e-ISSN: 3165-1453 https://doi.org/10.36782/gitcoin.v1i1.665

Journal of Digital Transformation, Computing, and Intelligence (GITCOIN)
Vol. 1, No. 1, March 2026

Implementation of Clean Architecture in the Back-End of an Alumni Program Application: A Case Study of a Non-Profit Organization

Burhanudin Malik[1)*], Guson Prasamuarso Kuntarto[1)], Irwan Prasetya Gunawan[1)]

[1)] Department of Informatics, Faculty of Enginering and Computer Science, Universitas Bakrie, Jakarta, Indonesia
*burhanudinmalik55@gmail.com, guson.kuntarto@barkie.ac.id, irwan.gunawan@bakrie.ac.id

ABSTRACT

Alumni data tracking within the non-profit organization located in South Jakarta previously relied on Google Forms, which presented significant limitations in dynamic data management and carried a high risk of data redundancy. Furthermore, an unstructured back-end codebase in the legacy internal system hindered long-term maintenance and system evolution. This study aims to build a highly structured back-end system using the Clean Architecture pattern for an alumni program application to facilitate efficient, scalable, and testable data management. The system was developed using Node.js and the Hapi.js framework to deliver a robust RESTful API. The development process was guided by the Scrum methodology across eight sprints, where Sprints 1–4 focused on database design using the Database Life Cycle (DBLC) method through normalization up to the third normal form (3NF), and Sprints 5–8 focused on back-end component development. Clean Architecture separates system responsibilities into four primary horizontal layers: Domain, Applications, Interfaces, and Infrastructures. The results indicate that the designed MySQL relational database successfully addresses university and domicile data redundancy through a centralized schema. Functional validation via unit, integration, and system testing demonstrated that all functional requirement scenarios were fully satisfied. Architectural validation using an Architecture Fitness Function based on stability and abstractness metrics proved full compliance with The Dependency Rule, where the Domain layer was empirically shown to be the most stable (I=0.04, A=0.62) and the Infrastructures layer was concrete (I=0.92, A=0.00). Consequently, the implementation of Clean Architecture successfully establishes a robust, modular, and future-ready back-end foundation.

Keywords: *Clean Architecture, Back-end, Alumni Program Application, Node.js, Relational Database.*

Article history: Received 11 June 2026, revised 24 June 2026, accepted 10 July 2026

1. INTRODUCTION

The non-profit organization located in South Jakarta is a prominent philanthropic institution in Indonesia, established in 2010 with a strategic vision to identify and develop leadership potential across all strata of Indonesian society [1]. In executing its mission, the organization concentrates on three primary pillars of social development: education, health, and environmental management. To achieve these objectives, the organization has launched and managed various empowerment programs targeting university students and community institutions to enhance both the technical proficiency (hard skills) and interpersonal capabilities (soft skills) of the participants [1]. One of the strategic initiatives developed is the alumni program, which is specifically designed as a formal platform for program beneficiaries or alumni to remain connected, collaborate, and synergize, both among themselves and directly with the central organization [2]. Through this alumni network, graduates are expected to cooperate and establish strategic partnerships to accelerate the achievement of Sustainable Development Goals (SDGs), which form the core agenda of this non-profit organization [2].

In its operational structure, the alumni program encompasses various crucial functional domains, including an alumni center, a research center, a business center, an advocacy center, and a community development center [2]. This diversity of fields is established to address the multi-dimensional needs of the alumni while ensuring that their contributions are structured and deliver a tangible impact on the broader community. However, in practice, the operational business processes governing alumni management encounter fundamental administrative challenges. Based on in-depth interviews conducted with the program manager in March 2025, it was revealed that administrative tasks such as tracking alumni profiles, registering for community activities, and applying for recommendation letters are still executed manually using Google Forms [3]. The inherent limitations of this platform present a major obstacle, as alumni are only permitted to submit their profile data once to prevent severe data redundancy issues within the storage repository [3].

© 2026 Journal of Digital Transformation, Computing, and Intelligence (GIT-COIN)
This is an open access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC BY-NC 4.0).

This single-entry constraint triggers a new operational issue where alumni cannot update their profile data dynamically as their professional careers advance or their residential locations change. If multiple submissions are allowed on Google Forms, legacy historical records risk being unproductively overwritten by new data in an unorganized manner, or otherwise creating duplicate records that complicate data analysis. This operational bottleneck is reinforced by an internal business process audit conducted by the organization's information technology (IT) staff in late March 2025, which confirmed that the absence of a centralized relational storage system severely degrades organizational data governance [3]. In fact, for non-profit organizations, the regular management of alumni data is crucial, as alumni serve as a benchmark for program success, a medium for long-term evaluation, and a strategic asset in building the institution's image and leverage within the community[4].

Web applications are software systems executed via web browsers, frequently engineered on a full-stack framework that segregates responsibilities into a front-end interface, a back-end server, and a storage database. User-driven commands are captured by the front-end and dispatched to the server-side infrastructure for execution. The back-end subsequently serves as the core operational layer, managing data access through the database while simultaneously handling cybersecurity and request validation [5]. The back-end serves as the underlying server and database layer that drives application logic invisibly to the user. This development predominantly entails building robust APIs and configuring core HTTP operations such as POST, GET, PUT, and DELETE [6]. In the context of developing the alumni program application for the non-profit organization in South Jakarta, the back-end component plays a vital role in ensuring that all operational logic and data transactions execute seamlessly. However, significant technical challenges arise regarding internal code management. According to IT staff during interviews in mid-March 2025, the development team frequently encountered an unstructured or disorganized codebase, commonly referred to as spaghetti code [7]. This structural defect renders the software highly difficult to read, understand, test independently, and extend. When developer turnover occurs or new features are introduced, the onboarding and adaptation process consumes excessive time and frequently introduces regressions into previously stable functionalities [7]. This underscores that maintaining a clean, modular, and well-organized code structure is a critical factor for the long-term success of the information systems being deployed.

To resolve these architectural challenges, a premier solution in software engineering is the application of the Clean Architecture pattern [8]. Clean Architecture, introduced by Robert C. Martin, focuses on the separation of concerns by partitioning the codebase into distinct horizontal layers [8]. Implementing this architecture yields a highly competitive technical advantage, delivering code that is modular, independent of external frameworks, independent of specific database systems, and highly amenable to isolated unit testing. Prior research evaluating Clean Architecture implementations within Node.js runtime environments demonstrates that this approach produces technically efficient, high-performance systems that are highly adaptive to evolving future business requirements [8]. In addition to architectural design, robust database design is critical for achieving high data integrity. Consequently, executing the database design process through the DBLC Methodology is indispensable. The Database Life Cycle (DBLC) is a systematic method utilized in database development. Implementing this methodology involves several distinct phases: data requirements analysis, conceptual data modeling, logical data modeling, physical data modeling, and schema and database release [9]. Driven by these requirements, this study aims to design and develop the back-end of an alumni program application for the non-profit organization in South Jakarta, adhering to Clean Architecture principles, leveraging the Node.js platform, the Hapi.js framework, and a MySQL RDBMS to achieve a scalable, maintainable solution with high data integrity.

Research focusing on the development of alumni management systems and graduate tracer studies frequently leverages diverse technologies and architectural patterns. In 2021, Alfarizi developed a web-based alumni association information system for the Al Binaa Islamic Boarding School by implementing a microservices architecture. This implementative research utilized the Waterfall methodology to ensure that each development stage progressed systematically, preventing any operational overlaps. In terms of its technical stack, the platform was built using the Laravel framework for the backend, JavaScript and CSS for the frontend interface, and MySQL for the database management system. The transition to a microservices architecture aimed to minimize the impact of system failures; consequently, the resulting system was divided into two independent services, each maintaining its own MySQL database, which were then integrated using an API Gateway as a single gateway to route user requests efficiently[10]. In 2023, Hidayatullah et al. developed a back-end system for an alumni web application named Fateka using the Extreme Programming (XP) methodology [11]. The system was designed as an interactive portal for distributing career vacancies and community events, using the Laravel framework based on the conventional Model-View-Controller (MVC) architecture. Performance testing conducted via ApacheBench under a simultaneous load of 1,000 concurrent requests demonstrated system stability, with average response times remaining under 5 seconds [11]. Although the MVC pattern in that study proved stable under load, conventional MVC frameworks often couple business logic tightly to the framework itself, complicating future technology migrations and the isolated testing of core enterprise rules.

Subsequently, Manurung and Matondang designed the Halo Alumni information system to facilitate communication among graduates and streamline administrative tracking by faculty administrators [12]. This

system was constructed using the Waterfall methodology and deployed on the MERN technology stack (MongoDB, Express.js, React.js, and Node.js). From a security perspective, the system enforced strict authentication across all exposed web endpoints using JSON Web Tokens (JWT) [12]. While the use of a non-relational database (MongoDB) provided schema flexibility, relational database approaches are generally considered superior for managing alumni records that exhibit strict transactional and referential relationships to ensure data consistency.

Furthermore, Wahyuni et al. in 2023 developed a web-based application for alumni registration and online diploma legalization using PHP 5 and a MySQL database to streamline administrative services [13]. A tracer study serves as a critical mechanism to track graduates and gather comprehensive data regarding both continuous curriculum alignment in the workplace and alumni contributions to their institutions. To optimize this process, this project developed a web-based tracking application designed to streamline methodical and effective data collection and analysis workflows. Engineered using the PHP programming language and a MySQL database under the traditional Waterfall development life cycle, the platform integrates vital features such as self-service data entry forms, administrative dashboards, statistical reporting tools, and career vacancy portals. Ultimately, the implementation proves that this real-time system enhances administrative data governance, facilitates career tracking for alumni, and provides empirical foundations to support institutional curriculum evaluation and strategic decision-making [14]. In 2025, Rachmat et al., deployed a web-based tracer study information system combining PHP and MySQL that simplified data retrieval and structured long-term storage [15]. Furthermore, in 2025, Purnama and Haryono developed a digital community platform for Al-Azhar University alumni using an integrated full-stack approach featuring the Next.js framework and Prisma ORM for PostgreSQL database management [16]. This study successfully centralized alumni data and facilitated community connectivity through the 'Campus Space' discussion forum feature, which served as an official alternative to unstructured general social media channels [16]. In 2025 Niswati et al.designed a web-based tracer study information system at SMK Negeri 1 Karang Baru utilizing the prototyping method and a RESTful API web service architecture. Their research focused on providing a technological solution to address inefficient manual alumni data collection through the effective integration of the front-end and back-end. The implementation results demonstrated that the utilization of a RESTful API successfully supports cross-platform system development (web, mobile, and desktop) and generates more accurate and comprehensive graduation records. These data can subsequently be leveraged by educational institutions to optimize educational quality, establish a foundation for strategic decision-making, and satisfy accreditation requirements [17].Lastly, in an effort to align curriculum relevance with dynamic industry demands, Al Ghyfari and Setiawan in 2025 engineered a digital tracer study platform for LP3I Bandung Polytechnic to evaluate both vertical and horizontal alumni career alignment. This implementation leveraged the Agile framework with a Scrum approach, effectively transforming conventional, inefficient online-form data collection methods into a centralized, web-based system. Utilizing a robust technology stack comprising React.js, Express.js, and a MongoDB database, the system demonstrated high flexibility in accommodating diverse alumni survey data formats. The empirical findings indicate that this approach significantly enhances real-time data processing efficiency and introduces a strategic recommendation feature, thereby facilitating data-driven academic decision-making for institutional management [18]. In contrast to traditional backend monoliths which often foster uncontrolled code duplication and low maintainability index scores, a study by Nugroho et al. in 2022 establishes that a layered clean architecture framework effectively insulates core business workflows from architectural degradation. To scientifically challenge the sustainability of monolithic designs, their research measured codebase transitions using Cyclomatic Complexity, Weighted Method Count, Kan's Defects, Halstead's metrics, and the Maintainability Index. The empirical findings demonstrated post-refactoring metric enhancements spanning 21% to 61%, proving that replacing monolithic architectures with clean, decoupled layers reduces overall structural complexity, maximizes code quality, and curtails developer maintenance overhead [19]. This study introduces a new approach from the existing back-end architecture by applying Clean Architecture to the back-end of an alumni program application, strictly decoupling core business logic from infrastructural mechanisms such as web frameworks and database systems.

2. RESEARCH METHOD

This study adopts an adaptive and iterative software engineering methodology leveraging the Scrum framework. The selection of Scrum is based on its capacity to provide clear, transparent development phases and facilitate incremental evaluations of code quality across successive iterations. The initial phase of this research focused on data collection and requirements analysis through direct observation and in-depth interviews. Observations were conducted from February 17 to 28 with the objective of mapping the existing alumni data collection mechanisms and identifying channels used to disseminate career and scholarship opportunities. The observational findings revealed structural efficiency constraints caused by the absence of an integrated supporting platform. Subsequently, interviews were conducted to investigate operational bottlenecks, validate the observational data, and formulate specific system requirements. Structured interviews were held with three

primary stakeholders: the Alumni Program Manager (interviewed on March 7, 2025) and the Organizational IT Staff (interviewed on March 14, 2025, and March 31, 2025).

The interview indicated that the alumni data management system and recommendation letter application workflow still rely on Google Forms, which limits data updates to a single entry to mitigate data redundancy risks. Furthermore, the organization lacks a centralized web application to interactively distribute career and scholarship information. From a technical standpoint, historical constraints involving an unstructured legacy software architecture underscored the necessity for technological standardization and an API architecture aligned with the organization's existing IT infrastructure. Collectively, these findings established the core requirements for the new system: a self-service platform to prevent data redundancy, a centralized information portal, and the implementation of a clean architecture to guarantee scalability.

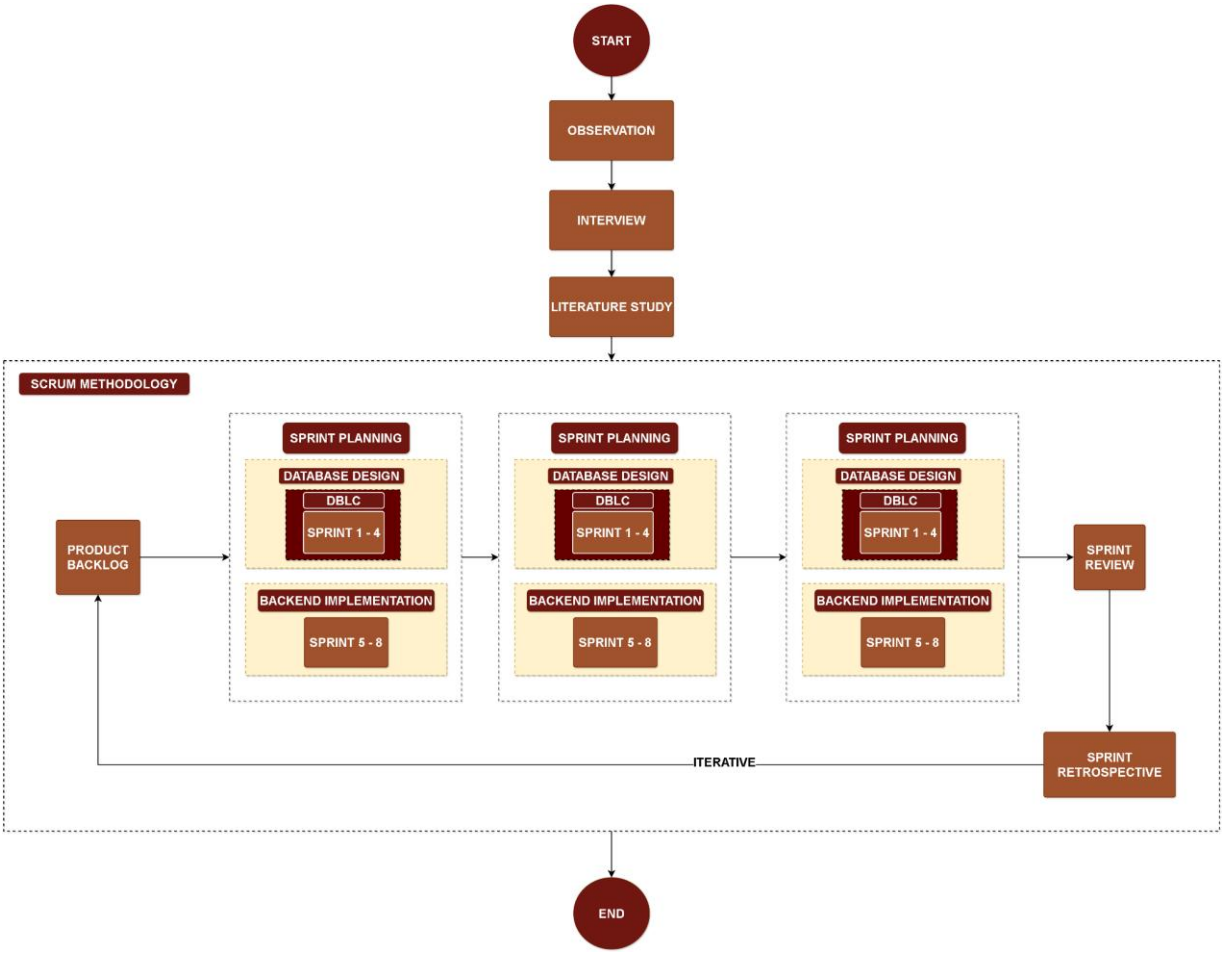


Figure 1 Research Methodology for Implementing Clean Architecture in the Backend of an Alumni Program Application

2.1. The Development Process

The development process in this study was comprehensively mapped out across a total of eight logical sprints. The operational focus was balanced to guarantee optimal system output: the first four iterations (Sprint 1 to Sprint 4) were dedicated entirely to database design, analysis, and relational data normalization adhering to the Database Life Cycle (DBLC) methodology. Conversely, the remaining four iterations (Sprint 5 to Sprint 8) were allocated specifically for the back-end code implementation of the alumni program application, alongside comprehensive functional and architectural validation testing.

2.2. Scrum

Each sprint within the Scrum methodology was executed under strict adherence to four standard ceremonies: 1) Sprint Planning to define the target back-end features for the cycle; 2) Daily Scrum to coordinate progress and identify technical impediments; 3) Sprint Review to demonstrate completed increments of back-end functionality; and 4) Sprint Retrospective to evaluate work processes and refine code structure to maintain compliance with Clean Architecture principles[20]. The technical stack leveraged in this research includes the

Node.js v20.9.0 runtime environment, the Hapi.js v20.1.5 HTTP server framework, a MySQL RDBMS for data persistence, the Railway cloud platform for continuous deployment, Postman for RESTful API functional validation, and automated testing scripts to measure the Architecture Fitness Function based on system stability and abstractness metrics.

3. RESULTS AND DISCUSSION

This particular research focuses on designing the database system and implementing the clean architecture for back-end.

3.1. Database Design and Normalization Results

The initial phase of the study involved conducting a rigorous audit and analysis of the raw alumni data structures previously collected by the non-profit organization in South Jakarta via Google Forms. This raw data was mapped into an unnormalized form (UNF) table, which suffered from structural defects, such as repeating and inconsistent text entries for university names, majors, graduation years, and residential history, which were manually re-entered for each new alumni submission. This structural disorder triggered data anomalies and caused significant storage inefficiencies. To resolve these issues, incremental normalization was performed. During the First Normal Form (1NF) stage, all attribute values were rendered atomic, and repeating groups were eliminated. Subsequently, transformation to the Second Normal Form (2NF) was achieved by isolating attributes with partial dependencies into independent master tables. This resulted in several referential master tables: the domicile master table (`master_domiciles`), the graduation batch master table (`master_batch`), the university master table (`master_universities`), and the university major master table (`master_universities_major`).

The subsequent critical phase involved executing Third Normal Form (3NF) normalization to eliminate all remaining transitive dependencies among non-key attributes. The final 3NF schema successfully established a centralized relational database structure comprising the primary user transaction table (`tx_users`), the specific alumni tracking table (`tx_users_alumni`), and the educational mapping bridge table (`tx_users_universities`). As a concrete example of normalization success, prior to schema refinement, university names and majors were stored as raw strings across thousands of individual alumni rows, allowing typos and text duplication to compromise data integrity. Following 3NF implementation, university and major details are stored exactly once within the centralized `master_universities` table. Each individual alumni profile originating from the same institution merely references the unique foreign key ID of that master entry. Consequently, data redundancy was completely eliminated, and database referential integrity was fully enforced via RDBMS MySQL constraints.

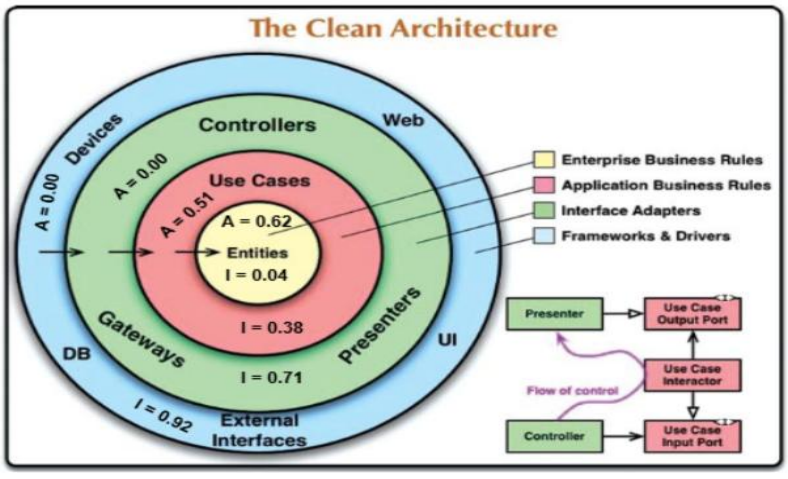


Figure 2. The overall score of Implementation of Clean Architecture in the Backend of an Alumni Program Application.

3.2. Clean Architecture Back-end Code Implementation

The back-end components of the alumni program application were concretely developed using Node.js and Hapi.js, strictly partitioned across four structural directory layers. This organization isolates core business logic from the volatility of external tooling changes. The detailed implementation within each layer is outlined below:

1. Domain Layer: This layer contains core business entities (such as Alumni, Articles, Activities, and Recommendation Letters) and defines abstract repository interfaces. Code within this layer is written in pure JavaScript without any external library imports, guaranteeing maximum stability against technology shifts.

2. Applications Layer: Implements the system's application-specific use cases, such as handling new alumni registration, creating career opportunity articles, and managing digital recommendation letter workflows. This layer interacts with data persistence layers indirectly by invoking the abstract interfaces defined in the Domain layer.
3. Interfaces Layer (Interface Adapters): Packages controller handlers and RESTful API route configurations. Its primary role is to bridge client HTTP requests, validate input payload formats, extract request parameters, execute the appropriate use case within the Applications layer, and serialize the results into standard JSON response formats.
4. Infrastructures Layer: The outermost layer handling technical operational details. It includes Hapi.js HTTP server initialization, MySQL database connection configurations, concrete repository implementations executing raw SQL queries or Sequelize operations against the database, and core security modules.
5. Commons/Exceptions: This layer stores shared components required across multiple modules, such as exception handlers. It manages error handling to facilitate debugging and ensure consistency in response structures when errors occur.

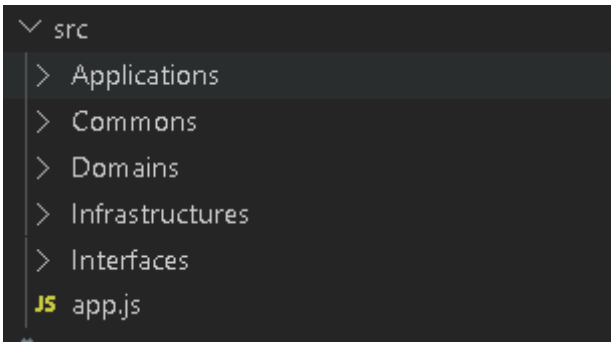


Figure 3. The Folder Structure of Clean Architecture in the Backend of an Alumni Program Application

3.3. Validation of Functional System

To ensure that all back-end components function correctly and are free from logical regressions, multi-tiered functional validation was executed, encompassing unit testing, integration testing, and system testing. Unit testing focused on evaluating code logic within the Domain and Applications layers in an isolated manner. These tests leveraged mocking techniques to simulate database repository behaviors, allowing test suites to execute rapidly without requiring an active database connection. For example, unit tests targeting the article entities successfully evaluated 9 distinct testing scenarios across 3 test suite files, achieving a 100% Passed status and validating core enterprise business rules.

Subsequently, integration testing validated end-to-end data flows across architectural layers during API request processing. Testing was concentrated on critical paths, such as the alumni recommendation letter submission feature. The results demonstrated that an HTTP POST request to the `/alumni/recommendation` endpoint successfully processed input data, yielding a '201 Created' status response and a success payload, while a GET request to the same endpoint successfully fetched the collection of submitted documents with a '200 OK' status. Finally, system testing confirmed the overall harmony of all system modules, from administrator authentication routines to transactional data processing, resulting in zero functional errors.

The architectural significance of this research was quantified using an Architecture Fitness Function approach. This validation empirically verifies that the back-end implementation strictly complies with Clean Architecture's Dependency Rule. Measurement was based on the component stability metrics defined by Robert C. Martin, specifically Instability (I) and Abstractness (A). Instability is calculated as the ratio of efferent coupling (C_e, outgoing dependencies) to total coupling ($I = C_e / (C_a + C_e)$), where a value approaching 0 indicates a highly stable component resistant to change, and a value approaching 1 indicates high instability. Abstractness ($A = N_a / N$) measures the proportion of abstract components (interfaces) within an architectural layer.

Automated architectural metric evaluations across the back-end layers of the alumni program application yielded the following results:

1. The Infrastructures layer recorded an Instability score of $I = 0.92$ and Abstractness of $A = 0.00$. This confirms that the outermost layer is highly unstable (flexible to change) and completely concrete, as it manages external technical details like the Hapi.js framework and MySQL drivers.
2. The Interfaces layer recorded an Instability score of $I = 0.71$ and Abstractness of $A = 0.00$, reflecting its nature as a concrete layer bridging HTTP RESTful communication protocols.

3. The Domain layer, serving as the innermost core of the architecture, recorded an exceptionally low Instability score of I = 0.04 and a high Abstractness rating of A = 0.62. This provides empirical evidence that the non-profit organization's core business logic is successfully insulated within a highly stable, independent environment dominated by abstract interfaces, ensuring it remains unaffected by changes in outer infrastructural technologies.
4. The Applications layer achieved an ideal architectural balance, with an Instability score of I = 0.38 and Abstractness of A = 0.51. The balanced abstraction of this use case layer provides high maintainability and flexibility for developers to extend system features without compromising the existing architecture. Overall, these findings scientifically prove complete adherence to Clean Architecture design paradigms.

4. CONCLUSION

Based on the design, implementation, and multi-tiered evaluation executed in this study, several critical conclusions can be drawn. First, the data redundancy and text inconsistency issues stemming from legacy Google Forms usage within the non-profit organization in South Jakarta have been fully resolved through database engineering guided by the Database Life Cycle (DBLC). Normalization up to the Third Normal Form (3NF) successfully converted disorganized raw records into an efficient, centralized, and consistent relational schema on a MySQL RDBMS, eradicating master data duplication across university names, majors, and domiciles through the strict enforcement of foreign key constraints.

Second, engineering the back-end components using Node.js and Hapi.js guided by Clean Architecture principles successfully produced a highly structured, clean, and modular codebase. Adherence to The Dependency Rule was scientifically validated via an Architecture Fitness Function, which demonstrated that the core enterprise logic layer (Domain) maintains the highest stability profile (I=0.04, A=0.62) and remains perfectly insulated from outer infrastructure modifications (I=0.92, A=0.00). The primary benefit of this Clean Architecture implementation is the realization of an alumni program application back-end that is highly maintainable and scalable, allowing the organization's IT staff to extend functional features or execute future technology migrations without endangering core system stability.

This section should briefly explain the significance of the findings and their implications for research or practice in the relevant field. Authors may also discuss the limitations of the study and provide recommendations or directions for future research. No new data, results, or citations should be introduced in this section.

To ensure sustainable system evolution and anticipate future operational demands, the following strategic recommendations are proposed: 1) The development team should compile comprehensive technical documentation and establish standardized scaffolding or code templates to streamline and accelerate the onboarding of new developers and the creation of features within this multi-layered architecture; 2) Given the projected growth of the alumni user base, proactive performance optimization strategies should be implemented, specifically by integrating an in-memory caching layer using Redis on high-frequency API endpoints to mitigate server overhead and database query latency.

DATA AVAILABILITY STATEMENT

The authors express their sincere gratitude to the non-profit organization located in South Jakarta for granting data access and providing the necessary operational insights required to complete this case study. Special appreciation is also extended to the institutional reviewers and technical staff whose scholarly guidance significantly enhanced the system architecture and database design phases of this research.

ACKNOWLEDGMENTS AND AI USE DISCLOSURE

During the preparation of this manuscript, the authors utilized an artificial intelligence (AI) language platform to translate the original draft from Bahasa Indonesia into English. This technology was employed strictly to optimize cross-lingual drafting efficiency and grammatical alignment with international publishing standards. Following the automated translation, the authors conducted a rigorous manual review, comprehensive editing, and technical verification of the entire text to guarantee conceptual precision and strict adherence to academic conventions. Consequently, the authors maintain full accountability for the validity, integrity, and scientific conclusions presented in this work.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper.

REFERENCES

[1] South Jakarta Non-Profit Organization, "Profile Report of Philanthropic Institutions and Leadership Empowerment Programs," Jakarta, 2024.

[2] South Jakarta Non-Profit Organization, "Operational Manual for Alumni Program Collaboration and Networking Platforms," Jakarta, 2024.

[3] R. Dorojhatun, "Business Process Audit and Administrative Limitations of Alumni Tracking," Internal Organizational Interview, Mar. 2025.

[4] B. A. Hilendria, L. T. Junaidi, L. Effendi, and W. Astuti, "Eksistensi Dan Peran Alumni Dalam Menjaga Kualitas Mutu Jurusan Akuntansi Fakultas Ekonomi Dan Bisnis Universitas Mataram," Jurnal Riset Akuntansi Aksioma, vol. 18, no. 2, pp. 48–60, Dec. 2019.

[5] Kinnunen, J., Designing a Node.js full stack web application [Bachelor's thesis], Metropolia University of Applied Sciences, April 3, 2023.

[6] M Barokah., M. & Sauda., S. "Penerapan NodeJS dan PostgreSQL sebagai Backend pada Aplikasi Ecommerce Localla", Infotech Journal, vol.8, no. 22, pp. 101-105, November 2022.

[7] F. Zufari, "Challenges in Source Code Structure and Internal IT System Maintenance," Organizational IT Department Interview, Mar. 2025.

[8] A. J. Chandra, R. Tan, et al., "Pengembangan Back-End dan Perancangan API Docs Website Think Action," Jurnal STRATEGI-Jurnal Maranatha, vol. 6, no. 1, pp. 107–121, 2024.

[9] S.Sugiarto and E. Triandini, "Pengembangan Database E-commerce De Janggelan Menggunakan Metode Database Life Cycle," Jurnal Sistem Dan Informatika (JSI), vol. 16, no. 2, pp. 122–132, 2022.

[10] N. Alfarizi, "Pengembangan Sistem Informasi Ikatan Alumni dengan Menggunakan Arsitektur Microservices (Studi Kasus: Ikatan Alumni Pesantren Al Binaa Islamic Boarding School)," Skripsi, Program Studi Teknik Informatika, Universitas Brawijaya, Malang, Indonesia, 2021.

[11] M. A. Hidayatullah, A. L. Prasasti, and F. C. Hasibuan, "Pembangunan Backend pada Website Aplikasi Fateka: Forum Alumni Teknik Komputer Universitas Telkom," eProceedings of Engineering, vol. 10, no. 5, 2023.

[12] J. S. Manurung and Z. A. Matondang, "Sistem Informasi Halo Alumni Pada Universitas Katolik Santo Thomas Medan", in Seminar Nasional Inovasi Sains Teknologi Informasi Komputer, 2023, pp. 173-182.

[13] W. S. Wahyuni, W. Wagino, and D. Agustini, "Aplikasi Pendataan Alumni dan Pelayanan Legalisir Online pada SMA Negeri 1 Bati-Bati Kab. Tanah Laut Berbasis Web," Jurnal Sains Sistem Informasi, vol. 1, no. 1, pp. 33–41, 2023.

[14] Ramadhan. R. and Ma'sum. H., "Perancangan Sistem Informasi Aplikasi Tracer Study Alumni Berbasis Website pada SMK YSB Suryalaya", Jurnal Nasional Komputasi dan Teknologi Informasi, vol. 8, no. 4, pp. 1927-1936.

[15] Z. Rachmat, A. Irfan, Z. Fadli, and J. Juliana, "Perancangan Sistem Informasi Manajemen Tracer Study Alumni Berbasis Web pada STMIK Amika Soppeng," REMIK: Riset dan E-Jurnal Manajemen Informatika Komputer, vol. 9, no. 1, pp. 102–116, 2025.

[16] R. Purnama and W. Haryono, "Pengembangan Platform Digital Komunitas Alumni Universitas Al-Azhar Berbasis Web Menggunakan Nextjs dan Prisma ORM," OKTAL : Jurnal Ilmu Komputer dan Science, vol. 4, no. 12, pp. 840–845, Dec. 2025.

[17] Z. Niswati, Syukriansyah, and D. Marlina, "Sistem Informasi Tracer Study Berbasis Web Menggunakan Konsep RESTful API Web Service pada SMK Negeri Karang Baru," STRING (Satuan Tulisan Riset dan Inovasi Teknologi), vol. 10, no. 1, Aug. 2025.

[18] M. Al Ghyfari and R. Setiawan, "Perancangan Aplikasi Tracer Study dengan Metode Agile untuk Evaluasi Keselarasan Karier Alumni," *Prosisko: Jurnal Pengembangan & Observasi Sistem Komputer*, vol. 12, no. 2, Jul. 2025. [Online].

[19] Y. N. Nugroho, D. S. Kusumo and M. J. Alibasa, "Clean Architecture Implementation Impacts on Maintainability Aspect for Backend System Code Base," 10th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia, 2022, pp. 134-139, [Online].

[20] K.Rubin,Essential Scrum:A Practical Guide to the Most Popular Agile Process (Addison Wesley signature series). Addison-Wesley, 2012, ISBN: 9780137043293.