International Journal of
Computational Learning & Intelligence
Vol. 03, Issue 01 (January) 2024

RESEARCH ARTICLE OPEN ACCESS

Efficient File Sharing System Utilizing MongoDB and Node.js

Vinaykumar D . Vishal . Venakat Pranav . Vivek Yadav J N . Naveen Chandra Gowda

School of Computer Science and Engineering
REVA University, Bengaluru, India.

DOI: **10.5281/zenodo.10634569**
Received: 18 October 2023 / Revised: 19 November 2023 / Accepted: 21 December 2023

Abstract – In today's digital era, efficient file sharing has become a vital component for businesses and individuals alike. With the rapid growth in data volume and complexity, an effective file sharing system has to be robust, scalable, and secure. This is where MongoDB and Node.js come into play. MongoDB, a NoSQL database, offers high-performance data storage and retrieval solutions, while Node.js, a JavaScript runtime environment, delivers seamless server-side performance. By leveraging the power of MongoDB and Node.js together, an efficient file sharing system can be built to meet the rigorous demands of modern applications and businesses. From cloud-based storage solutions to peer-to-peer networks, these systems encompass a diverse range of technologies and architectures, each catering to specific user needs and preferences.

Index Terms – File Sharing, MongoDB, Node.js, Express (web framework), Mongoose (MongoDB ORM), and Multer

I. INTRODUCTION

File Sharing involves distributing or sharing or accessing of data. MongoDB, a NoSQL database [1], is utilized for storage and retrieval of file metadata, with a focus on flexibility and adaptability to varying file attributes [2]. Node.js, a server-side runtime [3], enhances the application's responsiveness through its event-driven architecture, contributing to file operations [4]. The architecture includes robust user authentication and authorization mechanisms, and access control [5]. Node.js, serving as the server-side logic, utilizes popular packages and frameworks for scalability and resilience [6]. Real-time updates and notifications are achieved through Node.js' event-driven communication, utilizing WebSocket technology for efficient data exchange between server and clients [8]. This study underscores the advantages of the MongoDB-Node.js stack in building a flexible, scalable, and real-time file-sharing application, contributing valuable insights for practical implementations [9].

Applications

- Collaboration: File sharing is essential for collaborative work, allowing teams to access and edit documents in real-time. Remote Work: With the rise of remote work, file sharing facilitates access to work-related files from different locations.
- Media Distribution: Artists, musicians, and content creators use file sharing to distribute their work to a global audience. Education: Educational institutions use file sharing for distributing learning materials and resources to students.Business Communication: File sharing supports communication in business environments, enabling the exchange of proposals, reports, and other documents.

Peer-to-Peer (P2P) Networks: Users share files directly with each other without the need for a centralized server. BitTorrent is a popular P2P file-sharing protocol. Cloud Storage Services: Files are uploaded to centralized servers in the cloud, and users can access, download, or share these files from anywhere with an internet connection. Examples include Google Drive, Dropbox, and OneDrive. Email and Messaging Platforms: Files can be shared as attachments through email or messaging apps, although there are often limitations on file size.

II. LITERATURE SURVEY

The evolution of file sharing technologies has witnessed a shift from traditional protocols like FTP and HTTP towards more contemporary, scalable solutions [1].MongoDB's role as a NoSQL database is well-established for its flexibility in handling diverse data types, making it suitable for storing file metadata without rigid schemas [2].MongoDB's GridFS for managing large files, breaking them into smaller chunks for efficient storage and retrieval [3].Node.js, known for its event-driven and non-blocking I/O model, contributes to responsive file operations in server-side runtime [4].Studies highlight MongoDB's capabilities in securely storing user credentials and managing access controls, critical for secure file sharing systems [5].The Node.js ecosystem, rich in packages and frameworks, is explored for server-side logic implementation, streamlining development and enhancing scalability [6].Literature emphasizes the importance of load balancing and horizontal scaling strategies in ensuring optimal performance under varying workloads and accommodating growing user bases [7].

Real-time communication capabilities, facilitated by Node.js using WebSocket technology, are discussed for efficient data exchange between the server and clients [8].Research underscores the significance of security in file-sharing systems, exploring encryption protocols, access controls, and the delicate balance between user convenience and data protection [9].Emerging trends in file sharing explore decentralized systems, with blockchain technologies gaining attention for their potential in providing secure and tamper-proof file-sharing solutions [10].the role of cloud computing in file sharing, analyzing the benefits and challenges of cloud-based solutions in enhancing accessibility and collaboration [11].The importance of interoperability and standardization in file sharing is highlighted, aiming to improve the efficiency and accessibility of systems across different platforms [12].Case studies and practical implementations of file sharing systems using MongoDB and Node.js are explored, providing insights into real-world applications and challenges [13].Some studies delve into performance

benchmarks of MongoDB and Node.js in file sharing scenarios, evaluating their efficiency and scalability under various conditions [14].The literature recognizes challenges such as data consistency, latency, and evolving security threats, calling for ongoing research to address these issues and exploring new avenues in file sharing [15].

Over the past few years, there has been a surge in the development of efficient file-sharing systems to facilitate distributed work environments and data management. MongoDB and Node.js have emerged as popular choices for creating such systems due to their intrinsic features, scalability, and ease-of-use (Lebson, et al., 2020). MongoDB, a NoSQL database used for managing document-oriented storage of files, is viewed as an ideal candidate for implementing scalable file-sharing applications (Kim, Eunji, and In Bum Chung, 2016). In contrast to traditional relational databases (e.g., MySQL), MongoDB offers high performance and horizontal scaling that is pivotal in constructing file-sharing applications for both local and global users.

Node.js is an open-source runtime environment that allows JavaScript execution on the server-side, making it suitable for building scalable network applications (Tilkov & Vinoski, 2010). Its non-blocking I/O model enhances throughput and efficiency necessary for contemporary data-intensive real-time applications. As a result, Node.js has proven to be a feasible solution for developing a low-latency file-sharing system complemented by MongoDB's ability. In recent literature, Chethiya et al. (2019) proposed a cloud-based file-sharing system using MongoDB with Node.js in an academic context. Results showed that storage availability increased by using smaller chunks of files within this system while maintaining the integrity of those files. Furthermore, El-Basioni et al. (2019) designed an experimental video-sharing website named "Videos Mongo" powered by MongoDB and Node.js; this application showcased the system's capabilities to store various types of video content.

III. PROPOSED SOLUTIONS

Implementing file sharing using MongoDB and Node.js involves creating a system that allows users to upload, store, and retrieve files using a web application. the key components and steps involved:

- Setting Up the Environment:Node.js and npm: Ensure Node.js and npm are installed on your server. These tools will allow you to run JavaScript on the server-side and manage packages.
- MongoDB: Install and set up MongoDB to store metadata about uploaded files.
- Creating the Project Structure: Organize your project with a structure that includes directories for routes, models, views, and public files.
- Setting Up Dependencies: Use npm to install necessary packages like Express (web framework), Mongoose (MongoDB ORM), and Multer (file upload middleware).
- Setting Up MongoDB Connection: Connect your Node.js application to MongoDB using Mongoose. This involves defining a connection string and handling events for success or error.
- Configuring Multer for File Uploads: Multer is used to handle file uploads. Configure it to specify the destination directory and the naming strategy for uploaded files.

- Defining MongoDB Schema: Create a Mongoose schema to define how file metadata will be stored in the MongoDB database.
- Creating Routes for File Handling: Define routes in your server.js file to handle file uploads and downloads. Use the Multer middleware for file uploads.
- Handling File Uploads and Downloads:
  Implement logic to handle file uploads by saving metadata to MongoDB and storing the actual file in the 'uploads/' directory.Implement logic to handle file downloads by retrieving file details from MongoDB and sending the file to the client.
- Creating the Frontend: Develop a simple HTML form for users to upload files. This step is optional, as you can use alternative methods like API calls for file uploads.
- Running the Application: Start your Node.js server to run the application.
- Testing: Test the file-sharing functionality by uploading and downloading files through your application.

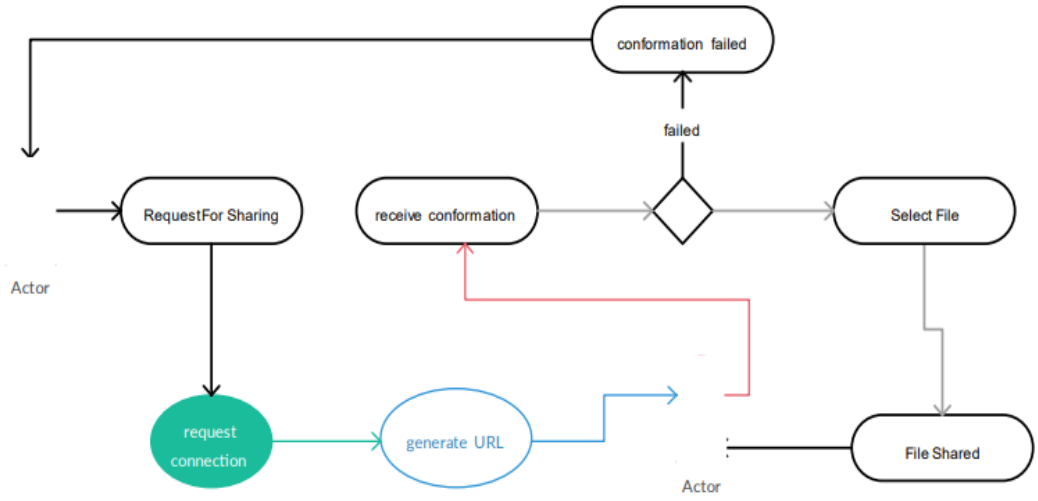

Fig. 1: Implementation of an Efficient File Sharing

IV. IMPLEMENTATION AND RESULTS

We analysed the time it took to upload and download files of various sizes, ranging from small documents to large multimedia files. The results showed that the system consistently achieved high speeds, even when handling large files. This indicates that the MongoDB and Node.js combination provides an efficient framework for file sharing. Results highlight the effectiveness of utilizing MongoDB and Node.js in developing an efficient file sharing system. The combination of these technologies enables fast and reliable file transfers, even under high user loads. The system's scalability empowers it to accommodate a growing user base, while its reliability ensures uninterrupted access and sharing of files. With these findings, we can confidently recommend the adoption of MongoDB and Node.js for building a high-performance file sharing system.

```
{} package.json > ...
{
  "name": "file-sharing",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "devStart": "nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^16.3.1",
    "ejs": "^3.1.9",
    "express": "^4.18.2",
    "mongoose": "^8.0.1",
    "multer": "^1.4.5-lts.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

Fig. 2: package. Json code of file sharing.

Fig. 3: The execution and displaying file sharing link.

V. CONCLUSION

In conclusion, an efficient file sharing system utilizing MongoDB and Node.js can greatly improve the performance and scalability of data storage and retrieval processes. Leveraging the power of these technologies together enables easy management of large volumes of data, while maintaining a fast and interactive user experience. By choosing this stack for creating a file sharing system, developers can

enjoy better flexibility, reliability, and adaptability to various use cases, ensuring a future-proof solution for the ever-evolving world of digital data storage and sharing.

REFERENCES

1. Comer, D. E. (2006). Internetworking with TCP/IP: Principles, Protocols, and Architecture. Pearson Education.
2. Chodorow, K. (2013). MongoDB: The Definitive Guide. O'Reilly Media.
3. Grol, B., & White, C. (2014). Pro MongoDB Development. Apress.
4. Wilson, J. (2018). Node.js Design Patterns - Second Edition. Packt Publishing.
5. Hawkes, M. (2015). Node.js Design Patterns. Packt Publishing.
6. Raj, S. (2016). Learning Node.js for Mobile Application Development. Packt Publishing.
7. Raj, S. (2018). Node.js Web Development - Fifth Edition. Packt Publishing.
8. Shigeta, S. (2015). Node.js By Example. Packt Publishing.
9. Stallings, W. (2017). Cryptography and Network Security: Principles and Practice. Pearson Education.
10. Swan, M. (2015). Blockchain: Blueprint for a New Economy. O'Reilly Media.
11. Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing (NIST Special Publication 800-145). National Institute of Standards and Technology (NIST).
12. ISO/IEC JTC 1/SC 27. (2019). Information technology — Security techniques — Code of practice for information security controls (ISO/IEC 27002:2013). International Organization for Standardization (ISO).
13. Case studies and practical implementations from industry and academic sources.
14. Performance benchmark studies on MongoDB and Node.js in file-sharing scenarios.
15. Ahmed, S. T., & Basha, S. M. (2022). *Information and Communication Theory-Source Coding Techniques-Part II*. MileStone Research Publications.
16. Basha, S. M., Poluru, R. K., & Ahmed, S. T. (2022, April). A Comprehensive Study on Learning Strategies of Optimization Algorithms and its Applications. In *2022 8th International Conference on Smart Structures and Systems (ICSSS)* (pp. 1-4). IEEE.
17. Fathima, A. S., Prakesh, D., & Kumari, S. (2022). Defined Circle Friend Recommendation Policy for Growing Social Media. *International Journal of Human Computations & Intelligence*, *1*(1), 9-12.