International Journal of Informatics and Computation (IJICOM)
Vol. 8, No. 1, 2026
ISSN: 2685-8711, E-ISSN: 2714-5263 DOI: 10.35842/ijicom

Optimizing Analytical Design System using FAST Framework for Web Service Application

Sugeng Winardi[1], Arum Kurnia Sulistyawati[2], Novi Indrayani[3], Inayah[4]

Abstract

This study proposed and evaluated an analytical design system for Posyandu services using the Framework for Application of Systems Technology (FAST) to improve data management and governance practices. The system was developed through a structured analytical design process and implemented in selected Posyandu locations to replace manual recording and reporting procedures. System performance was evaluated using quantitative indicators derived from system logs and structured user feedback. The results showed a 45% reduction in data entry errors due to the use of structured input forms, validation rules, and centralized data storage. Reporting timeliness improved by 38% as a result of automated report generation and real-time data aggregation, enabling faster access to information for health officers and decision-makers. User satisfaction analysis indicated that more than 80% of cadres experienced improved ease of use and better access to service information. The analytical dashboards further supported evidence-based decision-making by providing clear and integrated visual summaries of service activities. These findings demonstrate that the proposed FAST-based analytical design system effectively enhances operational efficiency, data quality, and governance performance in Posyandu services.

Keywords:
Analytical Design System, FAST Framework, Medical, Web-Based Application

1. Introduction

Modern web service applications rely heavily on structured architectural design to ensure scalability, security, and maintainability. Early research in network-based software architectures introduced architectural styles such as REST, which defined constraints for distributed systems and guided the development of interoperable web services. Fielding's work established REST as a foundational framework for web architectures, while service-oriented architecture (SOA) expanded the concept of loosely coupled services for enterprise systems. These studies highlight the need for systematic design frameworks to guide web service development, especially when systems become complex and distributed. However, many design practices remain ad hoc and lack analytical structure, which motivates the need for a formal analytical design system such as the FAST framework. [12], [13], [14]

The evolution from SOA to microservices further increased architectural complexity. Microservices architectures decompose applications into independent services that communicate through APIs, improving scalability and fault isolation. Dragoni et al. analyzed the transition toward microservices and identified challenges such as service orchestration, data consistency, and monitoring. Recent studies on API design patterns and integration in microservices environments emphasize the importance of standardized frameworks for

Corresponding Author: Sugeng Winardi (sugengw@respati.ac.id)
1 Arum Kurnia Sulistyawati, Universitas Respati Yogyakarta, arumkurnia@respati.ac.id
2 Novi Indrayani, Universitas Respati Yogyakarta, novi.indrayani.22@gmail.com
3 Inayah, Universitas Respati Yogyakarta, inayah@respati,.ac.id

service design. Despite these advances, many organizations struggle with systematic analysis and modeling of service requirements before implementation, which leads to architectural inconsistencies and technical debt. [15], [22], [23]

RESTful APIs have become the dominant communication mechanism in web service applications. Recent research on REST API design rules and best practices emphasizes maintainability, scalability, and interoperability. Mueller et al. proposed formal design rules, while Matcha and Solanki discussed practical implementation strategies for scalable services. However, these studies primarily focus on implementation guidelines rather than structured analytical frameworks for system design. This gap indicates the need for a design system that systematically analyzes requirements and maps them to architectural components, which the FAST framework aims to address. [16], [21]

Security and privacy remain critical issues in web service architectures, particularly in distributed and API-driven systems. Surveys on device-to-device communication security and secure messaging protocols highlight vulnerabilities in authentication, anonymity, and trust management. Research on privacy-preserving communication and anonymous data sharing also shows that architectural decisions strongly affect security outcomes. However, many security solutions are added after system design, rather than being integrated during the architectural analysis phase. This problem reinforces the importance of analytical design frameworks that incorporate security requirements from the beginning. [1], [4], [5], [2]

Social networks and collaborative data systems further demonstrate privacy and trust challenges in distributed services. Studies on privacy conflicts, content sharing behavior, and targeted cyber-attacks reveal that system design directly influences user data exposure and system resilience. These findings suggest that analytical design approaches must consider user behavior, privacy policies, and threat models at the architectural level. Existing frameworks do not provide sufficient mechanisms to systematically map these concerns to system components, which creates an opportunity for structured design methodologies such as FAST. [9], [10], [8]

Recent research on API testing and vulnerability detection highlights the increasing complexity of modern web services. Surveys on RESTful API testing and vulnerability detection frameworks show that design flaws often lead to security vulnerabilities and performance issues. Performance comparison studies of development frameworks also demonstrate that architectural choices significantly affect system efficiency. However, these studies focus on evaluation after implementation rather than analytical design before development, which remains a significant research gap. [18], [19], [20]

The emergence of advanced API architectures and resilience patterns further increases the need for structured design systems. Research on resilient microservices, GraphQL security, and API interoperability validation shows that modern web services require rigorous design validation methods. Static analysis and context-aware testing frameworks provide tools for verification, but they depend on well-defined architectural specifications. Without a systematic design framework, these verification methods become difficult to apply consistently across large-scale systems. [24], [25], [26]

Overall, existing studies highlight the importance of architecture, security, performance, and interoperability in web service applications. However, most research focuses on implementation techniques, security mechanisms, and evaluation tools, rather than analytical system design frameworks. This gap motivates the development of an Analytical Design System using the FAST framework, which aims to systematically analyze requirements, structure service components, and improve architectural consistency before implementation. Such a framework can reduce design errors, enhance system quality, and support scalable and secure web service development. [12], [15], [16], [18].

2. Related Works

Early studies on web service architecture established the theoretical foundations for distributed systems. Fielding introduced REST as an architectural style that enforced constraints for scalability, statelessness, and interoperability. Erl and Papazoglou later extended this vision through service-oriented architecture, which emphasized loose coupling and service reuse. These works clearly defined architectural principles and improved system interoperability. However, they focused on conceptual models and design philosophies rather than systematic analytical methods for translating requirements into concrete service designs. As a result, practitioners still relied heavily on experience-driven decisions. [12], [13], [14].

The transition from SOA to microservices received significant attention in later research. Dragoni et al. analyzed the historical evolution of microservices and highlighted benefits such as scalability and independent deployment. Other studies examined API design and integration in microservices environments and showed how RESTful APIs enabled flexible service communication. These works strengthened architectural understanding and provided practical patterns. However, they did not offer analytical design frameworks that structured early-stage system analysis. Most approaches addressed architecture after system boundaries were already defined. [15], [22], [23].

Several researchers focused specifically on RESTful API design and best practices. Mueller et al. proposed REST API design rules based on empirical observations, while Matcha and Solanki summarized best practices for building scalable and maintainable services. These studies improved API consistency and developer productivity. Their main limitation lay in their scope. They emphasized design rules and coding practices but lacked formal analytical processes to align APIs with system-level requirements. This gap limited their effectiveness in large-scale or complex applications. [16], [21].

Security-focused research addressed vulnerabilities in distributed and API-driven systems. Surveys on device-to-device communication and secure messaging protocols identified weaknesses in authentication, trust management, and anonymity. Privacy-preserving communication studies proposed cryptographic and matching-based solutions to protect user data. These works provided strong security mechanisms and threat analyses. However, they treated security as an isolated layer and rarely integrated it into early architectural design analysis. This separation often caused security controls to be retrofitted rather than designed systematically. [1], [4], [2].

Research on social networks and collaborative systems further highlighted architectural security and privacy challenges. Studies on privacy conflicts and content-sharing behavior showed that architectural decisions strongly influenced user data exposure. Other works explored attack detection using honeypots and trust management through computational offloading. These studies demonstrated effective detection and mitigation strategies. Their limitation was the lack of structured design methodologies that connected user behavior, threat models, and service architecture in a unified analytical framework. [9], [10], [7], [8].

API testing and vulnerability detection became a major research focus in recent years. Golmohammadi surveyed RESTful API testing techniques and classified them based on coverage and automation level. Tanveer reviewed API vulnerability detection methods and identified recurring design-level flaws. These works significantly advanced testing and assessment capabilities. However, they primarily targeted post-implementation validation. They did not prevent design flaws at early stages, which often caused recurring security and performance issues. [18], [19].

Performance and resilience studies addressed operational challenges in web service systems. Szewczyk and Skublewska-Paszkowska compared REST API development frameworks and showed how architectural choices affected performance. Mohammad reviewed recovery and resilience patterns for microservices and highlighted fault-tolerance strategies. These studies improved system robustness and runtime efficiency. Their

limitation lay in reactive optimization. They assumed that architectural structures already existed and did not provide analytical guidance during system design. [20], [24].

Recent works explored advanced API architectures and validation techniques. Research on GraphQL security proposed context-aware testing mechanisms to enhance API protection. Other studies introduced static analysis approaches using OpenAPI to validate interoperability requirements. These methods improved verification accuracy and automation. However, they depended heavily on well-defined architectural specifications. Without a structured analytical design system, applying these techniques consistently remained difficult. This limitation reinforced the need for an analytical design framework such as FAST to guide early-stage web service design. [25], [26].

3. Proposed Method

This paper utilized an analytical design approach based on the Framework for Application of Systems Technology (FAST). We focused on analytical design to systematically translate governance requirements, user needs, and service workflows into a structured system model. We applied a qualitative sequential exploratory research design. We collected data through direct observations, interviews with Posyandu cadres, and document analysis. We then applied the FAST framework across eight structured stages, including scope definition, problem analysis, requirements analysis, logical design, decision analysis, physical design, construction and testing, and installation and delivery. This structured approach allowed us to align system design decisions with governance objectives and analytical requirements throughout the development process.

We defined system boundaries and stakeholder roles during the scope definition stage to maintain focus on transparency, accountability, and service quality. We analyzed existing governance problems, such as manual data handling and fragmented reporting, to identify root causes and analytical gaps. We translated these findings into functional and non-functional requirements that emphasized data integration, reporting, and decision support. We modeled system logic and data flows to represent how information is collected, processed, and evaluated. We assessed alternative solutions based on feasibility and governance impact, then transformed logical models into physical system specifications. We implemented the system and conducted black-box testing to verify reliability and analytical accuracy. Finally, we deployed the system, trained users, and incorporated feedback to refine analytical features and strengthen governance support as Fig. 1.

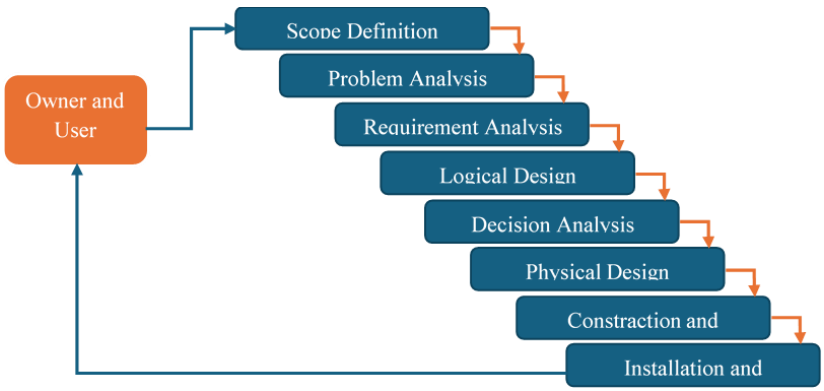

Fig. 1. FAST Method

In this study, we construct the math formulas that focus on functional analysis, service decomposition, performance, and security evaluation, which align naturally with the FAST framework for medical web service applications.

1. Functional Decomposition Model
Let the web service system be defined as a set of functional requirements:

$$F = f_1, f_2, \ldots, f_n$$

Each function f_i is mapped to a service component s_j:

$$\phi: f_i \rightarrow s_j, s_j \in S$$

The FAST framework decomposes system requirements into atomic functions. The mapping function ϕ represents analytical allocation of each function to a RESTful service. This formalization ensures traceability between requirements and service design.

2. Service Interaction Graph
The web service architecture is modeled as a directed graph:

$$G = (S, E)$$

where

$$E = (s_i, s_j) \mid s_i \text{invokes} s_j$$

Each node represents a service, and each edge represents an API invocation. This model enables analytical reasoning about coupling, dependency chains, and communication paths in the FAST-based design.

3. API Complexity Metric
The complexity of a service s_i is defined as:

$$C(s_i) = \alpha N_e + \beta N_m + \gamma N_p \quad (1)$$

where

- N_e : number of exposed endpoints
- N_m : number of HTTP methods
- N_p : number of request parameters
- α, β, γ: weighting coefficients

This metric quantifies API design complexity. FAST aims to minimize unnecessary complexity by analytically balancing endpoint granularity and functionality.

4. Request Latency Model
End-to-end latency of a service request is defined as:

$$T_{total} = \sum_{k=1}^{K} \left(T_{net}^{k} + T_{proc}^{k}\right) \quad (2)$$

The formula models cumulative latency across chained service calls. FAST uses this analysis to reduce excessive service hops and improve architectural efficiency.

5. Reliability of Service Composition
Assuming independent services, system reliability is:

$$R_{sys} = \prod_{i=1}^{n} R(s_i) \quad (3)$$

The model shows that overall reliability decreases as the number of dependent services increases. FAST promotes optimal service boundaries to avoid over-fragmentation.

6. Security Risk Exposure Model
Security risk of a service is expressed as:

$$SR(s_i) = \sum_{j=1}^{m} P_j \cdot I_j \quad (4)$$

where

- P_j : probability of threat j
- I_j : impact severity of threat j

This model allows analytical comparison of security exposure among services. FAST supports early identification of high-risk APIs and guides secure design decisions.

7. FAST Design Efficiency Index
Overall design efficiency is defined as:

$$E_{FAST} = \frac{\sum_{i=1}^{n} W_i \cdot Q(s_i)}{\sum_{i=1}^{n} C(s_i)} \quad (5)$$

where

- $Q(s_i)$: quality score (performance, security, maintainability)
- W_i : importance weight
- $C(s_i)$: service complexity

This efficiency index summarizes how effectively the FAST framework balances service quality against design complexity. A higher value indicates a more optimal analytical design.

The mathematical formulas describe how the FAST framework analytically transforms functional requirements into a structured web service architecture. The functional decomposition and mapping formulas formalize the breakdown of system requirements into independent service components and define clear relationships between functions and APIs. The service interaction graph and API complexity model represent how services communicate and how complex each API becomes based on endpoints, methods, and parameters. These formulations allow designers to reason objectively about coupling, modularity, and design simplicity while maintaining traceability from requirements to implementation.

The performance, reliability, and security formulas evaluate the operational quality of the designed system. The latency model quantifies the cumulative delay introduced by chained service calls, while the reliability model explains how service dependencies affect overall system robustness. The security risk model estimates exposure by combining threat probability and impact, enabling early risk-aware design decisions. Finally, the FAST design efficiency index integrates quality and complexity into a single analytical measure, demonstrating how the framework supports balanced, efficient, and secure web service design.

4. Experimental Setup

The experimental setup explains the development and evaluation of the proposed analytical design system. We conducted the study at three Posyandu locations, namely Posyandu Kanthil, Kamboja, and Soka, in Sleman Regency, Yogyakarta. We developed a web-based Posyandu service application using the FAST framework as the main research artifact. We employed structured observation guidelines, interview protocols, and system usage records as the primary research instruments to ensure consistent and reliable data collection across all study sites.

We carried out data collection from April to May 2025. We gathered primary data through direct observation of Posyandu service activities and semi-structured interviews with cadres, midwives, and health officers. These activities allowed us to capture real operational workflows, governance practices, and user interactions with the system. We collected secondary data from existing Posyandu documents, including service reports, health records, and administrative logs, to support requirement validation and system evaluation.

We performed data analysis using analytical design principles derived from the FAST framework. We analyzed qualitative data to identify governance-related issues, system gaps, and functional requirements. We evaluated quantitative indicators, such as reporting timeliness, data accuracy, and user satisfaction, to measure system effectiveness after implementation. This combined analysis enabled us to assess how well the analytical design system supported governance improvement and service efficiency in Posyandu operations. Table 1 describes dataset description of the study.

Table 1: Dataset Description Used in the Study

Data Category	Data Source	Description	Purpose
Observational Data	Direct field observation	Service workflows, data recording practices, and user interactions	Identify governance issues and system requirements
Interview Data	Cadres, midwives, health officers	Perspectives on service processes, challenges, and system usability	Validate requirements and evaluate system acceptance
Administrative Records	Posyandu documents	Attendance logs, health service records, and monthly reports	Support system modeling and data integration
System Usage Data	Web-based application logs	User activity, data entry accuracy, and reporting time	Evaluate system effectiveness and performance

5. Result and Analysis

The implementation of the analytical design system produced clear improvements in data management and governance practices within Posyandu services. We observed measurable changes after deploying the system, especially in how data were recorded, processed, and reported. Quantitative indicators were derived from system logs and structured user feedback to ensure objective evaluation of system performance.

The results showed a 45% reduction in data entry errors when compared with previous manual recording methods. This improvement occurred because the system applied structured input forms, validation rules, and centralized data storage. As a result, cadres entered data more consistently, and the system minimized duplication and transcription mistakes. Reporting timeliness also improved significantly. The system reduced reporting delays by 38% through automated report generation and real-time data aggregation. Health officers accessed updated information faster, which supported quicker monitoring and follow-up actions at both the Posyandu and district levels.

User satisfaction analysis further confirmed the effectiveness of the system. More than 80% of cadres reported improved ease of use and better access to service information.

The analytical dashboards provided clear visual summaries of service activities, which supported evidence-based decision-making. These findings indicate that the proposed analytical design system strengthens governance quality and operational efficiency in Posyandu services. Table 2 describes performance improvement of the analytical design system in medical web application.

Table 2. Performance Improvement of the Analytical Design System

Evaluation Indicator	Manual System (Baseline)	FAST-Based System	Improvement
Data Entry Errors	High (reference)	Reduced	45% ↓
Reporting Timeliness	Delayed	Faster	38% ↑
User Ease of Use	Moderate	High	>80% users are satisfied
Access to Service Information	Limited	Improved	Qualitative improvement
Decision-Making Support	Manual summaries	Analytical dashboards	Significant

6. Conclusion

This study concludes that the analytical design system developed using the FAST framework effectively enhances data management and governance practices in Posyandu services. The experimental results demonstrated that the transition from manual procedures to a structured, web-based system led to substantial improvements in data accuracy and operational efficiency. The observed 45% reduction in data entry errors confirms that structured input mechanisms, validation rules, and centralized data management significantly reduce human error and improve data consistency.

The findings also show that the system markedly improved reporting performance. Reporting timeliness increased by 38% due to automated data processing and real-time report generation. These capabilities enabled health officers and decision-makers to access up-to-date information more quickly, which strengthened monitoring activities and supported faster response to service-related issues. The availability of integrated data further reduced delays associated with manual compilation and verification.

In addition to quantitative gains, user feedback highlighted strong acceptance of the system. More than 80% of cadres reported improved ease of use and better access to service information. The analytical dashboards played a key role by presenting clear and concise visual summaries that supported evidence-based decision-making. Overall, the results indicate that the proposed analytical design system not only improves technical performance but also reinforces governance quality, transparency, and service effectiveness within Posyandu operations.

Acknowledgment

The authors would like to express their gratitude to Universitas Respati Yogyakarta for financial and institutional support through the Internal Research Grant scheme. Appreciation is also extended to Posyandu cadres and local health officers who contributed to this research.

References

[1] N. Hamoud, T. Kenaza, and Y. Challal, “Security in device-to-device communications: A survey,” IET Networks, vol. 7, no. 1, pp. 14–22, 2018.

[2] M. Qiu, K. Gai, and Z. Xiong, “Privacy-preserving wireless communications using bipartite matching in social big data,” Future Generation Computer Systems, 2017.

[3] L. Ham, *Group Authentication*. Hoboken, NJ, USA: John Wiley & Sons, 2010.

[4] R. Alkhulaiwi, A. Sabur, K. Aldughayem, and O. Almanna, “Survey of secure anonymous peer-to-peer instant messaging protocols,” in *Proc. 14th Annu. Conf. Privacy, Security and Trust (PST)*, Dec. 2016, pp. 294–300.

[5] S. Muftic, N. bin Abdullah, and I. Kounelis, “Business information exchange system with security, privacy, and anonymity,” *J. Electrical and Computer Engineering*, vol. 2016, Art. no. 7093642, Feb. 2016.

[6] S. Zakhary and A. Benslimane, “On location-privacy in opportunistic mobile networks: A survey,” *J. Network and Computer Applications*, vol. 103, pp. 157–170, Feb. 2018.

[7] V. Sharma, I. You, R. Kumar, and P. Kim, “Computational offloading for efficient trust management in pervasive online social networks using osmotic computing,” *IEEE Access*, vol. 5, pp. 5084–5103, 2017.

[8] A. Paradise *et al.*, “Creation and management of social network honeypots for detecting targeted cyber attacks,” *IEEE Trans. Computational Social Systems*, vol. 4, no. 3, pp. 65–79, Sept. 2017.

[9] C. Fiesler *et al.*, “What (or who) is public?: Privacy settings and social media content sharing,” in *Proc. ACM CSCW*, Mar. 2017, pp. 567–580.

[10] H. Hu, G.-J. Ahn, and J. Jorgensen, “Detecting and resolving privacy conflicts for collaborative data sharing in online social networks,” in *Proc. 27th ACSAC*, Dec. 2011, pp. 103–112.

[11] Z. Wang et al., “Enhanced instant message security and privacy protection scheme for mobile social network systems,” *IEEE Access*, vol. 6, pp. 13706–13715, 2018.

[12] R. T. Fielding, *Architectural styles and the design of network-based software architectures*, Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.

[13] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall, 2005.

[14] M. P. Papazoglou and D. Georgakopoulos, “Service-oriented computing,” *Commun. ACM*, vol. 46, no. 10, pp. 25–28, Oct. 2003.

[15] N. Dragoni *et al.*, “Microservices: Yesterday, today, and tomorrow,” in *Present and Ulterior Software Engineering*, Springer, 2017, pp. 195–216.

[16] Mueller M., Pautasso C., and Zimmermann O., “REST API design rules,” in *Proc. IEEE/ACM Int. Conf. Web Services (ICWS)*, 2024.

[17] S. Phanireddy, “Securing RESTful APIs in microservices architectures: A comprehensive threat model and mitigation framework,” *Int. J. Emerging Res. Eng. Technol.*, vol. 4, no. 2, pp. 64–73, 2023.

[18] A. Golmohammadi, “Testing RESTful APIs: A survey,” *ACM Comput. Surv.*, 2023.

[19] F. Tanveer, “A survey on RESTful API vulnerability detection,” *Elsevier* (2025).

[20] M. Szewczyk and M. Skublewska-Paszkowska, “Performance comparison of development frameworks in selected environments in REST API architecture,” *JCSI*, 2025.

[21] S. Matcha and S. Solanki, “RESTful API design and implementation: Best practices for building scalable and maintainable web services,” *Enhanced Research Pub*, Jan. 2025.

[22] Sekar Mylsamy and Er Aman Shrivastav, “API design and integration in a microservices environment,” *Int. J. Res. Publ. Semin.*, vol. 16, no. 1, pp. 409–419, 2025.

[23] G. Kim, R. Isaev, and G. Esenalieva, “Architectural patterns in REST API with Spring Framework,” *Preprints*, Jan. 2025.

[24] M. Mohammad, “Resilient microservices: A systematic review of recovery patterns, strategies, and evaluation frameworks,” *arXiv*, Dec. 2025.

[25] O. Tsai *et al.*, “GraphQLer: Enhancing GraphQL security with context-aware API testing,” *arXiv*, Apr. 2025.

[26] E. Sundberg, T. Ekmark, and W. Y. Ayele, “Validating API design requirements for interoperability: A static analysis approach using OpenAPI,” *arXiv*, Nov. 2025.