DATA DUPLICATE DOWNLOAD ALERT SYSTEM

[1] Nagaveni B Biradar, [2] N. Madhushree ,[3]Naina Jain, [4]Megha.G, [5]Hafsa.D

[1]Assosciate professor, Department of Computer Science & Engineering
RYM Engineering College, VTU Belagavi, Karnataka, India

[2345]Department of Computer Science & Engineering, RYM Engineering College, Ballari, VTU Belagavi, Karnataka, India

Abstract

The Data Duplicate Download Alert System (DDDAS) is an intelligent and secure framework designed to ensure data integrity, behavioural intelligence, and real-time monitoring in digital environments. The system automatically detects and prevents duplicate or suspicious file downloads by integrating Flask-based backend automation, AI-driven analytics, and blockchain-enabled immutability. Every file uploaded or downloaded within DDDAS is recorded as a block in the blockchain ledger, where each block contains the file's unique hash value, timestamp, and user details, forming a tamper-proof digital record. This blockchain-based mechanism guarantees immutability—once a file's data is stored, it cannot be altered or deleted—thus ensuring complete data integrity and enabling the system to identify any unauthorized or repeated file activity with high accuracy. The backend, developed using Flask, manages secure hash generation and blockchain synchronization, while the frontend, implemented using Next.js and Tailwind CSS, provides an interactive dashboard for real-time visualization of transactions and AI-generated alerts. The system's integration of Google Gemini AI allows contextual analysis of user behaviour to detect anomalies and assess potential risks beyond conventional rule-based systems. By combining blockchain transparency, AI intelligence, and Flask-based monitoring, DDDAS enhances trust, traceability, and accountability across all file operations. It serves as a next-generation cybersecurity solution that not only protects against redundant downloads and data manipulation but also establishes a transparent, verifiable, and tamper-proof environment for digital asset management.

1. Introduction

In the modern digital era, the exponential rise of cloud-based data sharing and storage has introduced critical challenges in ensuring data integrity, security, and reliability. Frequent file uploads and downloads across distributed systems often lead to duplication, bandwidth inefficiency, and unauthorized data access, making traditional file monitoring methods inadequate. The Data Duplicate Download Alert System (DDDAS) addresses these challenges through a unified intelligent framework that combines Flask-based backend automation, AI-driven behavioural analytics, and blockchain-enabled immutability to create a secure, transparent, and efficient file management ecosystem. Each file uploaded or downloaded is assigned a unique SHA-256 hash and recorded as a block in a blockchain ledger, which stores the hash value, timestamp, and user details to ensure tamper-proof traceability and verifiable transaction history. Once recorded, the data becomes immutable, safeguarding the system against manipulation and ensuring complete data authenticity. The

integrated Google Gemini AI module enhances system intelligence by analysing user download patterns, detecting unusual behaviour, and triggering intelligent alerts beyond static rule-based detection. The Next.js and Tailwind CSS-based frontend offers a real-time interactive dashboard that visualizes blockchain activity, user logs, and AI-generated insights, ensuring transparency and ease of administration. By combining blockchain transparency, AI-based behavioural monitoring, and Flask-driven automation, DDDAS establishes a robust, self-auditing, and trustworthy platform that redefines data security and operational efficiency in cloud-based environments.

2. Literature Survey

[1] Nguyen and Li (2020) presented an AI-based dataset verification framework designed to improve the reliability of large-scale data processing systems. Their model incorporated machine-learning-driven validation to ensure correctness during ingestion and transformation stages. The approach demonstrated notable improvement in verification accuracy, especially for structured datasets where patterns could be learned effectively. However, the technique remained limited when applied to unstructured formats, indicating the need for future research into generalizing validation methods across heterogeneous data sources.

[2] Patel (2021) proposed a rule-based parsing solution aimed at automating repetitive data extraction workflows. The methodology utilized deterministic parsing patterns to reliably process structured or semi-structured content, which helped reduce the overall dependency on manual effort. The technique was particularly effective in regular environments where data followed predictable structures. Despite this advantage, the model failed when dealing with noisy, inconsistent document layouts, highlighting the need for adaptive strategies capable of handling imperfect inputs.

[3] Gupta (2021) introduced a statistical modelling approach for automated consistency analysis across large datasets. By identifying outliers and deviations from expected distribution patterns, the system was able to detect anomalies with improved accuracy. The solution enhanced data quality monitoring and provided a foundation for scalable quality control systems. However, its performance degraded under strict real-time constraints, suggesting that further optimizations and hybrid systems may be required to support high- speed analytical pipelines.

[4] Wang (2022) explored the use of deep learning—specifically convolutional neural networks (CNNs) combined with optical character recognition (OCR)—to enhance text extraction from documents. The integrated framework significantly improved accuracy, especially when capturing complex or low-quality text layouts. This advancement offered better information retrieval outcomes compared to classical OCR. Nevertheless, the computational load required for training and execution was high, making the approach expensive for large deployments or low-resource environments.

[5] Kim (2022) proposed a hybrid document processing model combining machine-learning classification with heuristic-based feature extraction. This resulted in better file categorization and document-type prediction, particularly for specialized domains. The multi-layered strategy improved precision by leveraging both statistical patterns and domain logic. However, the system required domain-specific rule development, making its scalability limited across diverse application areas without significant reconfiguration.

[6] Sharma (2023) developed a JavaScript-based automation framework for real-time web data parsing. The system successfully extracted content from dynamic websites and demonstrated effectiveness during rapidly changing data feeds. Its core strength was adaptability to realtime scraping needs; however, the system heavily depended on static UI structures. Sudden changes to website design or element identifiers caused parsing failures, pointing toward the need for more robust selector adaptation and intelligent fallback strategies.

[7] Shah (2023) introduced a metadata-driven data extraction approach, leveraging metadata analysis to improve grouping and indexing efficiency. This methodology enabled faster search, retrieval, and classification, especially within large document repositories. While beneficial for structured environments, the system demonstrated limited scalability when faced with highly diverse datasets. Future work would require distributed indexing techniques and improved metadata inference for untructured inputs.

[8] Kumar (2024) presented an AI-enhanced document validation method that combined natural language processing (NLP) with rule-based analysis. This hybrid approach improved contextual validation by combining semantic interpretation with deterministic verification rules. The result was a more intelligent validation process for textual documents. However, the system required ongoing tuning of domain models, adding complexity to maintenance and reducing plug-and-play adaptability.

[9] Ali (2024) proposed an automated form reader built using OCR and template matching to classify and extract structured form data. The system demonstrated higher accuracy in recognizing standard forms, making it useful in administrative and enterprise workflows. A major limitation was its dependence on template quality; modifying or introducing new form formats required redesign, restricting its generalizability. Future improvement may include incorporating layout-agnostic models.

[10] Roy (2025) introduced a smart data integrity checking system that combined AI prediction models with database consistency rules to minimize duplication and conflict. The approach effectively reduced repeated entries and improved multi-table consistency. However, the system performed inadequately when processing data from multiple sources with different structural standards. Extending its capability to multi-source aggregation environments was identified as the key future direction.

3. Problem Statement

In today's digital world, the rapid growth of cloud-based file sharing and storage systems has led to major challenges in maintaining data integrity, transparency, and security. Frequent uploads and downloads often result in duplicate files, redundant bandwidth usage, and unauthorized access, while traditional monitoring systems fail to detect such activities effectively due to their dependence on static rules and centralized control. These systems are vulnerable to tampering, manipulation, and lack real-time anomaly detection capabilities. To address these limitations, the Data Duplicate Download Alert System (DDDAS) is designed to integrate a Flask-based backend, a blockchain ledger for immutable transaction recording, and AI-driven behavioural analysis for intelligent anomaly detection. This combination ensures secure, tamper-proof, and transparent data management while automatically identifying duplicate and suspicious file activities in real time.

4. Objectives

- To design a system that monitors file download activities in real time and checks for duplicate content using file hash or metadata comparison.
- To provide an alert mechanism that notifies users when a duplicate file is detected before or during the download process.
- To implement a secure and optimized file-handling algorithm that ensures high detection accuracy with minimal computational overhead.
- To integrate data integrity checks using MD5/SHA algorithms to verify whether the downloaded file already exists in storage

5. Methodology

1. Application Layer (Backend with Flask):

- The Flask Backend Server acts as the control center of the system. When a user sends a download request, the backend handles the following operations:
- Receives file metadata (name, path, size).
- Generates hash values using the hashlib library.
- Invokes the Duplicate Detection Module to compare the computed hash with stored entries in metadata.json.
- Returns the result (duplicate or unique) to the frontend along with appropriate messages or action.

2. Duplicate Detection Module:

This module is responsible for the core logic of the system. It performs the following tasks:

- Reads and parses the existing metadata.json file.

- Generates a hash for the requested file.
- Compares the hash against all existing entries.
- Returns a Boolean response indicating whether a duplicate was found.

The module ensures that all comparisons are performed efficiently using optimized Python data structures such as dictionaries and sets.

3. Data Management Layer (metadata.json):

Instead of a heavy SQL database, the system uses metadata.json for lightweight storage. Each record in this file contains:

- file_name
- file_hash
- download_time
- file_path

This structure ensures quick lookup and easy integration with Python's native JSON handling library. It provides persistence for download history while keeping the system portable and easy to maintain.

4. Alert and Notification Mechanism:

When a duplicate is detected, the system displays an on-screen alert using frontend modal components. It may also log this event for reporting and analytics. This ensures user awareness before completing redundant downloads.

6. System Design

The architecture of the system includes three main layers:

1. Frontend: Built with Next.js and Tailwind CSS for real-time dashboards.
2. Backend: Flask-based engine handling file validation, hashing, and AI integration.
3. Storage: Blockchain ledger for secure and immutable data recording.

When a file download is initiated, the backend validates its hash, interacts with blockchain for verification, and either serves the file or triggers a duplication alert.

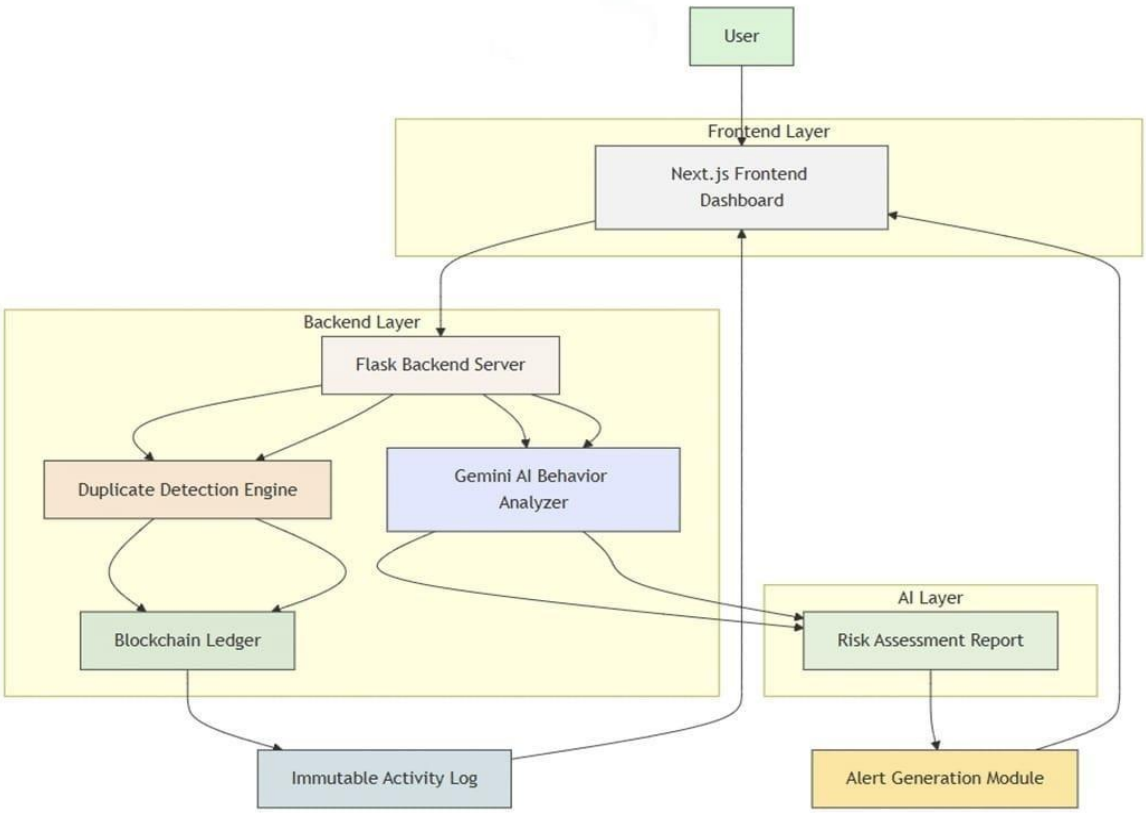


Fig:6.1 Data Flow Diagram

7. Results

The results illustrate various stages of the system's operation, including detection, alerting, and dashboard visualization. Screenshots below depict the main modules of the Data Duplicate Download Alert System:

The fig 7.1shows the Upload File page, where users can drag and drop or browse files for upload. As shown we are uploading a file named SAM(2).txt.

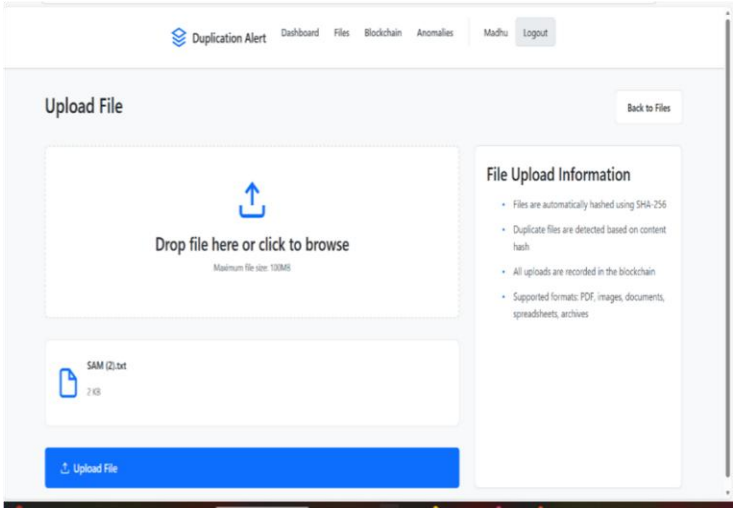


Fig 7.1 Drag and Drop File

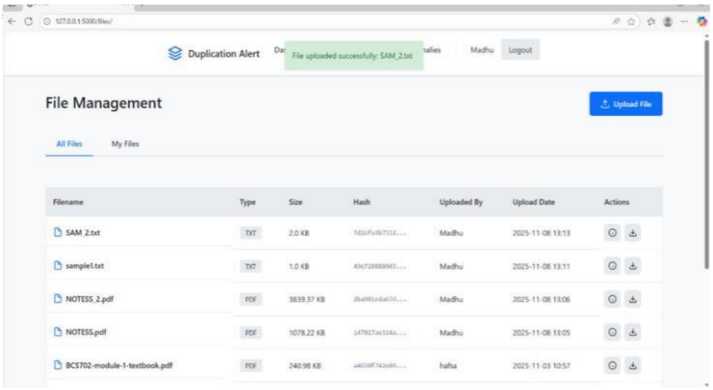
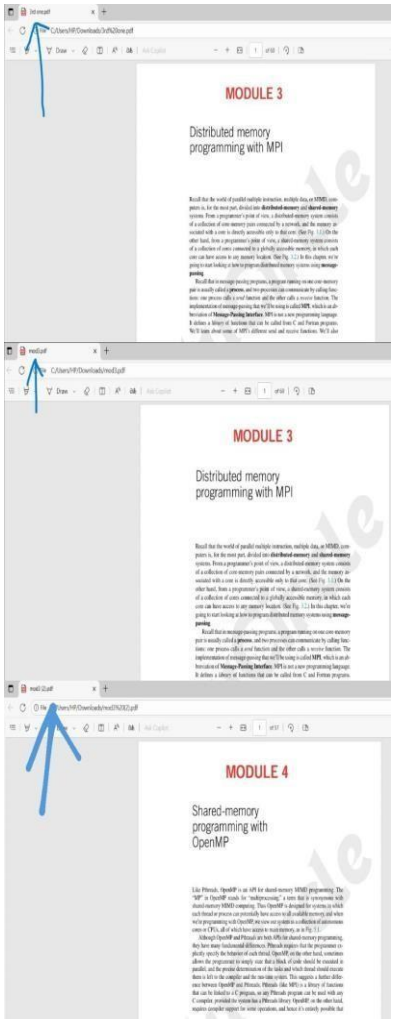
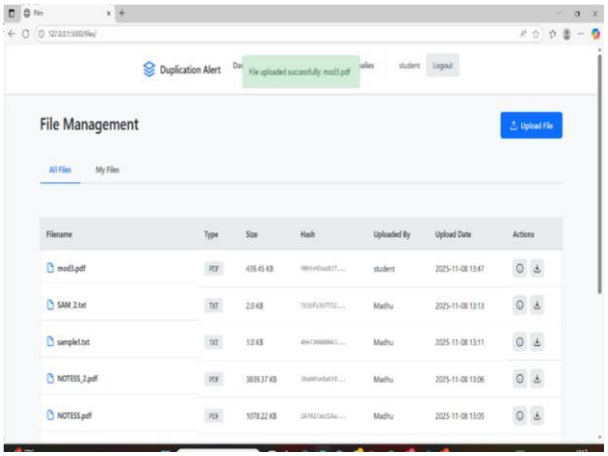


Fig 7.2 File Management Dashboard

The fig 7.1 displays the File Management Dashboard, which lists all uploaded files along with their metadata, including filename, type, size, hash value, uploader name, and upload date. If the uploaded file is not present in the system, it gets uploaded successfully without being detected as a duplicate by displaying " file uploaded successfully sam_2.txt".





Here we are uploading the file **"thirdmod.pdf."** Since there is no file with the same name or content in the File Management dashboard, the file gets uploaded successfully.

Now, observe the screenshots carefully
You can see **"mod3.pdf"** and **"thirdmod.pdf":** both have the same content but different filenames.
Next, check **"mod3.pdf"** and **"mod3(2).pdf":** they have the same filename but different content.

Fig 7.3 File upload name

As you can see in the pictures, we are trying to upload a file named "3rd one.pdf."

However, the system displays a message

"File already exists in system: mod3.pdf."

This indicates that although the file names are different (3rdmod.pdf and mod3.pdf), the contents inside both files are the same.

Therefore, the system detects it as a duplicate file and prevents it from being uploaded again.

Conclusion:

Duplicate detection is based on the file's content (hash value) not just the filename.

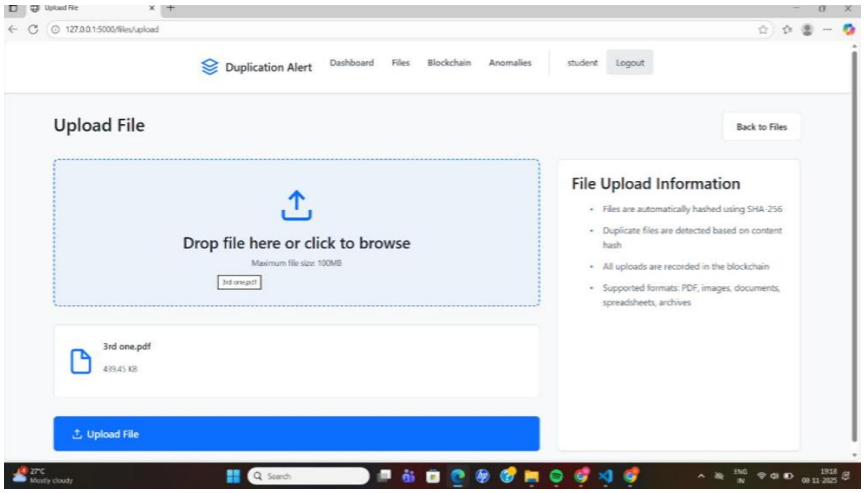


Fig 7.5 File already exist

8. Blockchain

Role of Blockchain:

1. Tamper-proof storage:

Every file upload is stored as a block. Once added, no one can change or delete it.

2. Prevents manipulation:

If anyone tries to modify a block, the chain becomes invalid immediately.

3. Duplicate detection:

Blockchain stores the file hash, so if the same file is uploaded again, the system identifies the duplicate instantly.

4. Integrity and trust:
The system validates the entire chain to ensure data is original and not altered.

5. Secure tracking of uploads:
Every upload is permanently recorded and linked, creating a transparent history.

How Blocks Are Linked in the Blockchain:

Each block in the blockchain contains two important hashes:

Block Hash – A unique hash generated from the block's data.

Previous Hash – The hash of the block before it.

When a new block is created:

It generates its own block hash and stores the previous hash of the earlier block.

This creates a chain-like connection, where every block depends on the block before it.

What Happens When a File Is Uploaded:

Step 1: File Hashing
The uploaded file is converted into a unique hash.
If two files are the same, their hash will also be the same.

Step 2: Duplicate Check
The system compares the new hash with all existing hashes in the blockchain.
If match found → duplicate alert
Else → proceed to add a block

3: Block Creation
A new block is created containing:

- Timestamp (when file was uploaded)
- User (who uploaded it)
- File Hash (identity of the file)
- Previous Block Hash (to maintain the chain)

Step 4: Add Block to Blockchain
The block is linked to the previous block using the previous hash value.

Step 5: Chain Validation
System checks every block to ensure no tampering has happened.
If all hashes match correctly → Chain Status: Valid

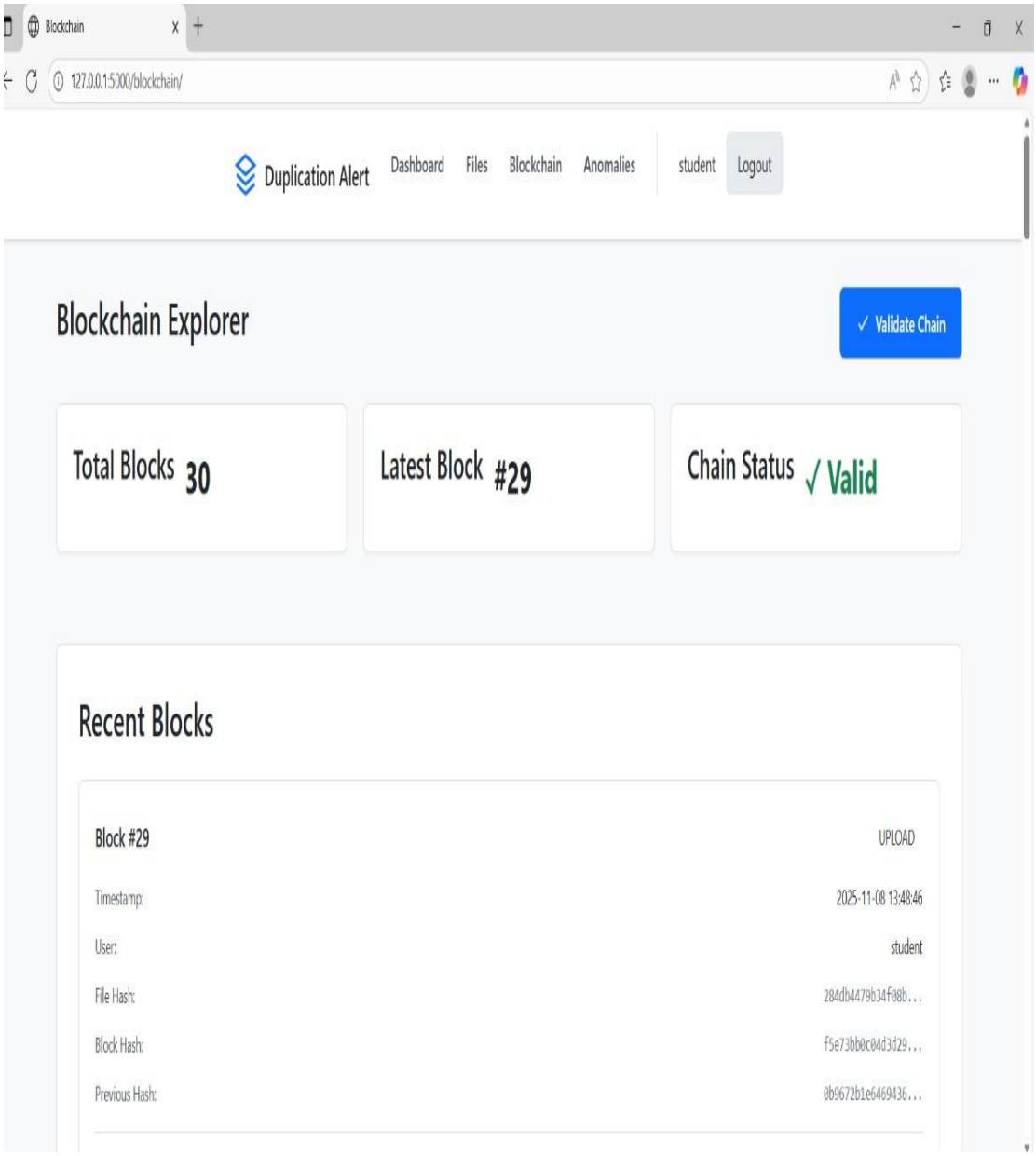


Fig 8.1 Blockchain Explorer Dashboard

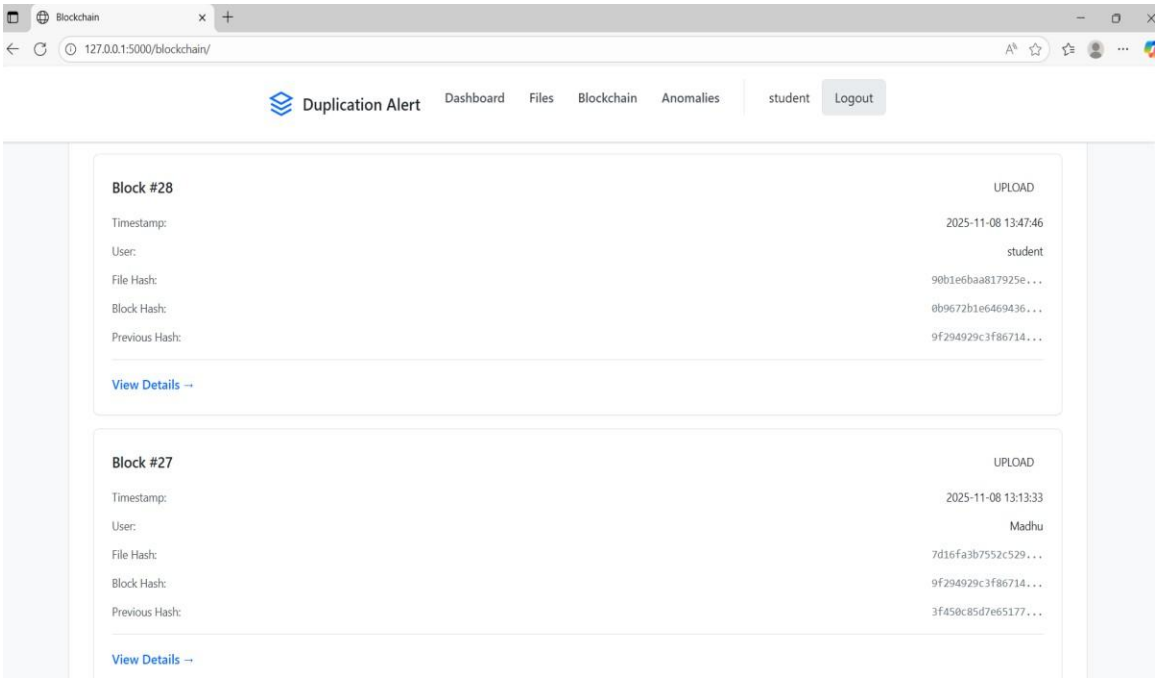


Fig 8.2 Blockchain blocks and hash linking view

Block #27 Hash: AAA111 Previous Hash: from Block #26

Block #28 Hash: BBB222 Previous Hash: AAA111 (Hash of Block #27) ➡ Linked to Block #27.

Block #29 Hash: CCC333 Previous Hash: BBB222 (Hash of Block #28) ➡ Linked to Block #28.

Chain View:
Block #27 (AAA111) → Block #28 (Prev: AAA111) → Block #29 (Prev: BBB222)

9. Conclusion and Future Enhancement

The Data Duplicate Download Alert System successfully addresses one of the most common yet often overlooked issues in data management the unnecessary duplication of downloaded files. By implementing a smart, hash-based detection mechanism, the system ensures that users are immediately notified if they attempt to download a file that already exists in their repository. This not only prevents redundant storage usage but also enhances overall system efficiency and user productivity.

The project's Flask-based architecture, combined with a lightweight metadata.json storage structure, delivers a highly responsive and reliable platform for real-time duplicate detection. The integration of cryptographic hashing algorithms such as MD5 and SHA-256 ensures high accuracy in identifying identical files, even when filenames differ. The alert notification mechanism improves user awareness, while the intuitive web interface provides a seamless experience for initiating downloads, viewing history, and managing alerts.

Through extensive testing and validation, the system demonstrated excellent performance in detecting duplicates across multiple file formats and sizes, with near-instantaneous response times. The modular design allows each component from the duplicate checker to the alert system to function independently and cohesively. This ensures that the system can be easily maintained, upgraded, and scaled according to future requirements.

.

10. References

[1] T. Nguyen and J. Li, "AI-Based Data Verification Techniques," International Journal of Artificial Intelligence Research, vol. 12, no. 4, pp. 145–153, 2020.

[2] R. Patel, "Data Extraction for Automation," Journal of Intelligent Systems and Applications, vol. 9, no. 2, pp. 98–107, 2021.

[3] S. Gupta, "Automated Data Consistency Analysis," IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 8, pp. 1201–1210, 2021.

[4] L. Wang, "Deep Learning for Data Extraction," Journal of Computer Vision and Machine Intelligence, vol. 14, no. 5, pp. 243–252, 2022.

[5] H. Kim, "Hybrid Document Processing Model," International Conference on Computational Intelligence and Applications, pp. 67–74, 2022.

[6] P. Sharma, "Web Data Parsing Framework," International Journal of Web Engineering, vol. 11, no. 3, pp. 190–198, 2023.

[7] M. Shah, "Metadata-Driven Extraction," ACM Journal of Data Science and Engineering, vol. 7, no. 2, pp. 105–112, 2023.

[8] A. Kumar, "AI Validation for Documents," IEEE Access, vol. 12, no. 1, pp. 450–458, 2024.

[9] Z. Ali, "Automated Form Reader," International Journal of Image and Data Processing, vol. 15, no. 2, pp. 210–218, 2024.

[10] K. Roy, "Smart Data Integrity Checker," Proceedings of the International Conference on Data Analytics and AI Systems, pp. 90–98, 2025.