J-COSINE (Journal of Computer Science and Informatics Engineering)
Vol. 9, No. 2, December 2025 E-ISSN:2541-0806
Accredited Sinta-4 by RISTEKDIKTI Decree No. 79/E/KPT/2023 P-ISSN:2540-8895

Development of a Web-Based System for Scientific Publication Data Management Using NoSQL Architecture

Luthfiyyah Az Zahro*, Heri Wijayanto, Wirarama Wedashwara
Dept Informatics Engineering, Mataram University
Jl. Majapahit 62, Mataram, Lombok NTB, INDONESIA
Email: fiyyah.zahro@gmail.com, [heri, wirarama]@unram.ac.id

Corresponding Author

Abstract **This study developed a web-based system for managing scientific publication data using a NoSQL architecture (MongoDB) and an Extract, Load, Transform (ELT) approach. The website allows users to easily access, manage, and visualize data through an interactive interface built with React.js on the frontend and Express.js on the backend. The system supports data input via CSV uploads and manual form submissions, with all data stored in MongoDB Atlas. Data transformation is handled using the Aggregation Pipeline, which performs normalization, validation, and upsert operations to ensure consistency and prevent duplication. Functional testing was conducted on 503 real publication records, while performance testing used 1,000 dummy records. The results show that the system can support up to 20 concurrent users with an average response time of 4,792 ms and no errors. Overall, the website offers a flexible and integrated solution for managing scientific publications in higher education institutions.**

Key words**: Scientific Publication Management, Website, NoSQL, MongoDB, ELT.**

I. INTRODUCTION

With the development of technology, big data has become an important component in information management and analysis. Initially, big data was considered a challenge related to storage capacity in data visualization. However, today the concept has evolved into a term that describes data that is very large, diverse, and unstructured, making it difficult to manage using traditional methods [1]. This development requires new technologies and methods to manage and analyze data on a large scale. Big data has five main characteristics, known as the 5Vs: volume, variety, velocity, veracity, and value [2]. Among these five characteristics, variety is one of the greatest challenges because it refers to the diversity of data structures and formats, also known as heterogeneous data.

Heterogeneous data is information that comes in various formats, locations, and structures. These differences can include structured tables, text documents, JSON files, and multimedia content such as images and videos. In a university setting, heterogeneous data encompasses academic, administrative, financial, and scientific publication data. A key focus of this discussion is scientific publication data, which continues to rise each year. Publication data, such as journals, final projects, proceedings, and research reports, is typically stored in institutional repositories. These repositories manage a variety of scientific documents, including theses, journals, and other materials [3]. Differences in the formatting of names, titles, affiliations, and publication years can increase the necessity to integrate heterogeneous information.

Data integration in heterogeneous systems refers to the process of combining data from different format, strucutures, and storages mechanisms [4]. Heterogeneous data integration is essential to support analytics, real time decision making, and machine learning applications [5]. However, it is not enough to intgrate heterogeneous publication data. Another thing that needs to be considered is how to maintain data consistency after integration. Without consistency, the integrated data will cause new problems such as duplication, entry conflicts, or information redundancy [6]. This is directly related to characteristics of veracity in Big Data, namely maintaining accuracy and trust in the data used.

Meanwhile, the increasing volume and complexity of publication data demands a management system that has good scalability performance. Traditional system such as Extract, Transform, Load (ETL) or relational databases (SQL) that are batch-oriented, have limitations, both in terms of data formats [7]. These systems also tend to be less flexible in handling data changes and growth in web-based applications. On the other hand, approaches using NoSQL architecture are starting to be widely applied. NoSQL systems are designed to handle unstructured data, support heterogeneous data integration, and are capable of horizontal scaling to accommodate the need for high performance [5], [8].

This research aims to design and implement a heterogeneous scientific publication data integration process within the scope of higher education. The integration process was carried out using MongoDB Atlas to store and manage publication data. MongoDB was chosen because it is a document-based NoSQL database that supports dynamic schemas, semi-structured data storage, and data transformation through aggregation pipelines [9]. MongoDB was chosen because it can store

data in a flexible document format and supports semi-structured formats such as JSON, which is suitable for diverse scientific publication metadata. Unlike Cassandra, which is specifically designed to handle large-scale time-series data and systems with high writing requirements, MongoDB is more suitable for applications that require schema flexibility and integration with user interfaces.[10].

Unlike previous studies that emphasized the integration of homogeneous structures using traditional ETL flows and relational databases, this study adopted an ELT (Extract, Load, Transform) approach combined with a NoSQL architecture to directly manage unstructured and semi-structured metadata. Meanwhile, previous literature has mostly discussed NoSQL scalability or metadata transformation separately. This study presents an integrated approach that combines the flexibility of heterogeneous data integration with system performance evaluation, particularly in terms of scalability and data consistency as data volume continues to increase.

The novelty of this study lies in the application of a comprehensive (end-to-end) integration process to various publication metadata formats, using flexible and efficient document modeling. Thus, this research contributes to the development of a scientific publication management system that is not only flexible and integrated but also reliable in performance. Additionally, this research aligns with current trends in academic data management and has the potential to serve as a foundation for the development of future academic information systems.

II. Literature Review

Various studies have been conducted on developing information systems for managing scientific publication metadata, covering aspects such as data storage, transformation, and user interfaces. Key challenges in such systems include inconsistent metadata structures, the need for flexible input formats, and efficient validation and deduplication processes. Some studies have focused on institutional repository design, the use of flexible databases like MongoDB, and automated data transformation workflows. These works serve both as references and evaluation benchmarks in designing a web-based scientific publication management system that prioritizes efficient data handling and high data quality.

Silva et al. [11] introduced EasyBDI, a logic-based data integration system capable of schema mapping and query transformation across heterogeneous data sources. While the system supports diverse input formats such as manual entry and CSV uploads, it does not address crucial issues like metadata validation and deduplication in the context of scientific publications. Complementing the schema flexibility aspect, Cabral et al. [12] proposed a conceptual modeling approach to generate schema-independent queries for MongoDB. This allows for flexible metadata access across varying document structures, although it lacks mechanisms for data quality enhancement such as validation or normalization.

Meanwhile, Reza et al. [3] emphasized the importance of web-based institutional repository platforms through the implementation of DSpace and the Dublin Core standard. While effective in managing academic archives, this approach remains limited to relational models and does not leverage schema flexibility or transformation automation as enabled by ELT. Locally, Anthony and Tony [13] developed a faculty publication management system using a manual approach with PostgreSQL. Although it improved administrative efficiency, the system lacked bulk data processing and automated transformation making ELT implementation in this study a distinctive advantage.

Regarding data quality and consistency, Ye et al. [14] proposed an entity resolution method using blocking and matching techniques to handle duplication. This is particularly relevant for scientific publication systems. In this study, deduplication is implemented using upsert operations and key attribute validation in MongoDB. Furthermore, MongoDB's support for flexible structures and high data loads is reinforced by findings from Mohan et al. [15], who demonstrated NoSQL's performance advantages in analytical scenarios. These findings support the decision to use MongoDB as the system's storage backbone. Additionally, Alflahi et al. [16] suggested transaction management strategies in MongoDB to maintain data integrity. Techniques such as separating read-write operations, using upserts, and validating through pipelines are also adopted in this system to support efficient and consistent management of publication metadata at scale.

TABLE I. Table of SOTA

Title	Method	Result	Limitations
Logical Big Data Integration and Near Real-time Data Analytics	Logical schema mapping, query rewriting	Supports integration from various data formats and structures	Does not handle deduplication or metadata validation
Enabling Schema Independent Data Retrieval Queries in MongoDB	Entity conceptualization, MongoDB query abstraction	Provides flexible access to data across documents with different schemas	Does not address metadata validation or quality
Perancangan dan Implementasi Institutional Repository dengan Metada Dublin Core	Waterfall model, DSpace, Dublin Core, PostgreSQL	Manages institutional archives using standardized metadata structure	Not flexible with heterogeneous input and lacks automated data transformation
Perancangan Aplikasi Manajemen Data Publikasi dan Penelitian	Web-based system (Node.js, jQuery), PostgreSQL	Improves administrative efficiency in managing publication data	Does not support bulk input and has not implemented the ELT transformation approach
Multi-Source Data Repairing: A Comprehensive Survey	Blocking-based matching, entity resolution framework	Reduces data duplication across entities through matching	Not integrated into scientific publication management systems

Evaluating NoSQL Database for OLAP Workloads: A Benchmarking Study of MongoDB, Redis, Kudu, and ArangoDB	Benchmark testing on large workloads	MongoDB efficiently handles big data workloads	Not focused on the domain of scientific publication metadata
Enhancement of Database Access Performance by Improving Data Consistency in Non-Relational Database System (NoSQL)	Upsert, pipeline validation	Improves consistency and efficiency in bulk data processing	Does not address integration of data from diverse publication input formats

Table I presents a summary of various approaches from previous research that are relevant in developing a scientific publication data management system. Each study has its own focus, such as schema mapping, query flexibility, system performance optimization, as well as data consistency and validity. Although these studies contribute significantly, most still have limitations, particularly in the direct application of flexible scientific repository systems, handling metadata entry duplication, and performing dynamic data transformation from various input formats. Therefore, this research proposes a more integrated approach through the development of a MongoDB-based system that supports the ELT (Extract, Load, Transform) process, duplication detection and handling using upsert techniques and aggregation pipelines, and accommodates data input both via CSV file uploads and manual form entries. This strategy is expected to address the key challenges that have not been fully resolved by previous studies.

III. Research Methodology

This research was conducted using the Research and Development (R&D) approach method with the aim of developing and evaluating a scientific publication data integration system using NoSQL architecture. To clarify the research flow, Fig. 1 presents the process sequence from the start to finish in a structured and systematic manner.

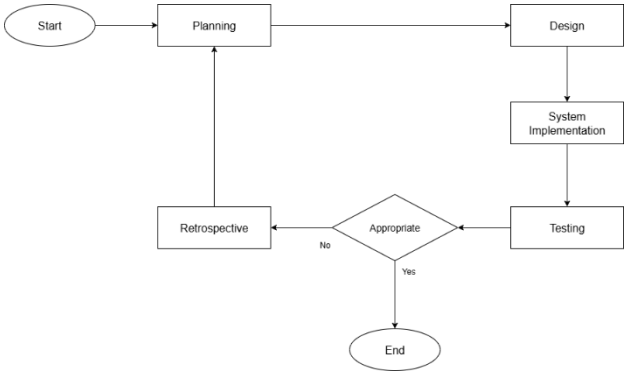

Fig. 1. Research Flow

Fig. 1 shows the flow of research stages using an iterative approach, which allows for iterative evaluation and improvement of the system until the results obtained meet the research objectives. The research process starts from planning, design, system implementation, and testing. The test results carried out at the testing stage will be evaluated in the appropriate. If it is not appropriate, the improvements will be made at the retrospective stage. If it is appropriate, the process will be stopped. The flow illustrates the XP principle that emphasizes continuous improvement.

A. Planning

The first stage is the planning stage, which includes the identification of needs and the formulation of problems. At this stage, a literature review of NoSQL architecture, the ELT method, and heterogeneous data integration techniques was conducted. Furthermore, this stage also determines the tools and technologies, such as MongoDB, React, and Express.js, to ensure the system development process runs effectively and meets its objectives.

B. Design

The second stage after planning is system design. At this stage, an overall architecture is designed that includes the frontend, backend, and database.

B.1. Use Case Diagram

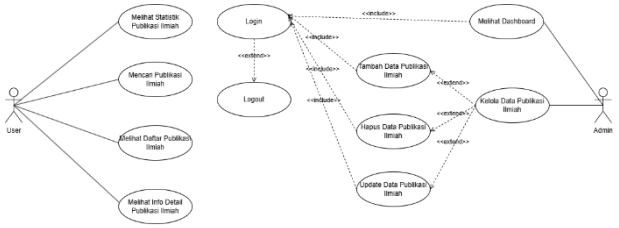

Fig. 2. Use Case Diagram

Fig. 2 illustrates the interaction between users and the system in managing scientific publication data. There are two main actors, namely Admin and User. Admins must first log in to acces all system features, such as viewing dashboards, adding data, editing, and deleting publication data. Meanwhile, User have limited access rights, which can only view statistics, search for publications, view lists and details of scientific publications without the need to log in.

B.2. Flowchart Diagram

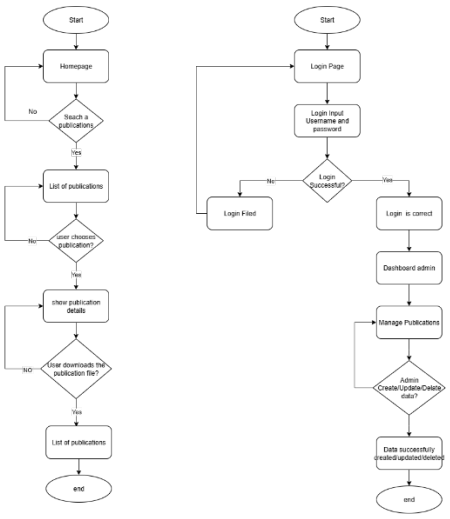

Fig. 3. Flowchart Diagram

As illustrated in Fig. 3, the flowchart on the left represents the interaction flow for general users. The process begins on the homepage, where users can search for publications using specific keywords. The system will then display a list of relevant publications. When a user selects an entry, the system presents the complete details of the publication. If available, users can also download the associated publication file before returning to the publication list. This flow does not require authentication and is designed to support open and user-friendly data exploration.

On the other hand, the flowchart on the right outlines the process flow for users with admin privileges. The admin begins by logging into the system, and upon successful authentication, is directed to the dashboard. From there, the admin can manage publications by adding, updating, or deleting data. After any action is performed, the system displays a confirmation message indicating that the operation was successful. This flow is designed to maintain data integrity and consistency by enabling direct management of the MongoDB database through backend API interactions.

B.3. Activity Diagram

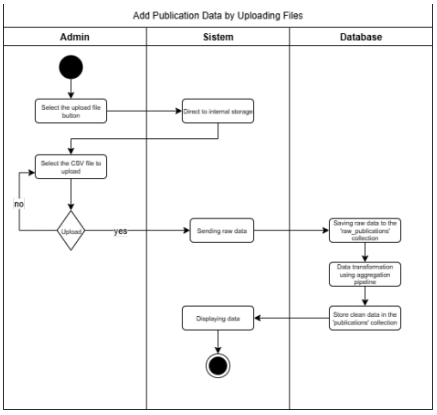

Fig. 4. Activity Diagram

Fig. 4 is one of the activity diagrams in this study, illustrating the process flow of adding publication data through the file upload feature by the admin. This diagram describes the interaction between three main components Admin, System, and Database in handling the upload, storage, and transformation of publication data.

B.4. Data Schema Structure

```
{
  "_id": "ObjectId",
  "ID_Jurnal": "String",
  "Judul_Jurnal": "String",
  "Nama_Jurnal": "String",
  "URL_Jurnal": "String",
  "File_Jurnal": "String",
  "Tahun_Jurnal": "Number",
  "Bulan_Jurnal": "Number",
  "Nama_Prodi": "String",
  "Data_Personil": {
    "ketua": "String",
    "anggota": [
      "String",  // anggota1
      "String",  // anggota2
      ...
    ]
  }

}
```

The data structure of scientific publications is stored as JSON documents in MongoDB, which includes metadata information of scientific publications. For personnel data, it is organized in a nested manner through the 'Data_Personil' object, which contains the name of the chairperson and a list of author members. This format takes advantage of MongoDB's flexibility to handle structured and semi-structured data, and supports efficient searching and data transformation.

B.5. System Architeture Diagram

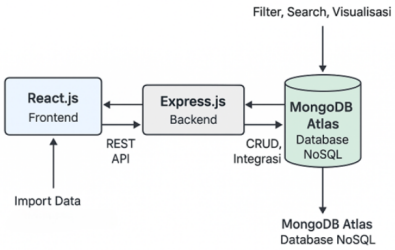

Fig. 5. System Architecture Diagram

Based on Fig. 5, this system architecture consists of three main components, namely React.js as the frontend, Express.js as the backend, and MongoDB Atlas as the NoSQL database. React.js is used to input manual data and upload CSV files, then the data is sent to the backend via REST API. The backend with Express.js processes, validates, and saves the data to MongoDB. The data transformation process is performed using Aggregation Pipeline after the data enters collection. The systems is designed to effectively manage heteregeneous data and support efficient searching, filtering, and visualization of data.

C. System Implementation

The system implementation phase is designed to support the efficient and structured management and processing of scientific publication data from various study programs. The system consists of three main components: the frontend, the backend, and the NoSQL database. The frontend, built with React.js, offers two methods for data input: uploading a CSV file and manually entering data through a form. Once submitted, the data is sent to the backend in JSON format for further processing.

The backend, developed using Express.js, acts as a bridge between the user interface and the database. It is responsible for processing the received data and storing it in the initial raw_publications collection within MongoDB Atlas. In addition, the backend is equipped with specific endpoints that trigger the data transformation process once loading is complete.

MongoDB Atlas serves as the NoSQL database solution, supporting the storage of semi-structured data. The raw_publications collection temporarily holds raw data in its original format. The transformation process is designed to run automatically using the MongoDB Aggregation Pipeline. This process involves several stages: $project reshapes the metadata structure, $addFields adjusts attribute formats, and $match filters for valid records. The final stage, $merge, saves the transformed data into the target publications collection. To avoid duplication, the system applies an upsert logic based on a combination of key attributes.

D. Testing

System testing was divided into two categories: functional and non-functional testing. Functional testing was carried out using the black-box method to ensure that all features operated as expected, including manual data input, CSV file uploads, search, filtering, data visualization, and CRUD operations. Meanwhile, non-functional testing focused on evaluating the system's performance and scalability. This involved simulating large-scale data processing to assess stability, response time, and data consistency throughout the Extract, Load, and Transform (ELT) process. Scalability was tested using stress and spike testing methods, while data consistency was evaluated through duplication checks.

E. Retrospective

The retrospective stage is an evaluation process to review the results of work in one development cycle, identify things that went well and thing that need to be imporved, and formulate improvement steps for the next iteration.

IV. Results and Discussion

This section presents the results of the development process for the scientific publication data management website, including the implementation of the user interface, core features, and data processing mechanisms that enable efficient metadata management. In addition, it discusses the outcomes of both functional and non-functional testing to evaluate the system's overall performance and reliability.

A. System Implementation Results

After the system components were fully developed, the implementation phase began by building a web interface integrated with both the backend and the database. The interface is designed to allow users to efficiently access, search, filter, and manage scientific publication data through an interactive and responsive platform.

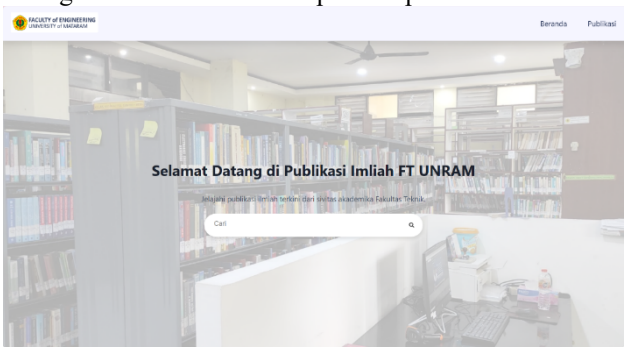

Fig. 6. Homepage

Fig. 6 shows the homepage of the scientific publication management website for the Faculty of Engineering at the University of Mataram. This page serves as the main entry point for users to access publication information. It features a header with navigation menus, including "Home" and "Publications," as well as a search function that allows users to find publications using specific keywords. The interface is designed to be simple and intuitive, making it easy to use for a wide range of users, including lecturers, students, and administrative staff.

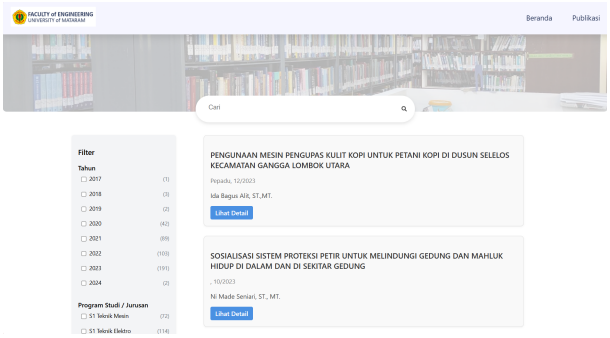

Fig. 7. Search, Filter, and List Publications

In Fig. 7, the search, filter, and list features of publications are shown, which can be accessed by all users without the need to login. Users can enter certain keywords in the search field, such as publication title or author name, to find relevant data. In addition, filter options are available based on study program and publication year, which helps users filter data more specifically.

E-ISSN:2541-0806
P-ISSN:2540-8895

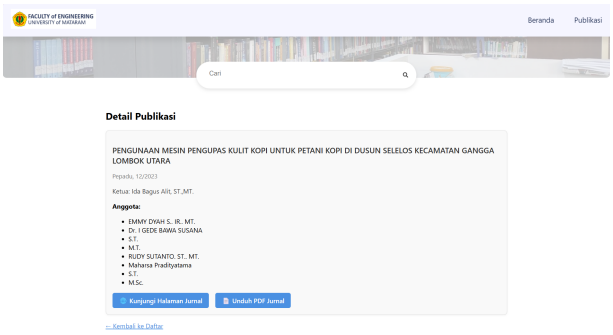

Fig. 8. Detail Publications

Based on Fig. 8, the publication details page is displayed, which contains the complate metadata of a scientific publication entry. This page provides a more comprehensive view than the publication list, allowing users to review the content and context of a publication before downloading or referring to it. All data is dynamically pulled from MongoDb database based on the selected publication ID.

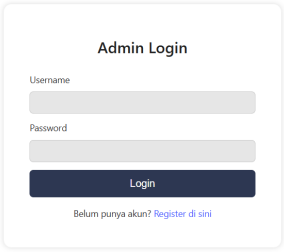

Fig. 9. Login Page

Fig. 9 shows the login page used by the admin to access the publication data management feature. Login is done by entering the username and password that has been registered previously. After a successful authentication process, the admin will be directed to the dashboard page and features such as add, change, and delete publications are available. The login page serves as an access restriction, so that only users with certain authorities can make changes to publication data.

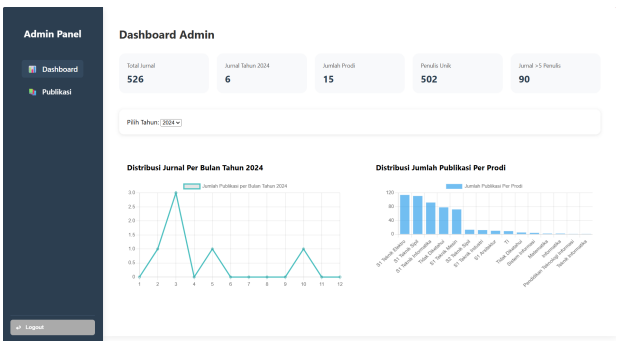

Fig. 10. Dashboard Admin

The admin dashboard display in this system is designed to facilitate the monitoring and analysis of scientific publication data stored in the database. The dashboard displays a number of key indicators such as total journals, number of journals in 2024, number of study programs, number of unique authors, and total publications from the five study programs with the highest contributions. At the top, there is a year filter feature that allows admins to costumize the data display based on the year of publication. Data visualization is presented through two graphs, namely a line graph illustration the distribution of journals per month in the selected year and a bar graph showing the distribution of publication counts per academic program. This visualization aims to provide a clear overview of scientific publication trends and the contribution levels of each academic program within the Faculty of Engineering.

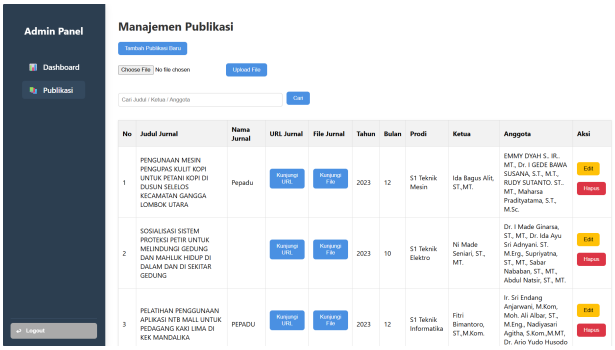

Fig. 11. Manage Publications Data

As shown in Fig. 11, the publication data management feature is accessible only to admin users after successful login. Through this page, admins can perform three main operations: adding, editing, and deleting scientific publication records stored in the MongoDB database. There are two methods for adding data. The first is by filling out a form containing multiple metadata fields. The submitted form data is sent in JSON format to the backend, where it follows the same processing flow as a CSV upload: it is first stored in the raw_publications collection, then automatically transformed through the ELT process using the MongoDB Aggregation Pipeline. This ensures that both manual and bulk inputs result in consistent and valid final data.

The second method allows admins to upload a CSV file containing multiple publication records at once. In this approach, the system automatically stores the raw data in a temporary collection and immediately triggers the transformation process on the backend. This process includes text format normalization, required attribute validation, restructuring of the Data_Personil field, and final storage into the publications collection using an upsert mechanism to prevent duplication. This approach follows the Extract, Load, Transform (ELT) principle, where data processing occurs after loading, making it more flexible in handling varying input structures.

An example of the final transformed data is shown in the document snippet in Fig. 12, which is stored in the publications collection in MongoDB. This JSON format

represents the normalized and validated metadata of a scientific publication successfully processed by the system.

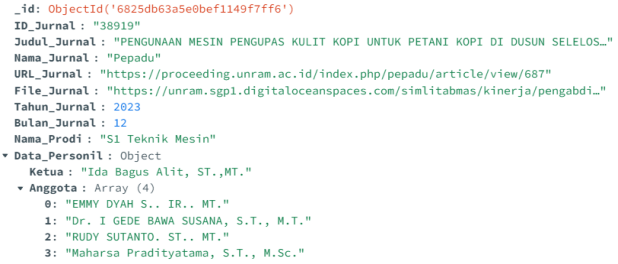

Fig. 12. Result of Database

In addition to adding new records, admins can also edit existing publication data. The system displays the current data in an editable form, allowing admins to make updates directly and save the changes back to the database. Any modifications are immediately reflected in the user-facing publication list. For entries that are irrelevant or contain incorrect information, a delete feature is available, enabling admins to remove specific publications. To prevent accidental deletions, the system prompts a confirmation message before proceeding. All data management operations are designed to interact directly with the MongoDB NoSQL database through backend APIs, ensuring efficient and consistent processes for storing, updating, and deleting data.

B. Testing

System testing is carried out to ensure that all features have run according to function and the system meets the required quality standards. Testing is done thoroughly on the functional (features) and non-functional (performance in scalability and data consistency).

B.1. Functional Testing

Functional testing uses the Black Box testing method, which is a testing method based on input and output without paying attention to the internal processes of the system, which aims to ensure that each feature on the system work correctly according to user needs. That results of functional testing are shown in Table III.

TABLE II. RESULTS OF FUNCTIONAL TESTING

No	Featured Tested	Input/Test Case	Expected Output	Remarks
1.	Admin Login	Valid username and password	Redirected to admin dashboard	Successful
2.	Admin Login	Incorrect username or password	Error message appears	Successful
3.	Publication Search	Keyword “website”	Display a list of relevant publications	Successful
4.	Filter by Study Program	Select “Informatics Engineering”	Display only publications from that program	Successful
5.	Filter by Year	Select “2023”	Display only publications from that year	Successful
6.	View Publication Details	Click detail button on one entry	Display complete publication detail page	Successful
7.	Add Publication (Form)	Fill all fields and click submit	Data successfully added to the system	Successful
8.	Add Publication (Upload File)	Upload valid CSV file	All data from the file successfully added	Successful
9.	Edit Publication Data	Chage the title and click save	Changes saved and displayed to the user	Successful
10.	Delete Publication	Click delete button and confirm	Data removed from the system	Successful
11.	Admin Logout	Click logout button	Redirect back tologin page	Successful

Table II shows that all the main features of the system are working well. Login successfully verifies credentials, search and publication filters work according to keywords, and publication details can be displayed correctly. Data addition, either through forms or CSV, successfully saves new data. Edit and delete features function as expected, and logout redirects back to the login page. All test scenarios produce appropriate outputs, indicating that the system is functioning stably and according to user needs.

B.2. Non-Functional Testing

Non-functional testing aims to evaluate aspects of the system that are not directly related to functional output, but still have a significant impact on the overall quality and reliability of the system. In this study, non-functional testing was carried out with two main components, namely system performance, particularly in terms of scalability and data consistency. The evaluation of scalability was conducted to measure the extent to which the system is capable of handling large requests within a limited time, especially during mass data uploads. This testing focused on the ‘POST api/elt/upload-and-transform’ endpoint, which plays an important role in handling CSV file uploads and storing scientific publication data in the MongoDB database.

To simulate large-scale data processing, a file containing 1,000 dummy publication entries was used. This dummy data was deliberately designed to resemble the structure and level of complexity of the original dataset, enabling it to realistically represent high system load conditions. This strategy allows for more optimal evaluation of system capacity limits without being constrained by the limitations of the actual dataset, which tends to be smaller. The five configuration scenarios used in the scalability testing are presented in Table IV.

TABLE III. CONFIGURATION USED FOR SCALABILITY TESTING

Configuration	Number of Threads	Ramp-Up	Data Volume
Test 1	10	30 sec	1,000
Test 2	20	30 sec	1,000

Test 3	50	30 sec	1,000
Test 4	100	30 sec	1,000
Test 5 (Spike)	100	1 sec	1,000

TABLE IV. RESULTS OF SCALABILITY TESTING

Configuration	Avg Resp Time (ms)	Error	Throughput (request/min)
Test 1	4.792	0.00%	12.5
Test 2	100.533	0.00%	2.8
Test 3	348.547	22.58%	1.5
Test 4	889.794	61.73%	1.7
Test 5	588.022	88.97%	3.8

Based on the scalability test results shown in Table V, the system shows significant performance differences as the number of threads or simultaneous users increases. In Test 1 (10 users), the system showed the best performance with an average response time of 4.792 ms, no errors (0.00%), and a throughput of 12.5 requests per minute. This reflects that the system performs very well under low load. However, in Test 2 and Test 3 (with 20 and 50 users), there was a significant increase in response time to 100.533 ms and 348.547 ms, respectively, along with the emergence of errors up to 22.58%. This indicates that the system begins to experience bottlenecks when handling higher parallel loads.

Conditions worsened in Test 4 and Test 5 (100 users), where the error rate reached 61.73% and 88.97% respectively, with the average response time reaching nearly 15 minutes. Although throughput was relatively stable (1.5–3.8 requests per minute), this did not reflect the system's success, but rather that requests were processed or failed in a consistent manner. The combination of high response times and high failure rates indicates that the load of data storage and transformation processes on the backend (MongoDB) became a critical point for this system.

When compared to systems in previous studies, such as those conducted by Yerra (2025), the performance difference becomes even more evident. Yerra implemented an ETL pipeline using SSIS and achieved high efficiency in large-scale data integration through optimization techniques such as Fast Load, buffer settings, and batch inserts. The processing time achieved in that study was significantly faster, even when processing large volumes of data repeatedly. Meanwhile, the system in this study uses an ELT approach with Node.js and MongoDB, which, although flexible and modern, has not been optimized in terms of batch processing and concurrency control. This indicates that there is still significant room for performance improvement, particularly in batching configuration, the use of the bulkWrite() method with upsert, and system tuning to match the performance of similar systems discussed in the literature.

Thus, Test 1 configuration can be considered the most balanced as it maintains system stability and integrity under light load conditions. However, to achieve performance comparable to other systems in the literature, technical enhancements are required, particularly in parallel processing of large data and storage efficiency in MongoDB.

Next, a data consistency test was conducted to evaluate the system's ability to prevent duplicate data storage in the ‘publications’ collection. This test focused on ensuring that after the data integration process, especially during scalability testing, there was no data duplication in the collection. This test is important because repeatedly loading data with the same file has the potential to create identical data copies. In the ‘publications’ collection, the system is designed to store only one version of each data entry and automatically ignore other identical copies. This is based on a combination of key attributes, namely ‘Judul_Jurnal’, ‘Tahun_Jurnal’, and ‘Ketua’. To detect duplicates after the upload process, an aggregation query is performed on the ‘publications’ collection using the combination of these three attributes. The query used is as follows:

```
db.publications.aggregate([
      {
        $group: {
          _id: {
            Judul_Jurnal: "$Judul_Jurnal",
            Tahun_Jurnal: "$Tahun_Jurnal",
            Ketua: "$Data_Personil.Ketua"
          },
          count: {$sum: 1}
        }
      },
      {
        $match: {
          count: {$gt: 1}
        }
      }
])
```

After the query was performed, the result was that the query returned an empty array ([]) in every test. This shows that the system successfully maintained data integrity even under high load pressure. In addition, the results also reinforce that the storage and validation mechanisms implemented in the system, including unique index settings and attribute mapping, have functioned effectively to prevent data duplication in this system.

V. CONCLUSION AND SUGGESTIONS

This study successfully designed and implemented a web-based scientific publication data management system using a NoSQL architecture, with MongoDB Atlas as the database and an Extract, Load, Transform (ELT) approach. The system is capable of handling varying metadata structures from different study programs within the Faculty of Engineering at UNRAM, through both manual input and CSV file uploads. Data transformation and validation are performed automatically using the Aggregation Pipeline to normalize the structure, remove duplicates, and ensure consistent data storage.

Functional testing showed that all key features such as search, filtering, data management (CRUD), and visualization worked as intended. However, since the testing was conducted by the developer, the results are

internally valid but still require further testing by external users for more objective evaluation. From a non-functional perspective, the system performs optimally under light loads (up to 10 users), but performance degrades under heavy loads. Consistency tests also showed that the system can technically prevent identical data, but it still struggles with detecting semantic similarity, such as variations in name spelling. This highlights the need to further develop smarter entity matching features.

For future development, it is recommended to optimize backend performance using batch processing or the bulkWrite() method, and to add name-matching algorithms (e.g., fuzzy matching) to detect duplicates despite spelling differences. The system should also be enhanced with advanced authentication, integration with institutional systems, and monitoring and analytics features to support long-term sustainability and functionality expansion.

REFERENCES

[1] A. A. Aydin, "A Comparative Perspective on Technologies of Big Data Value Chain," *IEEE Access*, vol. 11, 2023, doi: 10.1109/ACCESS.2023.3323160.

[2] V. Keskar*, Dr. J. Y. Yadav, and Dr. A. H. Kumar, "5V's of Big Data Attributes and their Relevance and Importance across Domains," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 11, pp. 154–163, Sep. 2020, doi: 10.35940/ijitee.K7709.0991120.

[3] F. Reza, I. K. D. Indah, and M. Ropianto, "Perancangan Dan Implementasi Institutional Repository Dengan Metadata Dublin Core," *Jurnal KomtekInfo*, pp. 125–132, Dec. 2022, doi: 10.35134/komtekinfo.v9i4.318.

[4] S. J. Pipin, "BIG DATA (Mengenal Big Data & Implementasinya di Berbagai Bidang)," 2024. [Online]. Available: https://www.researchgate.net/publication/378313489

[5] N. Reddy Mandala, "Data Integration in Heterogeneous Systems", doi: 10.56472/25832646/JETA-V2I4P122.

[6] I. M. Putrama and P. Martinek, "Heterogeneous data integration: Challenges and opportunities," Oct. 01, 2024, *Elsevier Inc.* doi: 10.1016/j.dib.2024.110853.

[7] P. Badgujar, "Optimizing ETL Processes for Large-Scale Data Warehouses," 2020. doi: https://doi.org/10.52783/jisem.v10i8s.1130.

[8] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, "SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review," 2023. doi: 10.3390/bdcc7020097.

[9] "Comparative Case Study Difference Between Azure Cloud SQL and Mongo Atlas MongoDB NoSQL Database," *International Journal of Emerging Trends in Engineering Research*, vol. 9, no. 7, 2021, doi: 10.30534/ijeter/2021/26972021.

[10] B. Balusamy, N. Abirami. R, S. Kadry, and A. H. Gandomi, "Big Data: Concepts, Technology, and Architecture," 2021.

[11] B. Silva, J. Moreira, and R. L. de C. Costa, "Logical big data integration and near real-time data analytics," *Data Knowl Eng*, vol. 146, Jul. 2023, doi: 10.1016/j.datak.2023.102185.

[12] J. V. L. Cabral, V. E. R. Noguera, R. R. Ciferri, and D. Lucrédio, "Enabling schema-independent data retrieval queries in MongoDB," *Inf Syst*, vol. 114, 2023, doi: 10.1016/j.is.2023.102165.

[13] E. Anthony and Tony, "PERANCANGAN APLIKASI MANAJEMEN DATA PUBLIKASI DAN PENELITIAN," *JIKSI*, 2024, doi: https://doi.org/10.24912/jiksi.v12i2.31564.

[14] C. Ye, H. Duan, H. Zhang, H. Zhang, H. Wang, and G. Dai, "Multi-Source Data Repairing: A Comprehensive Survey," May 01, 2023, *MDPI*. doi: 10.3390/math11102314.

[15] R. K. Mohan, R. R. S. Kanmani, K. A. Ganesan, and N. Ramasubramanian, "Evaluating NoSQL Databases for OLAP Workloads: A Benchmarking Study of MongoDB, Redis, Kudu and ArangoDB," May 2024, doi: https://doi.org/10.48550/arXiv.2405.17731.

[16] A. A. E. Alflahi, M. A. Y. Mohammed, and A. Alsammani, "Enhancement of database access performance by improving data consistency in a non-relational database system (NoSQL)," 2023. doi: https://doi.org/10.48550/arXiv.2308.13921.