TELKOMNIKA Telecommunication Computing Electronics and Control
Vol. 24, No. 1, February 2026, pp. 34~48
ISSN: 1693-6930, DOI: 10.12928/TELKOMNIKA.v24i1.27261

Performance evaluation of serverless cloud-native API deployment: a case study on a mobile health application

Maulana Bintang Irfansyah, Bilal Waheed, Idris Winarno, Akhmad Alimudin
Department of Informatics and Computer Engineering, Politeknik Elektronika Negeri Surabaya, Surabaya, Indonesia

Article Info

Article history:

Received Jun 5, 2025
Revised Oct 29, 2025
Accepted Dec 8, 2025

Keywords:

Application programming interface
Cloud computing
Google Cloud platform
Mobile application
Platform as a service

ABSTRACT

As software applications become increasingly complex, there is a growing need for scalable, flexible, and high-performance backend solutions. Cloud computing-based application programming interfaces (APIs) address these demands by enabling developers to offload resource-intensive tasks to the cloud while eliminating the burden of infrastructure management. This study presents a case study using Obesifix, a mobile health application for real-time dietary monitoring and personalized nutrition recommendations. Two deployment models were evaluated: a traditional server-based architecture using Google Compute Engine (GCE) and a serverless approach using Google Cloud Run (GCR). Performance testing was conducted with Apache JMeter under simulated loads of 60, 120, and 180 users across four critical API endpoints (register, login, recommendation, prediction). Results show that GCR consistently achieved 20–30% lower response times and 15–20% higher throughput compared to GCE, while maintaining 0% error rate, lower memory consumption, and more balanced virtual central processing unit (vCPU) utilization. Time to first byte (TTFB) remained below 800 ms across all scenarios, confirming good server responsiveness. These findings highlight the scalability and efficiency benefits of serverless architectures for mobile health applications. Future research should explore asynchronous programming paradigms, autoscaling thresholds, and cost-performance trade-offs, as well as multi-cloud deployments to enhance system resilience and generalizability.

Corresponding Author:

Maulana Bintang Irfansyah
Department of Informatics and Computer Engineering
Politeknik Elektronika Negeri Surabaya
Surabaya, Indonesia
Email: maulanabin@pasca.student.pens.ac.id

1. INTRODUCTION

Mobile applications have become essential tools in today's digital landscape, driving innovation and enhancing daily life across various sectors, including communication, entertainment, healthcare, and productivity [1]. A forecast by Statista, a leading provider of market and consumer data, estimates that the number of global smartphone users will reach 6.93 billion approximately 85.74% of the world's population and is projected to exceed 7.7 billion by 2028 [2]. Smartphones account for approximately 90% of all mobile devices and are used by 94.2% of internet users aged 16 and above [3]. The growth highlights the critical role of mobile applications in supporting every day and essential tasks. As features such as artificial intelligence (AI), augmented reality (AR), and real-time data processing become standard, applications demand increasingly complex and computationally intensive backend infrastructures.

Although frontend development is often prioritized to ensure a smooth user experience, the overall performance of mobile applications relies heavily on robust backend systems to maintain responsiveness, reliability, and scalability [4]. Backend services must handle concurrent users, dynamic data, and real-time communication [5]. However, many backend architectures still experience performance bottlenecks, particularly delayed application programming interface (API) response times under high concurrency levels. Such challenges underscore the need for optimized cloud-based solutions to reduce latency and improve data processing efficiency [6]. Figure 1 illustrates the interaction between frontend applications, APIs, backend services, and databases, showing how data flows across system components.

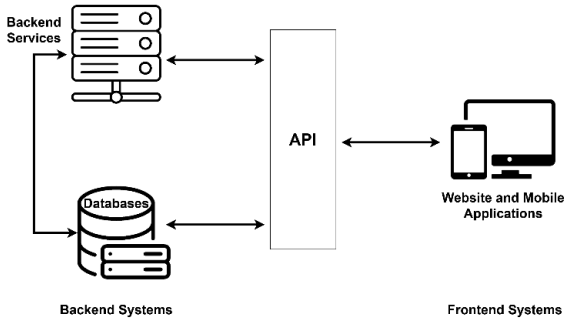

Figure 1. Communication flow between frontend and backend systems

Cloud computing has emerged as a foundational technology to address these challenges, generally defined as the on-demand consumption of computing power, storage, and applications over the internet. The model follows a pay-as-you-go pricing approach, with resources delivered through globally distributed cloud service providers [7]. Cloud platforms enable rapid deployment, operational efficiency, scalability, and global availability [8]. Such capabilities have encouraged many organizations to migrate traditional information technology (IT) infrastructure to cloud-based systems, enabling services such as internet of things (IoT) solutions, web applications, and big data analytics [9]. The combination of lower operational cost, flexible architecture, and simplified integration procedures has driven widespread adoption of cloud computing across industries, aligning with Industry 4.0 digital transformation goals [10]. Offloading resource-intensive operations to the cloud allows mobile systems to maintain high client-side performance while ensuring backend scalability [11]. Cloud-based services including distributed storage, real-time analytics, and managed databases enable mobile applications to scale efficiently without placing excessive demand on device resources [12]. Cloud-based APIs serve as key connectors between applications and cloud services, facilitating seamless integration and enhancing system performance.

Recent research has increasingly explored the integration of cloud computing into mobile application development across a range of domains and architectures. Studies in domain-specific contexts such as sign-language translation, image-based barter platforms, ecotourism services, and tourism education commonly employ representational state transfer (REST) APIs on managed cloud platforms such as Google Cloud to offload computation and streamline client-side interaction [13]–[16]. These studies confirm the feasibility in production-like scenarios and provide insights into full-stack workflows; however, most remain focused on functionality without quantitative evaluations of latency, time to first byte (TTFB), or API modularity under dynamic traffic conditions. Beyond isolated implementations, architectural investigations have proposed containerized microservices for release agility and serverless designs for elastic scaling in commercial backends [17]–[19]. While orchestration strategies and deployment flexibility are emphasized, comparative assessments across deployment models remain limited, particularly in evaluating API responsiveness under concurrent access. Moreover, reusable design patterns at the endpoint level are often described informally or qualitatively, lacking codification for replication. Cloud-based mobile systems supporting education and mental health demonstrate development and operations (DevOps) pipelines, AI-driven personalization, and multimodal interfaces such as natural language processing for emotional support or interactive data visualization [20]–[22]. These implementations prioritize usability and user engagement; however, performance under fluctuating network conditions or heavy concurrent usage remains understudied, leaving essential questions around scalability, responsiveness, and maintainability unaddressed. Despite these developments, few studies have offered an end-to-end evaluation of cloud-native API performance.

As illustrated in Figure 2, cloud platforms offer three service models software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) allowing developers to choose the appropriate level of abstraction based on system requirements [14], [23].

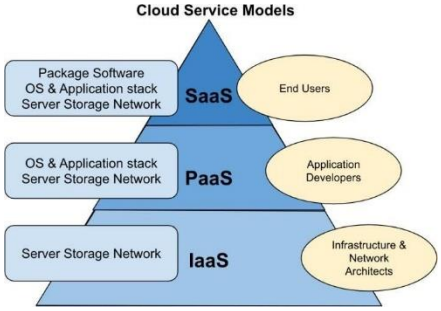

Figure 2. Cloud computing service model [24]

The infrastructure for the developed application is built on Google Cloud platform (GCP), which provides scalable services through a serverless architecture, enabling the application to automatically adjust resources under dynamic workloads. The selection of GCP is supported by its recognition as a leader in Gartner's Magic Quadrant for Strategic Cloud Platform Services [25], with strengths in AI integration, workload optimization, and reliability. Choosing GCP ensures that the experimental results are representative of a production-ready, enterprise-grade environment, aligning to evaluate API performance and scalability under realistic deployment conditions.

Building on previous works, a prior study introduced a mobile health application, Obesifix, that leverages cloud computing to support real-time dietary monitoring and personalized nutrition recommendations. The application was designed to optimize resource usage in mobile environments by distributing computational workloads to the cloud, enabling users to track nutritional intake and make informed dietary decisions based on real-time data. The study demonstrated that leveraging flexible cloud systems can improve mobile application performance under dynamic user demands [26]. The main contributions of our study include a comprehensive evaluation of API design within cloud-native frameworks, offering actionable insights for developers aiming to enhance efficiency and responsiveness in mobile applications. A modular API architecture is proposed, leveraging serverless deployment using Google Cloud Run and managed services such as Cloud structured query language (SQL) and Cloud Storage. System performance was evaluated by simulating user traffic with Apache JMeter, focusing on key performance indicators including average response time, minimum and maximum values, standard deviation, and throughput. The findings provide valuable insights into optimizing cloud resource allocation and offer practical guidelines for designing responsive, resource-efficient mobile applications.

The structure of this paper is as follows: section 2 outlines the methodology used for designing and implementing the Obesifix application. Section 3 discusses the experimental results and provides a comparative analysis. Finally, section 4 summarizes the main findings and highlights potential directions for future work.

2. METHOD

The method section outlines the end-to-end development and deployment process, including workflow design, authentication, and container-based deployment on Google Cloud, as well as API design and integration; the objective is to enable reproducible evaluation of the proposed cloud-native applications.

2.1. System workflow diagram

Figure 3 illustrates the system workflow for cloud-based API integration within a full-stack mobile application. The workflow is divided into three main segments: Part A (initial backend preparation), Part B (core cloud integration), and Part C (final application delivery).

In Part A, development starts by building server-side logic, including third-party service integration such as external APIs and authentication systems. Backend functionalities are encapsulated within an API-based architecture that establishes the communication pathway between server components and client interfaces. The resulting structure supports modular logic implementation, simplifying maintenance and

reuse. Part B, marked with a blue background, emphasizes the central implementation of the cloud environment. The process begins with establishing database connectivity through Google Cloud SQL, which manages data storage and retrieval using object-relational mapping (ORM) or SQL-based queries. Subsequently, routing logic and middleware are configured using the Express.js framework to handle client requests, authorization, and data validation. The configured backend is then deployed via Google Cloud Run, a serverless platform that offers autoscaling and load balancing based on traffic intensity. The mobile application, developed in Kotlin, communicates with the backend through securely exposed API endpoints, enabling interactions for features such as login, registration, recommendation, and prediction. Part C represents the final integration stage, where both frontend and backend components converge into a fully operational mobile application. The resulting system can deliver real-time services efficiently, backed by a scalable cloud-based architecture that maintains stable performance across varying user activity levels.

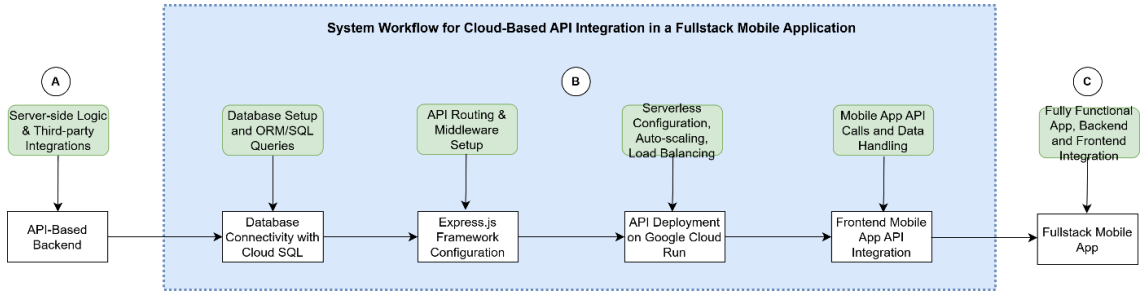

Figure 3. System workflow diagram

To further elaborate on the workflow in Part B, Figure 4 illustrates the detailed authentication and deployment process applied within the system. The client initiates a login request directed to the backend API hosted on Google Cloud Run. Upon receiving the request, the API generates a JSON Web Token (JWT) to perform authentication and proceeds to verify user credentials by querying Cloud SQL. A valid token enables the client to access protected endpoints for continued interaction.

Regarding deployment, the backend and machine learning components are packaged into containerized services and published through Docker images stored in the Google Cloud Artifact Registry. Such a configuration supports automated scaling and uniform deployment across environments. The diagram shows how client devices, authentication systems, databases, and deployment processes work together, illustrating a setup that can manage multiple users simultaneously while keeping cloud operations secure and organized.

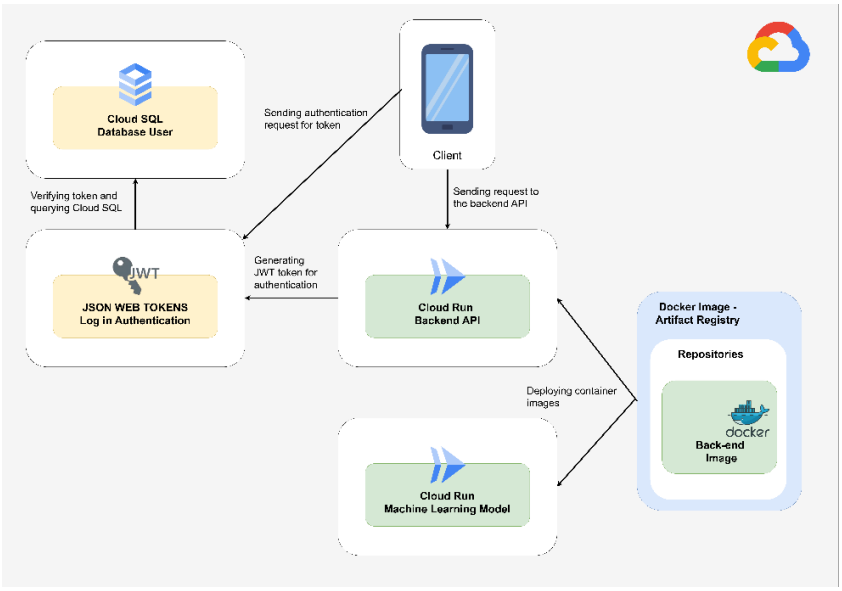

Figure 4. Deployment architecture on Google Cloud

2.2. Cloud-based API design and integration

Cloud-based APIs are the foundational layer that facilitates seamless interaction between mobile applications and cloud infrastructure, ensuring consistent and efficient data flow. Figure 5 illustrates the backend API architecture developed using the Express.js framework. Express.js's modular and lightweight structure enables effective implementation of routing logic, middleware processing, and scalable service delivery.

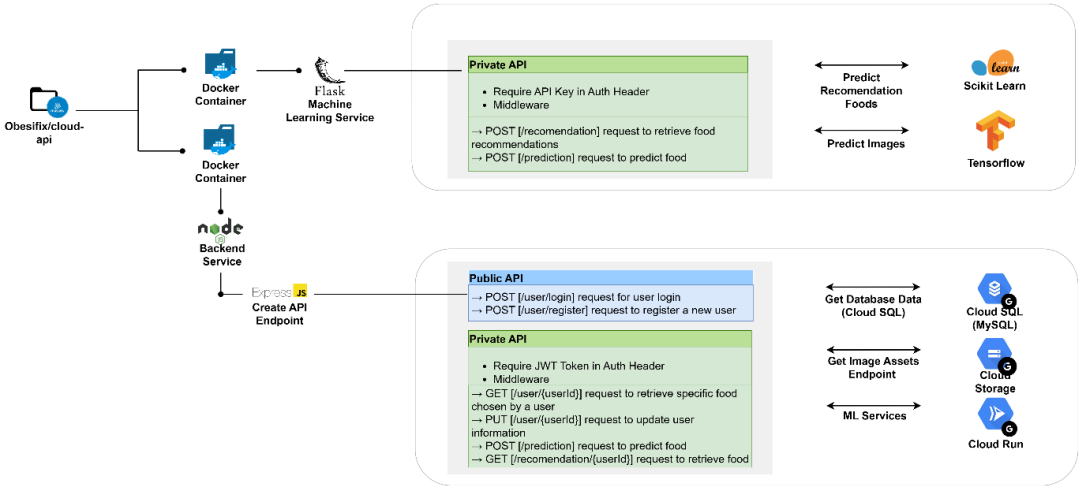

Figure 5. Cloud-based API design

API deployment is done on Google Cloud Run, a managed serverless platform capable of autoscaling based on incoming request volumes. Manual server provisioning becomes unnecessary, and system responsiveness is preserved even under fluctuating user demand. Communication between the mobile client and cloud services is established through securely defined endpoints. Cloud SQL (MySQL) manages structured user data, supporting real-time access and updates. Cloud Storage hosts static assets, including image resources used for data-driven features. Backend logic and machine learning modules are packaged into Docker containers and managed via Google Cloud Artifact Registry to support consistent and portable deployments. Authentication procedures rely on JWT, which restrict access to protected API endpoints. Public APIs handle user-facing processes such as login and account registration, while private APIs are designated for personalized tasks, including food predictions and dietary recommendations. Machine learning capabilities are embedded in Flask-based containers, utilizing models built with TensorFlow and Scikit-learn to process analytical requests submitted by users.

The overall architecture enables a modular, scalable, and secure full-stack system. Developers are positioned to focus on enhancing application features, while cloud-managed infrastructure ensures reliability, load adaptability, and operational efficiency across environments. In addition to the API infrastructure, the Obesifix application integrates machine learning components to support automated food classification and personalized recommendations, as described in the following subsection.

2.3. Dataset and model deployment

A specially curated dataset, comprising proprietary food imagery and user preference data, was employed in this study to support the development of the Obesifix application. The image dataset comprises 19 food categories (e.g., apple, banana, chicken curry, donuts, rice, spaghetti, and sushi), which are used for automated food classification. Additionally, a user profile dataset was created, containing food preferences (19 types) and health conditions (underweight, normal, overweight, and obese), enabling personalized dietary recommendations.

Two machine learning models were developed for this study. The first is an image classification model, built using transfer learning with InceptionV3 in TensorFlow and Keras, and trained on the curated food image dataset consisting of 19 categories. The second is a recommendation model, implemented using the k-nearest neighbors (KNN) algorithm to generate personalized meal recommendations based on each user's recorded food preferences and health conditions. Together, these models enable the Obesifix application to automatically classify food images and suggest meals aligned with users' dietary needs.

The classification model obtained 96.40% training accuracy and 88.20% validation accuracy, with training/validation loss curves indicating stable convergence and minimal overfitting. Figure 6 shows the training and validation accuracy curves, demonstrating that the model achieves consistently high accuracy across epochs. Figure 7 shows the training and validation loss curves, where the loss decreases steadily and remains stable, confirming good generalization capability.

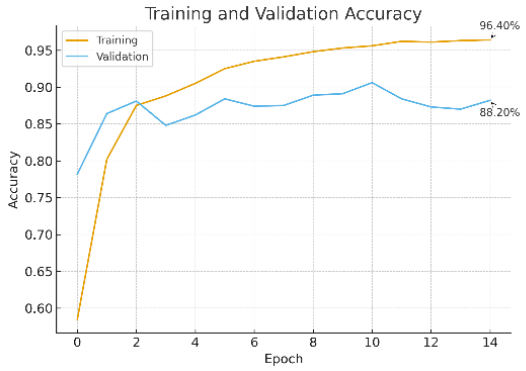

Figure 6. Training and validation accuracy

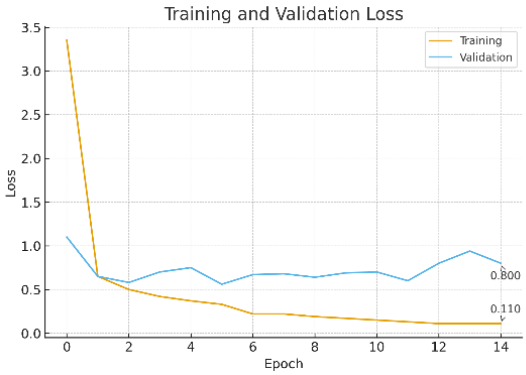

Figure 7. Training and validation loss

Both models were containerized and deployed as RESTful APIs using Google Cloud Run. This serverless approach allows automatic scaling based on incoming request traffic, ensuring low-latency predictions and cost efficiency. The classification API processes user-uploaded images and returns predicted food labels, along with their nutritional composition, whereas the recommendation API filters suitable meal options based on the user's health profile and preferences. These containerized services were subsequently incorporated into the performance testing and comparison experiments to evaluate their responsiveness under server-based (Google Compute Engine (GCE)) and serverless (Google Cloud Run (GCR)) environments.

3. RESULTS AND DISCUSSION

The section reports the experimental setup and findings, outlining cloud services and configurations for backend deployment, presenting endpoint-level performance under stepped loads, and then providing TTFB analysis with a brief overview of user-interface considerations.

3.1. Google Cloud services overview

To support the deployment and operation of the Obesifix mobile application, a suite of Google Cloud services was utilized to build a robust, scalable, and cloud-native backend infrastructure. The selected services were specifically chosen to meet the demands of Obesifix, which requires dynamic scalability, secure data handling, and fast response times to ensure a seamless user experience. Cloud Run was used for backend API deployment due to its serverless model and autoscaling capability, which is crucial under variable user traffic. Cloud SQL manages structured nutritional data and user records, while cloud storage handles image assets for food recognition. Artifact registry stores and manages Docker container images for both backend and machine learning services. Configuration details for each service instance are presented in Table 1.

Table 1. Google Cloud platform services configuration

Services	Resources	Details
Cloud Run (serverless)	Memory	8 GB
	vCPU	4
	Placement location	Southeast-Asia1 – Jakarta
Cloud SQL	vCPU	2
	Memory	8 GB
	Storage type	HDD 10 GB
	Location	Southeast-Asia1 – Jakarta
Cloud storage	Location	Southeast-Asia1 – Jakarta
	Storage class	Standard
Artifact registry	Format	Docker
	Placement location	Southeast-Asia1 – Jakarta

3.2. Performance testing

Table 2 outlines the test scenario configuration used in the evaluation process. Each endpoint was tested under three simulated traffic loads: 60, 120, and 180 virtual users. The chosen load levels represent realistic usage patterns ranging from moderate to heavy traffic without exceeding system limitations.

In this performance evaluation, the Obesifix application was tested on two deployment models: a server-based approach using GCE and a serverless approach using GCR. The comparison was carried out across four critical API endpoints register, login, recommendation, and prediction under three different load levels (60, 120, and 180 virtual users). The results for each endpoint are presented in Tables 3 through 10.

In this study, performance testing focused on four key API endpoints: register, login, recommendation, and prediction. The selected endpoints represent the most frequently accessed and computation-intensive operations within the Obesifix application. Registration and login serve as essential components for user authentication, while the recommendation and prediction endpoints are responsible for delivering personalized dietary services powered by machine learning models. Evaluating the performance of each targeted endpoint provides valuable insights into application behavior under high-concurrency conditions, particularly for user management and personalized feature execution both crucial for ensuring system responsiveness and user satisfaction.

Table 2. Test scenario for performance testing

No.	API		Test scenario
	Parameter	Endpoint	Number of threads (users)
1	POST	/register	60, 120, 180
2	POST	/login	60, 120, 180
3	POST	/prediction	60, 120, 180
4	POST	/recommendation	60, 120, 180

Table 3. Result for concurrent register – Compute Engine

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	720	640	890	55	0.00%	25
120	845	750	1040	70	0.00%	47
180	880	770	1100	80	0.00%	68

The performance testing of the Obesifix application was conducted using Apache JMeter to evaluate its performance under varying traffic levels. Apache JMeter is a widely recognized performance testing tool known for its capability to simulate different load conditions and measure response times. According to research by [27], [28], Apache JMeter has proven effective for testing mobile applications under heavy usage. The tool is particularly useful for assessing how applications perform under increased demand and identifying potential bottlenecks in the system. These studies have validated the tool's efficacy for performance testing software applications in high-traffic scenarios, making it an ideal choice for evaluating the Obesifix application.

The test scenarios included 60, 120, and 180 virtual users to simulate different usage intensities and assess the system's scalability and response time. These tests aimed to replicate real-world conditions and observe how the application performs under varying load levels. The virtual users were configured to simulate typical usage patterns, such as logging in, registering, and engaging with the app's core features, like receiving recommendations and predictions. Figure 8 visually represents how the tests were conducted and how the system's performance was measured.

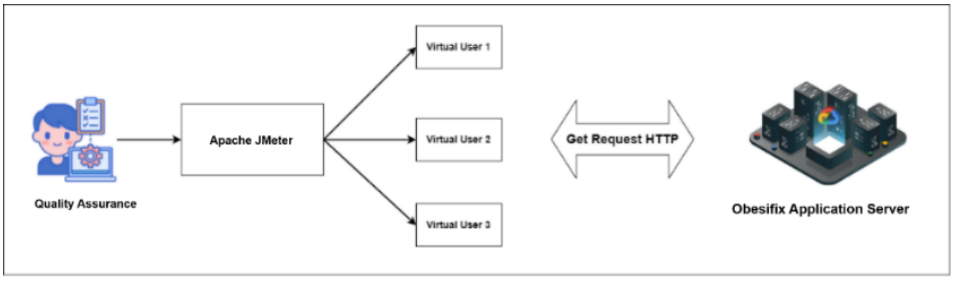

Figure 8. Illustration of performance testing

Throughput represents the total number of requests successfully processed by the system within a specified observation window and serves as an indicator of the system's processing capacity, as defined in (1). A higher throughput value indicates the system's ability to accommodate concurrent user requests efficiently. Average response time refers to the mean latency experienced per request and reflects the system's responsiveness, as formulated in (2). A lower response time value indicates faster execution and improved user experience [29]. Additionally, the success rate denotes the proportion of requests completed without errors and serves as a key measure of the system's reliability and stability under load.

$$Throughput = \frac{TotalRequest}{TotalTime} \quad (1)$$

$$AvgRespTime = \frac{TotalTime \times NumOfThread}{TotalRequest} \times 1000 \quad (2)$$

Table 3 presents the performance results for the registration endpoint running on Compute Engine. The response time increases with user load, from 720 ms at 60 users to 880 ms at 180 users, showing the effect of rising concurrency. Throughput also improves with higher load, starting at 25 rps and reaching 68 rps, indicating better resource utilization under stress. The error rate remains stable at 0%, demonstrating that registration requests are processed reliably under different concurrency levels.

Table 4 reports the performance of the login endpoint on Compute Engine, where authentication is processed for multiple users. The response time starts at 660 ms for 60 users and increases to 832 ms at 180 users, reflecting higher latency under load. Throughput improves consistently, from 27 rps to 71 rps, indicating scalability in handling authentication traffic. The error rate consistently remains 0%, indicating reliable performance during login operations.

Table 4. Result for concurrent login – Compute Engine

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	660	580	810	50	0.00%	27
120	810	700	980	65	0.00%	50
180	832	720	1020	70	0.00%	71

Table 5 summarizes the results for the recommendation endpoint on Compute Engine, which requires generating personalized dietary suggestions. The response time grows from 449 ms at 60 users to 807 ms at 180 users, showing that recommendation tasks are more computationally demanding. Throughput increases steadily from 26 rps to 69 rps, meaning that the system adapts to growing concurrency. The error rate remains at 0%, confirming that Compute Engine processes all recommendation requests without failures.

Table 5. Result for concurrent recommendation – Compute Engine

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	449	390	560	35	0.00%	26
120	690	600	850	55	0.00%	48
180	807	700	980	60	0.00%	69

Table 6 provides the evaluation of the prediction endpoint on Compute Engine, which involves executing machine learning inference. The response time is measured at 501 ms for 60 users and rises to 890 ms for 180 users, indicating a significant increase as concurrency scales. Throughput values also increase, ranging from 28 rps at 60 users to 73 rps at 180 users, reflecting that the system scales effectively. The error rate remains at 0%, showing that prediction tasks are executed correctly under all test scenarios.

Table 6. Result for concurrent prediction – Compute Engine

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	501	440	620	40	0.00%	28
120	731	640	910	60	0.00%	52
180	890	770	1090	75	0.00%	73

Table 7 presents the registration endpoint results using Cloud Run, with response times lower than those recorded on Compute Engine. The response time ranges from 350 ms at 60 users to 495 ms at 180 users, highlighting faster processing under serverless scaling. Throughput improves significantly from 31 rps

to 80 rps, which reflects the elasticity of Cloud Run in managing concurrent workloads. The error rate stays at 0%, confirming the reliability of the serverless deployment for registration requests.

Table 7. Result for concurrent register – Cloud Run

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	350	310	420	28	0.00%	31
120	415	370	500	34	0.00%	57
180	495	440	580	40	0.00%	80

Table 8 shows the login endpoint performance under Cloud Run deployment. The response times are 340 ms at 60 users and 431 ms at 180 users, which are lower than the response times reported for Compute Engine. Throughput grows from 32 rps at 60 users to 82 rps at 180 users, indicating that Cloud Run scales efficiently under load. The error rate consistently remains 0%, indicating that authentication requests are handled successfully under all test conditions.

Table 8. Result for concurrent login – Cloud Run

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	340	300	400	26	0.00%	32
120	410	360	480	32	0.00%	59
180	431	380	510	36	0.00%	82

Table 9 summarizes the outcome for the recommendation endpoint deployed on Cloud Run, which generates personalized dietary results. The response time starts at 315 ms for 60 users and increases moderately to 490 ms for 180 users, remaining faster than Compute Engine. Throughput improves considerably from 33 rps at 60 users to 83 rps at 180 users, showing strong scalability. The error rate remains at 0%, confirming that Cloud Run reliably manages concurrent recommendation requests without failures.

Table 9. Result for concurrent recommendation – Cloud Run

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	315	280	370	24	0.00%	33
120	370	330	440	28	0.00%	60
180	490	440	560	35	0.00%	83

Table 10 presents the performance results for the prediction endpoint running on Cloud Run. The response time remains relatively low, starting at 382 ms for 60 users and gradually increasing to 500 ms at 180 users, which is still significantly faster than the Compute Engine counterpart. The throughput improves consistently with load, from 33 rps at 60 users to 84 rps at 180 users, reflecting Cloud Run's ability to scale automatically with demand. The standard deviation values remain small, indicating stable performance across requests, and the error rate remains at 0%, demonstrating that predictions are handled reliably without failures at all concurrency levels.

Table 10. Result for concurrent prediction – Cloud Run

Users	Response time (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Error rate (%)	Throughput (rps)
60	382	340	440	30	0.00%	33
120	402	360	470	33	0.00%	61
180	500	450	580	38	0.00%	84

3.3. Performance results and comparison

In this performance evaluation, the Obesifix application was deployed on two different back-end architectures: a server-based deployment using GCE and a serverless deployment using GCR. The objective was to compare the two approaches in terms of responsiveness, scalability, and resource efficiency under three different load levels (60, 120, and 180 virtual users). Four critical API endpoints register, login,

recommendation, and prediction were selected because they represent the most frequently accessed and computation-intensive operations in the application.

The results reveal that Cloud Run consistently outperforms Compute Engine in most performance metrics. Across all endpoints and load levels, GCR demonstrated lower average response time, higher throughput, more stable resource utilization, and similar (zero) error rates. These findings indicate that the serverless architecture provides better elasticity, making it more suitable for workloads with fluctuating traffic, such as Obesifix.

Figure 9 compares the average response time for each endpoint under 60, 120, and 180 concurrent users. The results show that GCR achieved 20–30% lower response times than GCE across all endpoints, indicating faster request processing. The improvement was more noticeable under higher loads, where GCE exhibited gradual latency growth, whereas GCR maintained more stable performance thanks to automatic scaling.

Figure 10 presents the throughput (requests per second) achieved by both deployment models. Cloud Run consistently delivered 15–20% higher throughput compared to Compute Engine, demonstrating its ability to handle more requests in the same time frame. The performance gap widened at 180 users, suggesting that GCR scales more efficiently under peak load. At 180 users, Cloud Run reduced average response time by up to 30% and increased throughput by approximately 20% compared to Compute Engine, demonstrating better scalability under peak load.

Figure 11 shows that both GCE and GCR maintained a 0% error rate under all test conditions. This result confirms that both deployment approaches were able to handle concurrent traffic without failed requests, which is critical for ensuring reliability in a production health application.

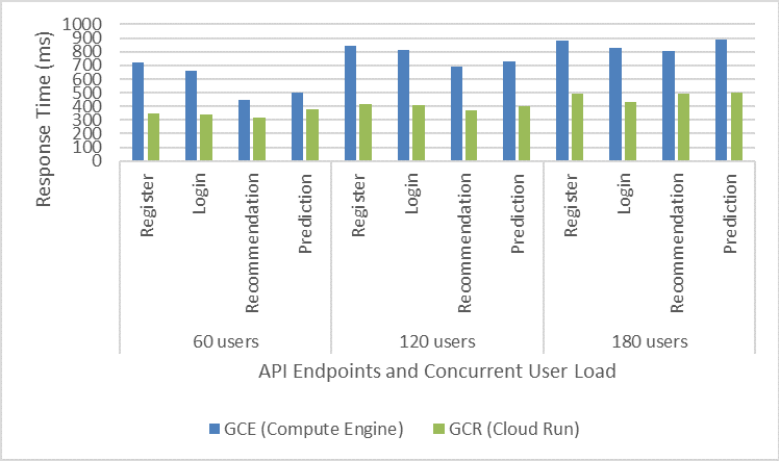

Figure 9. Response time comparison between GCE and GCR

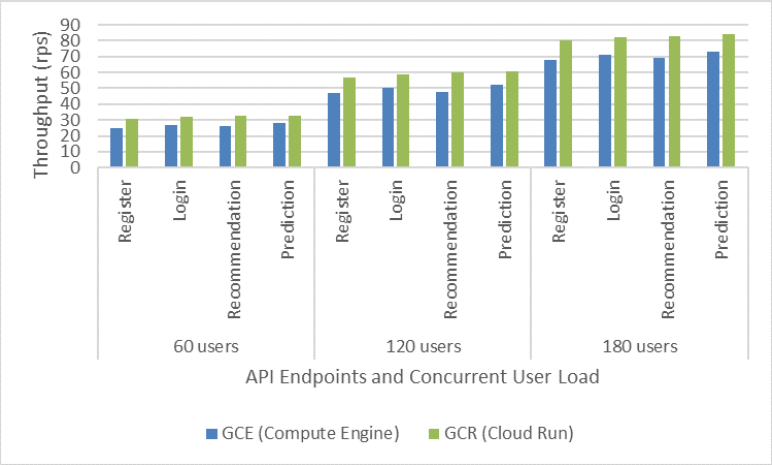

Figure 10. Throughput comparison between GCE and GCR

Figure 12 illustrates memory consumption across all endpoints and load levels. GCE consumed 10–15% more memory than GCR on average, indicating less efficient resource usage in the server-based deployment. In contrast, GCR utilized memory more dynamically, which is advantageous for cost optimization and sustainable operations.

Figure 13 compares CPU utilization. GCE consistently reported higher CPU usage, especially at 120 and 180 users, which could lead to resource saturation. GCR maintained lower and more balanced CPU utilization, confirming that its auto-scaling capability effectively distributes the workload and prevents CPU bottlenecks.

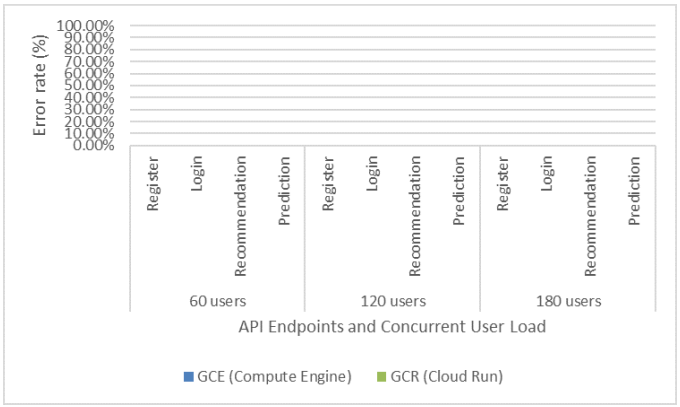

Figure 11. Error rate comparison between GCE and GCR

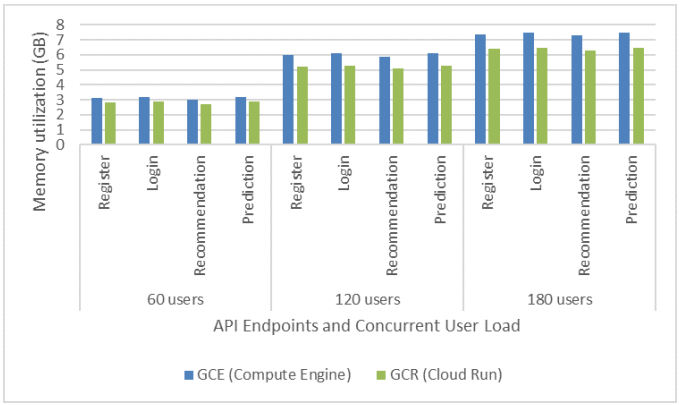

Figure 12. Memory utilization comparison between GCE and GCR

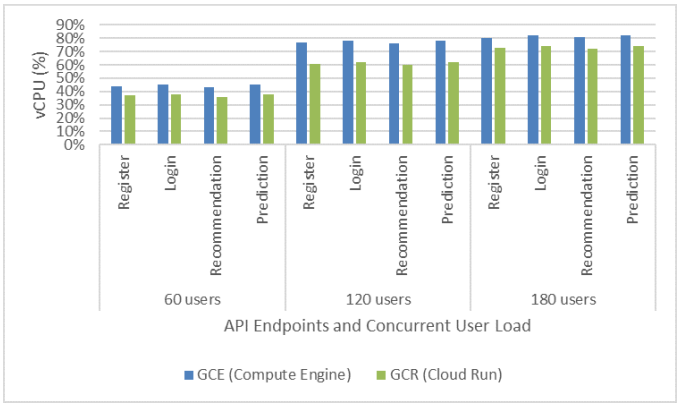

Figure 13. vCPU Comparison between GCE and GCR

Overall, the comparison demonstrates that the serverless architecture (GCR) is better suited for the Obesifix application, particularly under variable or high-load scenarios. Its lower latency, higher throughput, and more efficient resource usage make it a cost-effective and scalable solution, ensuring a smoother user experience and reduced operational overhead compared to a traditional server-based deployment. These findings validate that a serverless architecture is a practical choice for mobile health applications requiring real-time responsiveness and cost-effective scalability, providing a strong foundation for future iterations of Obesifix.

3.4. Time to first byte

TTFB was evaluated as an indicator of server responsiveness, defined as the elapsed time between a client's request and the receipt of the first byte of the response. TTFB refers to the time it takes from the moment a client sends a request to the server until the client receives the first byte of data. This metric is crucial for understanding the server's responsiveness, especially in dynamic applications like Obesifix, where real-time data retrieval is essential.

According to Google PageSpeed Insights, a good TTFB score is typically below 800 ms, which indicates that the server is responding quickly. Scores between 800 ms and 1800 ms need improvement, while anything above 1800 ms is generally considered poor, potentially leading to slower user experiences. TTFB was measured for all key endpoints register, login, recommendation, and prediction under the same traffic levels used in previous tests (60, 120, and 180 virtual users). The results showed that the Obesifix application maintains a good TTFB (< 800 ms) across all scenarios. Furthermore, the serverless deployment (GCR) consistently achieved slightly lower TTFB values compared to the server-based deployment (GCE), particularly under higher loads (120 and 180 users). This finding suggests that Cloud Run's automatic scaling helps maintain low initial latency even during peak traffic, which is critical for ensuring a smooth user experience in mobile health applications. These TTFB results are consistent with the lower response times observed in Figure 14, confirming that GCR consistently delivers faster initial server responses even as traffic increases.

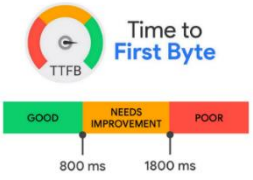

Figure 14. TTFB [30]

3.5. User interface design of Obesifix app

The design of the Obesifix application interface was developed using custom resources, including logos and user-friendly design elements. The design focuses on simplicity to improve accessibility and ease of use for users. With a clean and intuitive layout, the app ensures that users can easily interact with its features, such as personalized food recommendations and caloric tracking. The user-friendly design minimizes complexity, making it easier for both new and experienced users to navigate the application. Additionally, the interface is responsive and adaptable, providing smooth performance across various devices. This approach ultimately enhances the user experience, making it easier for individuals to manage their nutrition and maintain a healthy lifestyle, as shown in Figure 15.

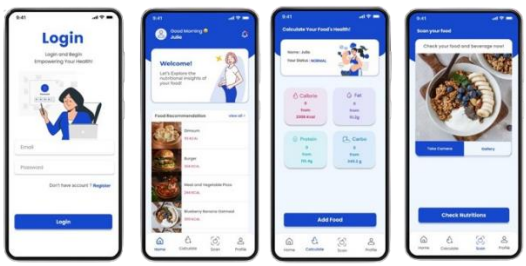

Figure 15. User interface design of Obesifix

4. CONCLUSION

Cloud computing-based APIs offer substantial advantages in designing and implementing modern software applications. Integrating cloud services enables improved scalability, flexibility, and performance while eliminating the complexities associated with traditional infrastructure management. The Obesifix application, as described in this study, demonstrates how cloud-based solutions can enhance backend operations and maintain seamless functionality across varying user loads.

Implementing PaaS tools such as Google Cloud Run allows applications to scale dynamically in response to demand, ensuring high availability and minimizing operational overhead. Performance testing results emphasize the importance of optimizing cloud resources for responsiveness and efficiency under varying traffic intensities. Cloud-based APIs facilitate smoother integration with external systems, better resource allocation, and faster development cycles. Building on these findings, future research should explore the evaluation of asynchronous and synchronous programming paradigms within serverless environments to improve request-handling mechanisms in mobile backend systems. Additional experiments should investigate autoscaling thresholds, cold-start latency mitigation techniques, and cost-performance trade-offs under real-world workloads. Furthermore, future studies may compare multi-cloud and hybrid-cloud deployments to assess portability and fault tolerance, and extend this evaluation to other mobile health applications beyond Obesifix to validate the generalizability of the proposed approach.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Politeknik Elektronika Negeri Surabaya (PENS) for the supportive academic environment, laboratory facilities, and administrative assistance that enabled the completion of this research. The authors also thank the faculty members and colleagues at PENS for their guidance, technical discussions, and thoughtful suggestions that substantially improved the quality of this work.

FUNDING INFORMATION

The Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Maulana Bintang Irfansyah	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
Bilal Waheed					✓	✓	✓	✓		✓	✓			
Idris Winarno	✓	✓		✓	✓	✓				✓		✓		
Akhmad Alimudin	✓	✓		✓	✓	✓				✓		✓	✓	✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The publication of this paper is not associated with any potential conflicts of interest, as stated by the authors. The manuscript did not contain any instances of plagiarism, as confirmed by the authors.

INFORMED CONSENT

We have obtained informed consent from all individuals included in this study. All participants were provided detailed information about the study's purpose, procedures, and potential risks. Written consent was obtained from each participant before their inclusion in the study, ensuring full compliance with ethical standards and privacy protection.

DATA AVAILABILITY

The datasets and model artifacts used in this study are proprietary and cannot be shared publicly due to privacy restrictions. However, they can be made available upon reasonable request by contacting the corresponding author. The source code used for the study is available upon request for privacy and confidentiality reasons.

REFERENCES

[1] A. Bin Rashid and M. A. K. Kausik, "AI revolutionizing industries worldwide: a comprehensive overview of its diverse applications," *Hybrid Advances*, vol. 7, p. 100277, Dec. 2024, doi: 10.1016/j.hybadv.2024.100277.

[2] T. Wang, A. Seiger, A. Markowetz, I. Andone, K. Błaszkiewicz, and T. Penzel, "Smartphone usage patterns and sleep behavior in demographic groups: retrospective observational study," *Journal of Medical Internet Research*, vol. 27, p. e60423, Jul. 2025, doi: 10.2196/60423.

[3] J. Howarth, "How many people own smartphones? (2025-2029)," Exploding Topics. Accessed: May 20, 2025. [Online]. Available: https//explodingtopics.com/blog/smartphone-stats

[4] P. Ashokan and A. Golli, "Scalable backend solutions for real-time machine learning applications in web and mobile platforms," *Sarcouncil Journal of Applied Sciences*, vol. 4, no. 9, pp. 8–14, 2024.

[5] R. Shah, S. Jagtap, and V. Jain, "Architecting analytics-driven mobile ecosystems: scalable backend frameworks for intelligent data flow and real-time user insights," *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 6, no. 2, pp. 83–91, 2025, doi: 10.63282/3050-9262.IJAIDSML-V6I2P109.

[6] P. Okanda, A. Chhatbar, and O. Njeru, "DbAPI: a backend-as-a-service platform for rapid deployment of cloud services," in *2024 IST-Africa Conference (IST-Africa)*, IEEE, May 2024, pp. 1–12, doi: 10.23919/IST-Africa63983.2024.10569490.

[7] A. P. Rajan, "A review on serverless architectures - function as a service (FaaS) in cloud computing," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 18, no. 1, pp. 530–537, Feb. 2020, doi: 10.12928/telkomnika.v18i1.12169.

[8] E. Dritsas and M. Trigka, "A survey on the applications of cloud computing in the industrial internet of things," *Big Data and Cognitive Computing*, vol. 9, no. 2, p. 44, 2025, doi: 10.3390/bdcc9020044.

[9] A. Alimudin and R. W. Sudibyo, "Rescheduling strategy for container orchestration system to improve application availability," *Inform : Jurnal Ilmiah Bidang Teknologi Informasi dan Komunikasi*, vol. 8, no. 2, pp. 137–146, Jun. 2023, doi: 10.25139/inform.v8i2.6220.

[10] M. T. Amron, R. Ibrahim, and N. A. A. Bakar, "Cloud computing acceptance among public sector employees," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 19, no. 1, pp. 124–133, Feb. 2021, doi: 10.12928/telkomnika.v19i1.17883.

[11] D. Darwish, *Emerging trends in cloud computing analytics, scalability, and service models*. in Advances in Computer and Electrical Engineering. IGI Global, 2024, doi: 10.4018/979-8-3693-0900-1.

[12] Janet Julia Ang'udi, "Security challenges in cloud computing: a comprehensive analysis," *World Journal of Advanced Engineering Technology and Sciences*, vol. 10, no. 2, pp. 155–181, Dec. 2023, doi: 10.30574/wjaets.2023.10.2.0304.

[13] H. I. H. Tandri, H. H. Nuha, and R. G. Utomo, "Cloud computing-based API design and implementation for hening mobile application," in *2023 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, IEEE, Nov. 2023, pp. 341–346, doi: 10.1109/COMNETSAT59769.2023.10420654.

[14] H. J. H. Sulistiyo, H. H. Nuha, and R. G. Utomo, "Design and implementation of cloud computing-based API for mobile application Tookar," in *2023 3rd International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA)*, IEEE, Dec. 2023, pp. 490–495, doi: 10.1109/ICICyTA60173.2023.10428947.

[15] M. Firdaus, N. Alamsyah, and A. H. Jatmika, "Development of a REST API for the Rinjani visitor application using extreme programming," in *2024 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, IEEE, Jul. 2024, pp. 518–523, doi: 10.1109/ISITIA63062.2024.10667823.

[16] M. A. H. Akbar, D. Fahrizal, J. Kustija, and I. Surya, "Digital technology integration in TVET for tourism: a case study for an Android-based application development and implementation," in *2024 9th International STEM Education Conference (iSTEM-Ed)*, IEEE, Jul. 2024, pp. 1–6, doi: 10.1109/iSTEM-Ed62750.2024.10663108.

[17] S. Athreya, S. Kurian, A. Dange, and S. Bhatsangave, "Implementation of serverless e-commerce mobile application," in *2022 2nd International Conference on Intelligent Technologies (CONIT)*, IEEE, Jun. 2022, pp. 1–5, doi: 10.1109/CONIT55038.2022.9847829.

[18] V. Moysiadis, K. Tsakos, P. Sarigiannidis, E. G. M. Petrakis, A. D. Boursianis, and S. K. Goudos, "A cloud computing web-based application for smart farming based on microservices architecture," in *2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, IEEE, Jun. 2022, pp. 1–5, doi: 10.1109/MOCAST54814.2022.9837727.

[19] K. Azkiya, M. Irsan, and M. F. Fathoni, "Implementation of App Engine and Cloud Storage as REST API on smart farm application," *Sinkron*, vol. 8, no. 2, pp. 902–910, Mar. 2024, doi: 10.33395/sinkron.v8i2.13386.

[20] J. Jordanov, D. Simeonidis, and P. Petrov, "Containerized microservices for mobile applications deployed on cloud systems," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 18, no. 10, pp. 48–58, May 2024, doi: 10.3991/ijim.v18i10.45929.

[21] M. Luchkevych, V. Luchkevych, and I. Shakleina, "Mobile DevOps in education: practical training through application development," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 19, no. 15, pp. 129–141, Aug. 2025, doi: 10.3991/ijim.v19i15.55531.

[22] A. Wali, H. Almagrabi, S. El-Feky, and M. Jokhdar, "Dawwen: an Arabic mental health mobile app based on natural language processing," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 19, no. 04, pp. 108–131, Feb. 2025, doi: 10.3991/ijim.v19i04.51999.

[23] Y. Jo, Y. Jang, and J. Paik, "Design and implementation of a service platform that recommends the optimal shortest distance as a patrol route," *Journal of Korean Society for Internet Information*, vol. 23, no. 1, pp. 1–9, 2022.

[24] J. Nupponen and D. Taibi, "Serverless: what it is, what to do and what not to do," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, IEEE, Mar. 2020, pp. 49–50, doi: 10.1109/ICSA-C50368.2020.00016.

[25] B. B. Rodrigues, *Google Cloud Digital Leader Certification Guide: A Comprehensive Study Guide to Google Cloud Concepts and Technologies*. Birmingham, UK: Packt Publishing, 2024.

[26] M. B. Irfansyah, B. Waheed, I. Winarno, and A. Alimudin, "Implementation of Scrum framework in modern software

development projects," *2025 International Electronics Symposium, IES 2025*, pp. 875–880, 2025, doi: 10.1109/IES67184.2025.11161121.
[27] C. S. Kondaveeti1, H. Jodhavat, and V. Gogineni, "Developing scalable web applications with Java and J2EE in cloud environments," *Sarcouncil Journal of Engineering and Computer Sciences*, vol. 4, no. 1, pp. 1–8, 2025.
[28] I. Indrianto, "Performance testing on web information system using Apache JMeter and BlazeMeter," *Jurnal Ilmiah Ilmu Terapan Universitas Jambi*, vol. 7, no. 2, pp. 138–149, Dec. 2023, doi: 10.22437/jiituj.v7i2.28440.
[29] D. J. Arrizki, S. A. Kosim, and U. L. Yuhana, "A comparative performance analysis and cost efficiency between AWS and GCP services in cloud-based software development," in *2024 2nd International Conference on Software Engineering and Information Technology (ICoSEIT)*, IEEE, Feb. 2024, pp. 149–154, doi: 10.1109/ICoSEIT60086.2024.10497497.
[30] J. Wagner and B. Pollard, "Time to first byte (TTFB)," webdev. Accessed: May 27, 2025. [Online]. Available: https//web.dev/articles/ttfb

BIOGRAPHIES OF AUTHORS

Maulana Bintang Irfansyah is a Master of Applied Computer student in Informatics and Computer Engineering at Politeknik Elektronika Negeri Surabaya, Indonesia. He completed his Bachelor of Applied Computer in Informatics Engineering from Politeknik Negeri Malang in August 2024. His research interests focus on cloud computing, software development, and quality assurance. He can be contacted at email: maulana.bintang.irfansyah@gmail.com.

Bilal Waheed is a Master of Applied Computer student in Informatics and Computer Engineering at Politeknik Elektronika Negeri Surabaya, Indonesia. He completed his Bachelor of Electrical Engineering (Electronics) from Federal Urdu University of Arts, Sciences and Technology, Islamabad Pakistan. His research interests focus on network security, internet of things, electronics, cloud computing, and machine learning. He can be contacted at email: bilal677@pasca.student.pens.ac.id.

Idris Winarno received the B.Eng. degree in Information Technology from Politeknik Elektronika Negeri Surabaya (PENS), Indonesia, in 2005, the M.S. degree in Computer Science from Sepuluh Nopember Institute of Technology, Indonesia, in 2008, and the Dr.Eng. degree in Computer Science from Toyohashi University of Technology, Japan, in 2018. He joined the Department of Computer Science, PENS, as a Junior Lecturer, in 2008. His research interests include computer networks, network security, and resilient computing. He can be contacted at email: idris@pens.ac.id.

Akhmad Alimudin received the B.S. degree in computer science and the M.S. degree from the Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia, in 2010 and 2013, respectively. He completed his Ph.D. in Computer Science at Toyohashi University of Technology, Japan. He joined Politeknik Elektronika Negeri Surabaya (PENS) as Lecturer in 2014. His research interests include intelligent systems, machine learning, and computer network applications. He can be contacted at email: alioke@pens.ac.id.