The Development of Polimdo Restfull API to Support Data Exchange Effectively

Veny Ponggawa
Electrical Dept of Manado State Polytechnic
Buha, Kairagi Manado

Maksy Sendiang
Electrical Dept of Manado State Polytechnic
Buha, Kairagi Manado

Jusuf L. Mappadang
Electrical Dept of Manado State Polytechnic
Buha, Kairagi Manado

ABSTRACT
The rapid growth of interconnected systems in various domains, including web services, mobile applications, and cloud-based solutions, has increased the need for efficient data exchange mechanisms. This paper explores the development of a RESTful (Representational State Transfer) API designed to support seamless data exchange between systems in POLIMDO (Manado State Polytechnic). The REST architecture, known for its stateless nature, lightweight design, and scalability, provides an efficient approach for enabling communication between heterogeneous systems. This study documents the design, implementation, and evaluation of a RESTful API tailored to exchange data in JSON format, focusing on performance, scalability, and security considerations.

General Terms
Web Development, backend platform

Keywords
RESTful API, data exchange, web services, JSON, scalability, security, POLIMDO

1. INTRODUCTION
The exchange of data between various systems has become a critical requirement in the modern digital ecosystem. From cloud services to mobile applications, APIs (Application Programming Interfaces) serve as the backbone of communication, enabling efficient interactions between software systems. Among the available API design architectures, RESTful APIs have gained significant traction due to their simplicity, scalability, and compatibility with a wide range of platforms.[5]

This paper focuses on the development of a RESTful API that supports data exchange between multiple systems founded in Manado State Polytechnic. At the moment, there are some system or aplications running in Manado State Polytechnic and all of them run with their own databases and business logic. There is no data interchange between them. A particular system is isolated with others.

The aim of this paper is to examine the effectiveness of the REST architectural style in providing an efficient and scalable mechanism for data communication. The study also considers security challenges and solutions, as well as the performance metrics involved in the exchange of JSON-encoded data.

1.1 Objectives
- To develop a RESTful API for data exchange between heterogeneous systems in POLIMDO (Manado State Polytechnic)
- To evaluate the performance, scalability, and security of the API.
- To provide recommendations for improving API design based on empirical findings.

2. LITERATURE REVIEW

2.1 REST Architecture and Principles
Roy Fielding's REST architectural style, introduced in 2000 [1], has become a widely adopted standard for web service design. REST emphasizes statelessness, client-server communication, cacheability, and uniform interfaces. Several studies have demonstrated the benefits of REST, particularly its simplicity, ease of implementation, and ability to handle large volumes of data exchange.

REST Architecture and Principles focus on designing network-based applications in a lightweight, scalable, and maintainable way. REST (Representational State Transfer) is a popular architectural style for building APIs, especially web services, due to its simplicity and scalability. Key REST Principles:[2]

- Client-Server Architecture; REST separates the concerns of the client and the server. The client is responsible for managing the user interface and user experience, while the server handles the data and business logic. This separation allows each side to evolve independently
- REST APIs are stateless, meaning each client request must contain all the necessary information for the server to fulfill the request. The server does not store client context between requests. This improves scalability by removing the need for session management on the server.
- Responses from RESTful APIs should be explicitly labeled as cacheable or non-cacheable. If a response is cacheable, the client can reuse that response for future requests, reducing the number of calls to the server and improving efficiency.
- REST allows for a layered system architecture, meaning that components (such as proxies, gateways, and load balancers) can be added between the client and the server without affecting communication. This enhances scalability and security.
- REST allows servers to temporarily extend client functionality by transferring executable code, such as JavaScript. However, this is an optional constraint, and most RESTful APIs do not use this feature.

2.2 Data Exchange Formats: JSON vs XML
The two most common formats for data exchange over RESTful APIs are JSON (JavaScript Object Notation) and

XML (Extensible Markup Language). JSON has emerged as the preferred format due to its lightweight structure and ease of use in modern web applications[6]. Prior studies highlight the efficiency of JSON over XML in terms of data transfer speed, payload size, and parsing efficiency.

JSON is a lightweight, human-readable format for representing structured data. It is derived from JavaScript but is language-independent and can be used by many programming languages. **XML** is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It was designed to transport and store data, with an emphasis on being both self-descriptive and extensible.

2.3 Security Challenges in RESTful APIs

While RESTful APIs are widely adopted due to their simplicity, scalability, and performance, they also introduce several security challenges. Since REST APIs are often exposed to the internet, they become potential attack vectors for malicious users. Here are some of the key security challenges associated with RESTful APIs:

- Lack of Built-in Security Mechanisms; RESTful APIs rely on external mechanisms such as HTTPS, OAuth, and JWT (JSON Web Tokens) to provide security features. This can lead to inconsistent or inadequate security practices if not properly implemented
- Authentication (verifying user identity) and authorization (determining access rights) are critical concerns in RESTful APIs, especially in public-facing APIs. Without robust mechanisms, APIs may become vulnerable to unauthorized access and data exposure.
- Insufficient Encryption (HTTPS); using HTTPS (SSL/TLS) is critical for protecting data exchanged between clients and servers. Without encryption, sensitive information (such as passwords, personal data, or API keys) can be intercepted during transmission, leading to man-in-the-middle (MITM) attacks. Many REST APIs still use HTTP without encryption, which exposes them to these risks.

3. METHODOLOGY

3.1 System Design

The API is designed following REST principles and built using Node.js and Express, which offer flexibility and performance advantages. The database used for persistent storage is MongoDB, chosen for its scalability and ease of integration with JSON data formats. The API endpoints are developed to perform CRUD (Create, Read, Update, Delete) operations on the dataset.[4]

- **API Endpoints:**
  - GET /api/resources – Retrieve all data.
  - POST /api/resources – Create a new data entry.
  - PUT /api/resources/{id} – Update a specific data entry.
  - DELETE /api/resources/{id} – Delete a data entry.

3.2 Data Exchange Format

In the context of a RESTful API, a Data Exchange Format refers to the structure and encoding of the data that is transmitted between the client and the server. It defines how data is formatted for both requests and responses, enabling effective communication. JSON was selected as the primary format due to its compact structure, ease of parsing, and broad compatibility with most modern programming languages [6]. Other formats like XML, CSV, and YAML can be used depending on the specific needs of the application

3.3 Security Implementation

The API employs **OAuth 2.0** for user authentication and access control. All communications are secured using **HTTPS** to encrypt the data. Additionally, **JWT** tokens are used to validate user sessions, ensuring secure and stateless interactions between the client and server..

3.4 Performance Testing

The performance of the API is tested under various conditions:

- Load Testing: Simulates multiple concurrent users to evaluate response times and server load.
- Latency and Throughput: Measures the average response time for data requests and the rate at which data is exchanged.
- Scalability Testing: Assesses the API's ability to scale horizontally by adding additional instances under high load.

4. RESULTS AND DISCUSSION

4.1 System Design

The aim of the POLIMDO Restful API system was created to provide efficient and structured access. Used to retrieve some required data such as lecturer data, course data, research data, service data and tridharma data at the Manado State Polytechnic. The API enables effective data management, including CRUD (Create, Read, Update, Delete) operations and supports the needs of users such as lecturers, students and admins.

The scope of this system includes endpoints on the API which are used to manage lecturer data, research, community service and has functional and non-functional requirements as follows

- To manage lecturer data; the main function is to use endpoints to display, add, delete and update lecturer data
- To manage course; the main function is to use endpoints to display, add, delete and update course data
- To manage research and community service data
- To manage data public access

4.2 System Modelling

System modeling in a RESTful API context involves designing and representing the various components and interactions of the system in a way that adheres to REST principles. This includes defining resources, their relationships, and how clients interact with these resources through HTTP methods. The modelling is described as follows

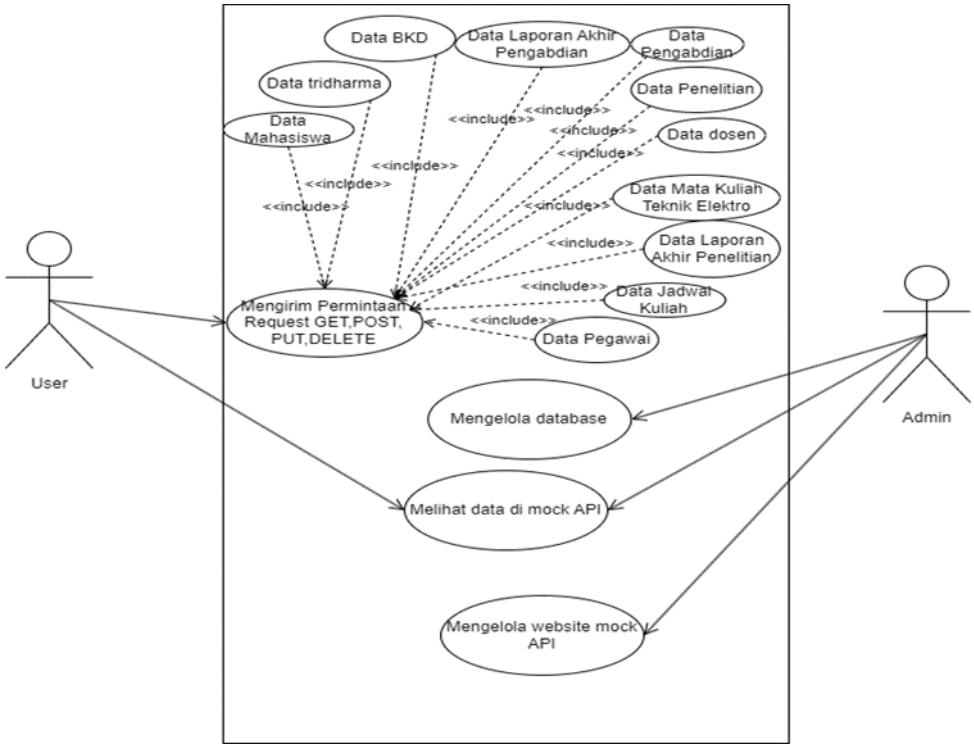

Fig 1. Use case diagram

The endpoints, is shown in the next table

Table 1. REST API Endpoints

	Routes	
Resource	**HTTP Methods**	**Endpoints**
/api/apiResearch	GET	/api/apiResearch
		/api/apiResearch/:id
	POST	/api/apiResearch
	PUT	/api/apiResearch/:id
	DELETE	/api/apiResearch/:id
/api/apiService	GET	/api/apiService
		/api/apiService/:id
/api/apiLecturer	GET	/api/apiLecturer
/api/apicivilServant	GET	/api/apicivilServant
/api/apiStudents	GET	/api/apiStudents
/api/apiSubjects	GET	//api/apiSubjects

The complete implementation of POLIMDO REST API is shown into the picture below (can be accessed directly via https://polimda-api.my.id/)

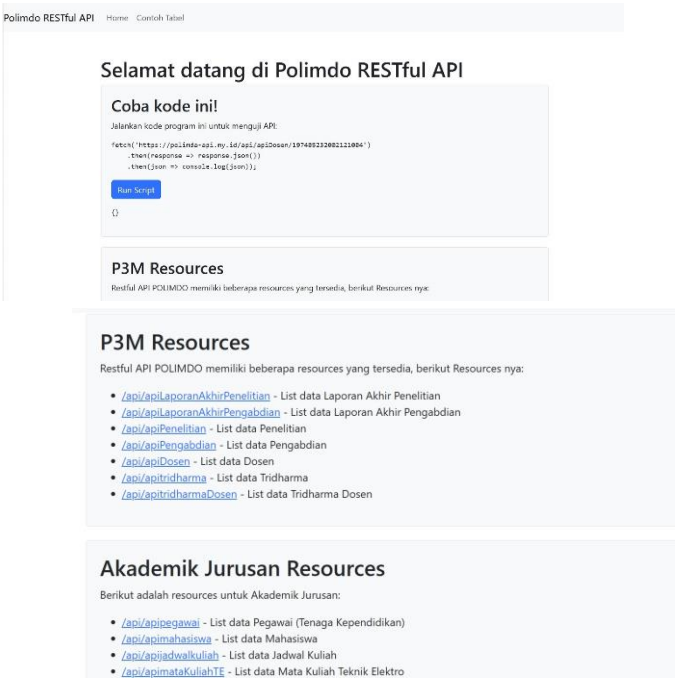

Figure 2.POLIMDO REST API

5. CONCLUSION

This paper demonstrates the successful development and implementation of a RESTful API to support efficient data exchange. Through performance testing and security evaluation, it was found that the API meets the necessary requirements for scalability, speed, and security. While REST remains a robust choice for modern web services, future

iterations could explore hybrid approaches, such as combining REST with GraphQL for enhanced flexibility.

6. REFERENCES

[1] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.. .

[2] Richardson, L., & Ruby, S. (2007). *RESTful Web Services*. O'Reilly Media..

[3] Koren, M. (2018). Performance Comparison: JSON vs. XML. *Journal of Web Services*, 23(1), 45-58Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[4] Hardt, D. (2012). The OAuth 2.0 Authorization Framework. *RFC 6749*..

[5] Narayan, V. (2020). API Security: Best Practices for Modern Web Applications. *Cybersecurity Journal*, 12(3), 70-82..

[6] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE.

[7] Y.T. Yu, M.F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions", Journal of Systems and Software, 2005, in press.

[8] Spector, A. Z. 1989. Achieving application requirements. In Distributed Systems, S. Mullender