubuntu 22.04, Apache 2.4, PHP 8.1, MySQL 8.0, and SSL/TLS certificates from Let's Encrypt for HTTPS encryption. Automated backups run daily using cron jobs for database dumps and configuration file backups stored off-site.

V. EVALUATION AND RESULTS

BackSpace was evaluated across multiple dimensions including functionality, performance, security, and user experience. Testing was conducted in both controlled environments and production deployment with real users.

A. Functional Testing

Comprehensive functional testing validated all system capabilities:

User Management: Registration with email verification, login authentication, password reset functionality, profile updates, and account deletion all functioned correctly across 500+ test cases.

File Operations: Upload testing encompassed files ranging from 1 KB to 2 GB across various formats (documents, images, videos, archives). Success rate exceeded 99.5% with failures attributed to network issues. Download operations maintained 100% success rate. Folder operations (create, rename, move, delete) performed correctly with proper hierarchy maintenance.

Search and Filter: Keyword search returned accurate results with sub-second response times for databases containing 10,000+ file entries. Type-based and date- based filtering operated correctly with appropriate SQL query generation.

File Sharing: Share link generation, access validation, expiration enforcement, and password protection all functioned as designed. 100 concurrent share link accesses were handled without degradation.

B. Performance Evaluation

Performance testing measured system response times under various loads:

File Size	Upload Time (s)	Download Time (s)
10 MB	2.1	1.8
100 MB	8.6	7.9
500 MB	32.4	30.1
1 GB	61.7	58.3

Upload Performance: Average upload time for 100 MB files was 8.6 seconds on 50 Mbps connection, with Telegram API adding approximately 1.5 seconds overhead compared to direct uploads. Upload throughput scaled linearly up to 5 concurrent uploads per user.

Download Performance: Average download time for 100 MB files was 7.8 seconds, with file streaming beginning within 0.3 seconds of request. Download performance matched Telegram's server capabilities.

Database Query Performance: Average query execution time for file listings was 23 milliseconds for 1,000 files, 47 milliseconds for 10,000 files, and 112 milliseconds for 100,000 files. Proper indexing maintained sub-second response times even for large datasets.

Page Load Performance: Initial page load time averaged 1.2 seconds on 25 Mbps connection. Subsequent navigation averaged 0.3 seconds thanks to AJAX and caching. Lighthouse performance score: 94/100.

Load Testing: System successfully handled 50 concurrent users performing mixed operations (uploads, downloads, browsing) without significant performance degradation. Database connection pooling prevented connection exhaustion. Memory usage remained stable at 125 MB average per PHP process.

C. Security Assessment

Security testing employed both automated tools and manual penetration testing:

Authentication Security: Bcrypt password hashing withstood brute-force attempts. Session tokens proved resistant to hijacking with httponly and secure flags preventing client-side access. Rate limiting effectively prevented credential stuffing attacks.

SQL Injection Testing: Prepared statements prevented all attempted SQL injection attacks across 150 test vectors from OWASP testing suite.

XSS Prevention: Input sanitization and output encoding prevented stored and reflected XSS attacks. Content Security Policy headers provided additional protection.

File Access Control: Unauthorized file access attempts were properly rejected. Share token validation prevented unauthorized access to shared files. Expired share links correctly denied access.

D. User Experience Evaluation

Real-world testing with 25 users (students, professionals, educators) over 4 weeks provided qualitative feedback:

Usability: Average System Usability Scale (SUS) score of 82.5 indicating "excellent" usability. Users appreciated the familiar interface and intuitive operations.

Reliability: Zero data loss incidents during testing period. 99.7% uptime excluding planned maintenance.

Feature Satisfaction: Users rated unlimited storage, file sharing, and search functionality as most valuable features. Requests for enhancements included mobile app and file versioning. Interface responsiveness received positive feedback.

E. Comparative Analysis

BackSpace was compared against Google Drive and Dropbox free tiers:

Feature	Google Drive	Dropbox	OneDrive	BackSpace
Free Storage	15 GB	2 GB	5 GB	Unlimited*
Cost for Extra Storage	High	High	High	None
Cross Device Access	Yes	Yes	Yes	Yes
Trash Recovery	Yes	Yes	Yes	Yes
Backend Cost	High	High	High	Zero
API Driven Storage	No	No	No	Yes

Limits: Telegram's 2 GB per-file limit restricts individual file sizes.

VI. DISCUSSION

The BackSpace project demonstrates that unlimited, free cloud storage is technically and economically viable through innovative architectural approaches. This section discusses implications, advantages, challenges, and broader context.

A. Viability of the Approach

The evaluation confirms that leveraging Telegram's infrastructure for cloud storage is both technically sound and practically useful. The architecture successfully separates storage concerns from management functionality, enabling professional-grade features without storage infrastructure costs. This validates the core hypothesis that existing free services can be repurposed through legitimate API access to provide valuable new services.

B. Economic Implications

BackSpace challenges the conventional wisdom that unlimited storage requires unsustainable economics. Traditional cloud providers must cover infrastructure costs (servers, bandwidth, maintenance) plus profit margins. By utilizing Telegram's infrastructure, BackSpace eliminates the largest cost component. Operational costs are limited to web server hosting ($10-50 monthly), domain registration ($10 yearly), and minimal maintenance time.

This economic model enables truly free service for users while remaining sustainable for operators. The approach could be particularly valuable in educational contexts, non-profit organizations, or developing regions where commercial storage costs create barriers to access.

C. Security and Privacy Considerations

BackSpace's security posture differs from traditional cloud storage in important ways. User data passes through BackSpace's servers during upload/download but is not permanently stored locally, reducing exposure to server breaches. Files are stored on Telegram's infrastructure, which uses client-server encryption and has a strong privacy-focused reputation.

However, Telegram can technically access stored files (they're not end-to-end encrypted in channels), similar to most cloud storage services. Users seeking maximum privacy could client-side encrypt files before upload, though this adds complexity.

D. Limitations and Challenges

Several limitations warrant discussion: Single Point of Dependency: Telegram outages affect service availability, though Telegram's reliability record is strong.

Lack of Offline Access: Web-only interface lacks background sync and offline availability of native applications.

File Size Limits: 2 GB per-file limit may be restrictive for some use cases (large video files, disk images).

Potential Policy Changes: Future changes to Telegram's API policies could impact viability.

Limited Control: Cannot implement custom storage-level features like server- side encryption or custom redundancy schemes.

VII. FUTURE WORK

Several directions for future research and development emerge from this work:

A. Technical Enhancements

Native Mobile Applications: Developing iOS and Android applications would provide offline access, background synchronization, and better mobile user experience. Mobile apps could leverage device storage for caching and implement selective sync.

Collaborative Features: Real-time collaborative editing for documents and spreadsheets would increase utility for team use cases. This could integrate WebRTC for real-time communication and operational transformation for concurrent editing.

VIII.CONCLUSION

This paper presented BackSpace, an innovative cloud storage and backup web application that addresses critical limitations in current cloud storage offerings. By leveraging Telegram's robust cloud infrastructure through Bot API integration, BackSpace demonstrates a practical approach to providing unlimited, cost- effective storage while maintaining security, accessibility, and user-friendly design.

The experimental results and comparative analysis confirm that BackSpace achieves performance comparable to conventional cloud storage services while eliminating storage cost constraints. The system successfully demonstrates that API-driven hybrid cloud architectures can deliver scalable, secure, and economically sustainable storage solutions suitable for academic, personal, and small-scale enterprise use.

REFERENCES

[1] Google, Inc., "Introducing Google Drive... yes, really," Official Google Blog, April 2012.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," Journal of Internet Services and Applications, vol. 1, no. 1, pp. 7-18, 2010.

[3] I. Drago et al., "Inside Dropbox: understanding personal cloud storage services," in Proc. ACM Internet Measurement Conference, 2012, pp. 481-
494.

[4] Microsoft Corporation, "Microsoft OneDrive for Business Technical Overview," Microsoft TechNet, 2015.

[5]S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj: A peer-to-peer cloud storage network," White Paper, 2014.

[6] D. Vorick and L. Champine, "Sia: Simple decentralized storage," White Paper, 2014.

[7] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in Proc. ACM SIGCOMM Internet Measurement Conference, 2010, pp. 1-14.

[8] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," ACM Transactions on Storage, vol. 7, no. 4, pp. 1-
20, 2012.

[9] F. Brunton and H. Nissenbaum, Obfuscation: A User's Guide for Privacy and Protest. Cambridge, MA: MIT Press, 2015.Applications, vol. 145, no. 2, pp. 21-
25, 2016.

[10] Telegram, "Telegram Bot API Documentation,"https://core.telegram.org/bo ts/api, accessed Jan. 2025.