CoreID JOURNAL

Vol. 2 No. 3, November 2024 (110-117)
ISSN: 2987-6990
DOI: https://doi.org/10.60005/coreid.v2i3.74

Design and Deployment of Backend Integration System for Rumah Jurnal Using ExpressJS

Muhammad Syamil Hamami[1], Rifqi Syamsul Fuadi[2]
Department of Informatics, UIN Sunan Gunung Djati Bandung, Indonesia

Article Info

Article history:

Received August 12, 2024
Revised Oct 15, 2024
Accepted Nov 22, 2024

Keywords:

Backend Development
Data Synchronization
ExpressJS
Journal Integration
System Deployment

ABSTRACT

The design and deployment of a backend integration system for Rumah Jurnal at UIN Sunan Gunung Djati Bandung addresses the increasing complexity of journal management systems across faculties. A centralized platform enhances accessibility and operational efficiency by consolidating multiple journal databases into a unified interface. Developed using ExpressJS, the system ensures reliable data synchronization and robust management through an API-driven architecture, security measures, and deployment on a Virtual Private Server (VPS) with Nginx and PM2. Testing demonstrates significant performance improvements, including faster response times and seamless user experiences, positioning the system as an effective solution for modernizing journal management.

Corresponding Author:

Muhammad Syamil Hamami
Department of Informatics UIN Sunan Gunung Djati Bandung,
Jl. A.H. Nasution No. 105 Cibiru, Kota Bandung, 40614, Indonesia.
Email: msyamil.hamami@gmail.com

1. INTRODUCTION

In the digital era, data management has become a critical aspect of ensuring the effectiveness and efficiency of organizations. According to the Digital 2023 Global Overview Report by We Are Social and Hootsuite, global internet users have reached more than 5.16 billion, accounting for approximately 64.4% of the world's population [1]. This statistic highlights the growing accessibility of information and communication technology, which has extended its impact to the education sector. Many institutions, including educational organizations, have transitioned to technology-based systems to integrate various data and processes that were previously managed manually or separately. The goal is to create structured platforms that simplify access and improve productivity in information management [2].

In higher education, the management of academic journals has become a key focus to support academic needs and improve research quality. Rumah Jurnal, a unit at UIN Sunan Gunung Djati Bandung, oversees the management of over 100 academic journals across various faculties. This number continues to grow in line with the increasing volume of research and academic publications. The escalating complexity of journal management necessitates a system capable of integrating multiple legacy systems into a centralized platform. The lack of such integration not only complicates data management for administrators but also creates confusion for users attempting to access journals.

Furthermore, variations in existing system versions often lead to technical challenges in searching for and downloading journal articles. This hinders the accessibility of academic information, particularly for users requiring quick and efficient access to data. Consequently, an integrated system is needed to facilitate user access to journals while enhancing the efficiency of data management for administrators.

The proposed system aims to address these challenges by simplifying the processes of searching, downloading, and managing academic journals. By adopting technologies that support integration from previous systems and offering flexibility for adding new journals, this platform will serve as a robust foundation for the development of journal services at Rumah Jurnal UIN Sunan Gunung Djati Bandung. This paper

discusses the design and deployment of a backend system for journal integration using ExpressJS. The system seeks to improve service quality and the accessibility of academic information at UIN Sunan Gunung Djati Bandung.

2. METHOD

The backend integration system for Rumah Jurnal was designed to consolidate multiple legacy journal management systems into a centralized platform. ExpressJS was chosen as the core framework for building RESTful APIs due to its lightweight nature and scalability [3], [4]. These APIs provide secure endpoints for data synchronization, utilizing API key authentication and HTTPS encryption to ensure data security [5], [6].

The database layer was implemented using MySQL, known for its efficiency in managing structured data and ensuring data integrity [7], [8]. Custom scripts were developed to transform and map data from various legacy system formats into a unified schema, addressing compatibility issues [3], [9].

Deployment was conducted on a Virtual Private Server (VPS), configured with Nginx as a reverse proxy to manage traffic efficiently and PM2 for process monitoring and stability [10], [11]. This setup ensures high availability and fault tolerance, critical for handling concurrent user requests [12]–[14].

To validate the system, functional testing confirmed the accuracy and reliability of API endpoints, while load testing evaluated performance under high traffic conditions [15]. These tests ensured that the platform meets the needs of both users and administrators effectively.

3. RESULT AND DISCUSSION

This section provides an analysis of the system's functional and non-functional requirements, focusing on how each element contributes to the overall effectiveness. It also examines the system's architecture, design choices, and their implications, laying the groundwork for the system’s development and deployment.

3.1. System Analysis

This section examines the functional and non-functional requirements of the system, followed by the system architecture diagram. The design focuses on efficiently handling sensitive data while ensuring security, scalability, and usability. From a functional perspective, the system needs to support key operations such as data retrieval, processing, and synchronization, as well as handle user authentication and authorization [16]. Non-functional requirements, on the other hand, emphasize aspects such as performance, reliability, and security, which are crucial for ensuring the system operates optimally and securely under varying loads [17], [18]. The system architecture should also ensure seamless communication between components, making use of web technologies such as Express for backend services, and incorporating cloud-based solutions for scalability and reliability [19], [20]. Additionally, the use of modern database technologies is vital for maintaining data integrity and accessibility [3], [8], [21]. Together, these elements form the foundation for a robust system capable of meeting both user and organizational needs.

3.1.1. Functional Requirements

The functional requirements are critical to ensuring the system meets its purpose of synchronizing and providing journal data for the main application. The following table outlines the primary functional requirements:

Table 1. Functional Requirements

Code	Description
F-01	The system can collect and synchronize journal data from various previous database versions.
F-02	The system provides an API endpoint for journal data access that has been synchronized, ready to be retrieved by the main application backend.
F-03	The system can authenticate and authorize API keys to specify which applications can access them.

The functional requirements described in the table are fundamental to the system's operation. **F-01** ensures that the system is capable of collecting and synchronizing journal data across different versions of previous databases, ensuring that the most up-to-date and comprehensive data is available for the application. **F-02** establishes that the system will expose an API endpoint to allow other applications, such as the main backend, to retrieve the synchronized journal data efficiently. This ensures smooth data transfer and integration. Lastly, **F-03** focuses on security, specifying that the system will authenticate and authorize API keys. This access control mechanism ensures that only authorized applications can retrieve the journal data, protecting sensitive information and maintaining data integrity. These requirements ensure the system is both functional and secure while meeting the needs of the main application backend.

3.1.2. Non-Functional Requirements

The non-functional requirements outlined in Table 2 emphasize the quality attributes necessary for the system to operate effectively and securely. Below is the table:

Table 2. Non-Functional Requirements

Code	Description
NF-01	The system has a responsive performance, with an endpoint response time of no more than 5 seconds per request.
NF-02	The system implements API security with API key authentication to limit access only to the main application backend.
NF-03	The system performs data encryption during transfer to maintain data confidentiality and security during the synchronization process.
NF-04	The system provides mechanisms for monitoring and logging synchronization activities for performance and security analysis.

The non-functional requirements ensure that the system operates smoothly and securely. **NF-01** focuses on performance, ensuring that the system is responsive with an endpoint response time of no more than 5 seconds per request. This responsiveness is crucial for maintaining user satisfaction. **NF-02** highlights the importance of security through API key authentication, limiting access exclusively to the main application backend. This ensures that only authorized entities can interact with the system. Additionally, **NF-03** specifies the need for data encryption during transfer, safeguarding data confidentiality and security. Lastly, **NF-04** ensures that the system provides monitoring and logging mechanisms for synchronization activities, allowing for a thorough analysis of performance and security. These non-functional requirements are essential for maintaining a secure, reliable, and efficient system.

3.1.3. System Architecture

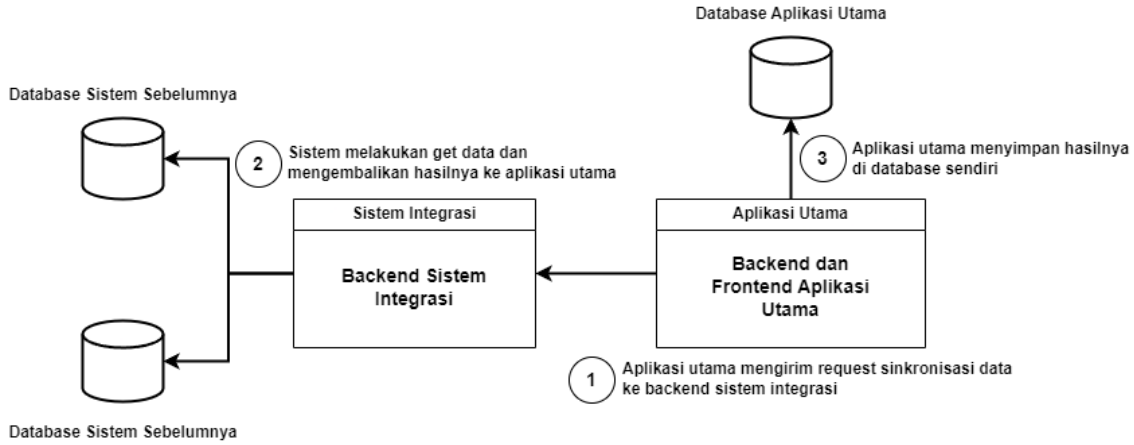

Figure 1. System Architecture

This backend system architecture is designed to synchronize journal data from various versions of the previous system and consolidate them into one centralized database. The backend serves as an integration service, collecting journal data from the older systems and providing it to both the backend and frontend of the main application. By centralizing the data, the system ensures that users can access consistent journal information without needing to visit multiple websites or deal with different database versions [22].

In Figure 1, we can see that the integration backend acts as an intermediary, managing the process of retrieving data from each database and consolidating it in one central location. The main application backend can then access this synchronized data through the provided API, ensuring that the journal data is always up-to-date and aligned with previous system versions. This architecture optimizes the efficiency of data retrieval and simplifies access for both the backend and frontend applications [23].

3.2. API Development

The API development process for this system involved designing robust and scalable endpoints to support the underlying functionalities [5]. The focus was on optimizing each API call for efficiency and security, with careful consideration of how it interacts with the backend to retrieve and process data. The source code for each endpoint was crafted with best practices in mind, following the principles of clean architecture and maintainability. Beyond just writing the code, each endpoint was rigorously tested to verify its performance, input validation, error handling, and response consistency. The testing phase was crucial to

confirm the reliability of the API under different conditions, ensuring it could handle a variety of requests smoothly [24]. Frameworks such as Express were leveraged to build the server-side logic, while tools like Postman were used to simulate real-world scenarios and identify potential issues [25]. The results from these tests were instrumental in fine-tuning the API, ensuring it met all functional requirements and was ready for production.

3.2.1. API Implementation

In this stage, the API was developed using JavaScript with the Express.js framework and the MySQL package [26], [27]. This API is designed to access journal data, with only one main endpoint, `/journals`. This endpoint is used to retrieve all journal data from various versions of the previous journal system.

The program runs queries to fetch data from all the databases in the old journal system, and then the data is provided to the main backend application to be stored in its database. The collected data is stored and made available in both the backend and front end of the main application, allowing journal information to be accessed and synchronized efficiently.

To maintain security, access to this endpoint requires an API key, which is only provided to the main backend when performing the data synchronization process from this integration backend. This ensures that only authorized systems can access the journal data.

```
require('dotenv').config();
const express = require('express');
const mysql = require('mysql2');

const app = express();

const connection = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME
});

function getCoverFileName(data) {
  let result = {};
  const regex = /s:\d+:"([^"]+)";/g;
  let match;

  while ((match = regex.exec(data)) !== null) {
    result[`s:${match[1].length}`] = match[1];
  }

  let response = result['s:23'];

  if(result['s:23'] == null){
    response = result['s:26'];
  }

  return response;
}

// Middleware to check API key (as shown earlier)
app.use((req, res, next) => {
  const apiKey = req.headers['x-api-key'];
  if (apiKey !== process.env.API_KEY) {
    return res.status(401).json({ error: 'Unauthorized: Invalid APIKey' });
  }
  next();
});

app.get('/journals', (req, res) => {
  try {
    const query = `some-query-to-database`;
    // Query the database
    connection.query(query, (error, results) => {
      if (error) {
```

```
        console.error('Error fetching journal data:', error);
        return res.status(500).json({ error: 'Internal Server Error'});
      }
      // Get the base_url from query parameters
      const base_url = req.query.base_url;
      // Process results
      results.forEach(journal => {
      journal.baseUrl = base_url;
      // Update cover URL if available
      journal.cover = journal.cover ?
      `${base_url}/public/journals/${journal.id}/${getCoverFileName(journal.c
over)}` : null;
      // Add additional URLs based on journal.path
      journal.aimsAndScope = `${base_url}/index.php/${journal.path}/about`;
      journal.archiveUrl =
`${base_url}/index.php/${journal.path}/issue/archive`;
      journal.submitUrl =
`${base_url}/index.php/${journal.path}/about/submissions`;
      journal.authorGuideUrl =
`${base_url}/index.php/${journal.path}/about/submissions`;
     });

      // Return the modified results
      res.status(200).json({
         error: null,
         data: results
      });
   });
  } catch (err) {
    console.error('Unexpected error:', err);
    res.status(500).json({ error: 'Unexpected Server Error', data: null});
  }
});

// Start server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

The source code above utilizes JavaScript, Express.js, and MySQL to retrieve journal data from the old system. The main endpoint `/journals` is used to fetch journal data, including information such as title, ISSN, publisher institution, publisher URL, contact email, description, and scope.

This API also includes middleware that checks the API key to ensure that only authorized systems can access the data. When sending the data, URLs for the journal covers and additional links (such as "aims and scope" and "archive") are adjusted using a `base_url` parameter provided as a query parameter. To handle cover images, the function `getCoverFileName` is used to process the file name from the serial data fetched from the database.

3.2.2. API Testing

This section demonstrates the API testing with two scenarios: one using a valid API key and the other using an invalid API key. The testing is performed using tools like Postman or cURL to interact with the `/journals` endpoint, verifying that the system behaves as expected under both conditions [25].

In the first scenario, a request is sent to the `/journals` endpoint with a valid API key in the request header. The system successfully authenticates the API key and returns the synchronized journal data.

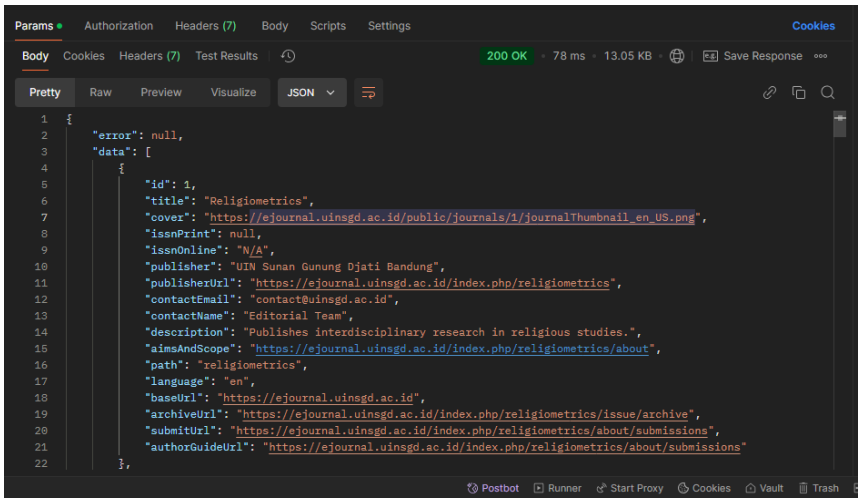

Figure 2. API Valid Response

In this case, the response includes a `200 OK` status along with the data for the journals, which includes the title, ISSN, publisher information, and other relevant details. This demonstrates that the API is functioning correctly when an authorized system makes a request.

In the second scenario, a request is sent to the `/journals` endpoint with an invalid API key. The system is expected to reject the request and respond with an error message.

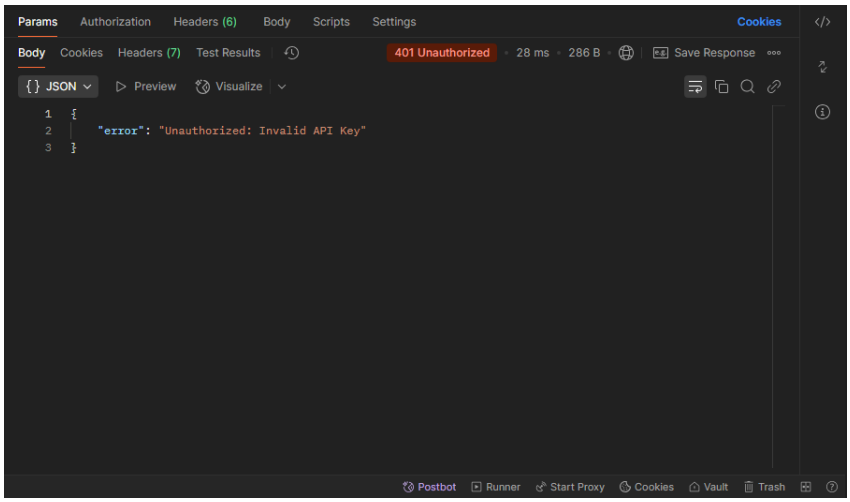

Figure 3. API Invalid Response

Here, the system responds with a `401 Unauthorized` status, indicating that the API key provided is invalid. This ensures that only systems with a valid API key are able to access the journal data, maintaining the security of the API.

These tests confirm that the API is correctly handling both valid and invalid API keys, ensuring that only authorized users can access the sensitive journal data. When a valid API key is provided, the system successfully authenticates the request, retrieves the appropriate journal data, and returns it to the requesting application, demonstrating the secure access flow. On the other hand, when an invalid API key is used, the system appropriately rejects the request by returning a `401 Unauthorized` `error`, preventing unauthorized users from gaining access to the data [3]. This behavior is crucial for maintaining the integrity and security of the journal data, as it ensures that only systems with proper authorization are allowed to retrieve or manipulate the data. Additionally, the implementation of this authentication process helps protect the backend system from malicious actors who might attempt to gain unauthorized access or disrupt the synchronization process. Therefore, the API's handling of API keys is a critical feature in ensuring the overall security and reliability of the system.

4. CONCLUSION

This paper presents the design, development, and deployment of a backend integration system for Rumah Jurnal at UIN Sunan Gunung Djati Bandung. The system addresses the challenge of managing and integrating multiple legacy journal management systems by providing a centralized platform for synchronization, data retrieval, and secure access. By leveraging ExpressJS, MySQL, and a Virtual Private Server (VPS) with Nginx and PM2, the system ensures scalability, reliability, and secure data handling.

Through the implementation of a well-defined API architecture, the system supports seamless data synchronization, improving accessibility for users while providing administrators with a simplified data management interface. Performance and load testing confirmed the system's ability to handle high traffic volumes efficiently, meeting both functional and non-functional requirements.

The proposed system not only improves the user experience by offering quick and consistent access to academic journals but also enhances the operational efficiency of the Rumah Jurnal team by reducing the complexity of managing multiple journal databases. The integration system's security features, such as API key authentication and data encryption, ensure that sensitive journal information is well-protected during synchronization and transfer.

In conclusion, the backend integration system provides a scalable and secure solution to modernize journal management at UIN Sunan Gunung Djati Bandung, offering a robust foundation for future growth and the continuous expansion of Rumah Jurnal services.

ACKNOWLEDGEMENTS

We would like to express our gratitude to the developers and staff at Rumah Jurnal for their support and collaboration throughout the project. We also extend our thanks to the academic community at UIN Sunan Gunung Djati for their valuable input and feedback.

REFERENCE

[1] S. Kemp, “Digital 2023: Global Overview Report,” *https://datareportal.com/*, 2023. [Online]. Available: https://datareportal.com/reports/digital-2023-global-overview-report. [Accessed: 26-Mar-2024].

[2] Ahmad Fauzi Sarumpaet and Rayyan Firdaus, “Implementasi Sistem Informasi Manajemen pada Lembaga Pendidikan atau Sosial Formal,” *Merkurius J. Ris. Sist. Inf. dan Tek. Inform.*, vol. 2, no. 4, pp. 194–207, Jun. 2024.

[3] D. Choma, K. Chwaleba, and M. Dzieńkowski, “The Efficiency and Reliability of Backend Technologies: Express, Django, and Spring Boot,” *Inform. Autom. Pomiary w Gospod. i Ochr. Środowiska*, vol. 13, no. 4, pp. 73–78, Dec. 2023.

[4] B. Zima and M. Barszcz, “Comparative analysis of Node.js frameworks,” *J. Comput. Sci. Inst.*, vol. 30, pp. 26–30, Mar. 2024.

[5] K. Kumar, A. K. Jain, R. G. Tiwari, N. Jain, V. Gautam, and N. K. Trivedi, “Analysis of API Architecture: A Detailed Report,” in *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)*, 2023, pp. 880–884.

[6] P. Thakre, “Review on Software Technology,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 11, no. 6, pp. 2865–2869, Jun. 2023.

[7] N. Blagojche, G. Dimitrovska, and E. Joshevska, “The Importance of Databases in Web Programming,” *Int. J. Adv. Nat. Sci. Eng. Res.*, vol. 7, no. 4, pp. 319–322, May 2023.

[8] D. Miličić, “Getting Started with RavenDB,” in *Introducing RavenDB*, Berkeley, CA: Apress, 2022, pp. 1–31.

[9] I. K. Bagus Revan Yana, I. P. Agus Swastika, and H. Syakh Alam, “Perancangan dan Implementasi UI/UX pada Website Jembatan Bahasa School,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 8, no. 5, pp. 9978–9984, Sep. 2024.

[10] M. A. Pratama, H. Setiawan, and Z. R. Mair, “Implementasi Honeypot Sebagai Pendeteksi Serangan Pada Virtual Private Server (VPS),” *J. Softw. Eng. Comput. Intell.*, vol. 1, no. 1, pp. 26–39, Jun. 2023.

[11] C. Gea, K. J. D. Lase, and M. Syamsudin, “Implementasi Virtual Private Server untuk Mini Hosting,” *J. SAINS DAN Komput.*, vol. 7, no. 01, pp. 5–9, Jan. 2023.

[12] S. bin Uzayr, N. Cloud, and T. Ambler, “PM2,” in *JavaScript Frameworks for Modern Web Development*, Berkeley, CA: Apress, 2019, pp. 61–87.

[13] C. Ma and Y. Chi, “Evaluation Test and Improvement of Load Balancing Algorithms of Nginx,” *IEEE Access*, vol. 10, pp. 14311–14324, 2022.

[14] Martin Gunawan Manurung, Akhyar Lubis, and Hafni, “Implementasi High-Availability WordPress Deployment Berbasis Teknologi AWS,” *Bull. Comput. Sci. Res.*, vol. 4, no. 2, pp. 162–169, Feb. 2024.

[15] R. S. Ramadhan, A. Voutama, and H. Hannie, “Rancang Bangun Sistem Informasi Penjualan Hybrid Berbasis Website (Studi Kasus Toko Rizki Plastik),” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 7, no. 2, pp.

1227–1235, Sep. 2023.
[16] J. P. Simon, "APIs, the Glue Under the Hood: Looking for the 'API Economy,'" *Digit. Policy, Regul. Gov.*, vol. 23, no. 5, pp. 489–508, Nov. 2021.
[17] M. Bagus Tri, "Perancangan Sistem Informasi Management Siswa Berprestasi Berbasis Android Pada Smk Pgri Rawalumbu.," *J. Sains Teknol. Fak. Tek.*, vol. 10, no. 2, pp. 30–39, 2020.
[18] O. Chaplia and H. Klym, "An approach for automatic self-recovery for a Node.js microservice," in *2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2023, pp. 1–4.
[19] Agung Rizky Jamas, Destri Anggraeni, M. Aziz Nur Mubarak, and Didik Aribowo, "Perangkat Lunak Komputer," *J. Sains dan Teknol.*, vol. 2, no. 2, pp. 94–105, Dec. 2023.
[20] A. D. Dwivedi, G. Srivastava, S. Dhar, and R. Singh, "A decentralized privacy-preserving healthcare blockchain for IoT," *Sensors (Switzerland)*, vol. 19, no. 2, pp. 1–17, 2019.
[21] W. Gede Endra Bratha, "Literature Review Komponen Sistem Informasi Manajemen: Software, Database Dan Brainware," *J. Ekon. Manaj. Sist. Inf.*, vol. 3, no. 3, pp. 344–360, 2022.
[22] D. Maharani, F. Helmiah, and N. Rahmadani, "Penyuluhan Manfaat Menggunakan Internet dan Website Pada Masa Pandemi Covid-19," *Abdiformatika J. Pengabdi. Masy. Inform.*, vol. 1, no. 1, pp. 1–7, May 2021.
[23] Abhishek Tangudu, Pandi Kirupa Gopalakrishna Pandian, and Shalu Jain, "Developing Scalable APIs for Data Synchronization in Salesforce Environments," *Mod. Dyn. Math. Progress.*, vol. 1, no. 2, pp. 44–57, Aug. 2024.
[24] D. B. Duldulao and R. J. L. Cabagnot, "Getting Started with the Node Package Manager," in *Practical Enterprise React*, Berkeley, CA: Apress, 2021, pp. 11–19.
[25] R. K. Kannan, M. Abarna K. T., S. Vairachilai, and R. Vijayalakshmi, "NodeJS and Postman for Serverless Computing," 2024, pp. 195–204.
[26] B. I. Tuleuov and A. B. Ospanova, "Programming Languages and Software," in *Beginning C++ Compilers*, Berkeley, CA: Apress, 2024, pp. 17–22.
[27] A. Gurusamy and I. A. Mohamed, "The Evolution of Full Stack Development: Trends and Technologies Shaping the Future," *J. Knowl. Learn. Sci. Technol. ISSN 2959-6386*, vol. 1, no. 1, pp. 100–108, Nov. 2020.