BIS INFORMATION TECHNOLOGY AND COMPUTER SCIENCE

Volume 3 2026 | DOI: 10.31603/bistycs.544
7th Borobudur International Symposium on Science and Technology (BIS-STE) 2025

UNIMMA PRESS
Universitas Muhammadiyah Magelang

A queue layer approach to enhancing the performance of CRM message broadcasting

Wening Raharjo Pura Wilantara[1], Setiya Nugroho[1*], Purwono Hendradi[1], Nuryanto[1], and Andi Widiyanto[1]

[1] Muhammadiyah University of Magelang, Magelang, Indonesia
[*]Corresponding author email: setiya@unimma.ac.id

Abstract

In the digital era, Customer Relationship Management (CRM) is a key strategy to improve long-term relationships with customers. There is an application with a microservices-based CRM platform. One important feature at CRM is broadcasting messages to many customers simultaneously. This CRM application has a large number of customers. There is a problem in this CRM application, namely the overload on the hardware in processing message delivery. The purpose of this research is to analyze the impact of queue layer implementation on message delivery performance. This research proposes the implementation of a queue layer on the message broadcast feature in the CRM application. The queue layer allows message delivery to be organized in stages, thereby reducing the risk of bottlenecks, and ensuring more efficient message delivery. The method used in this research is Rapid Application Development. The results showed that the implementation of the queue layer can provide a technical solution by reducing the use of RAM and CPU resources in the CRM system. The queue layer implementation in the CRM system results in a decrease in CPU and RAM usage.

Keywords

Message broadcast, Queue layer, Microservices, Performance optimization, CRM, Resource utilization

Published:
May 04, 2026

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Selection and Peer-review under the responsibility of the 7th BIS-STE 2025 Committee

Introduction

In today's digital era, the ability to effectively manage customer relationships is one of the key factors for business success [1]. Customer Relationship Management (CRM) is a business strategy that aims to build long-term customer relationships and increase company profits through customer data management [2].

One important feature of CRM is the ability to broadcast messages, which allows companies to deliver important information to all customers simultaneously [3]. However, broadcasting messages can be cumbersome in scenarios with many customers and potentially cause performance issues [4]. Simultaneously sending messages to thousands or even millions of subscribers can cause overload on third

parties, such as messaging service providers or cloud infrastructure, leading to delays or even message delivery failures [5]. Therefore, a solution is needed to manage the messaging process to keep it efficient and reliable [6].

One approach that can be used to overcome this problem is to implement a queue layer in the message broadcast feature. The implementation of queues shows that this approach improves the efficiency of message delivery and reduces the risk of bottlenecks in third-party infrastructure [7]. With the queueing layer, messages can be organized in a queue and sent gradually according to the existing infrastructure's capacity and the third party's capacity [8]. This helps maintain the performance of the message delivery service and ensures that each message is delivered successfully without interruption [9].

This research will conduct a case study on a CRM application, a platform that provides CRM services to its users. CRM applications' main focus is improving customer experience and supporting digital marketing strategies. The CRM application is a system with a microservices architecture. Queue is one of the important components in microservices architecture [10], [11]. This makes the queue a determinant of the scalability and performance of other services from CRM applications [12].

This research has several limitations that need to be considered. The Queue Layer implementation only focuses on managing message queues for broadcast features in the CRM system without covering other features outside of queue management [13]. The technology used is limited to Golang, Redis, Docker, Apidog, and Asynq Library; alternative approaches are not discussed [14]. Performance testing is done in limited simulations with certain messages and user scenarios without covering large scales or real production environments. Performance evaluation only considers message delivery latency and throughput, while other quality of service aspects, such as reliability and security, are beyond the scope of this research.

This research will study the effect of the queueing layer's implementation on the message delivery performance in the CRM application. This research aims to implement the queue layer on the message broadcast feature in the CRM application and analyze the effect on the performance of the message delivery service. This research has provided benefits by providing technical solutions to improve the efficiency and reliability of the message broadcast feature in CRM applications.

Method

The design of the system to overcome the problem will follow a roadmap that applies the Rapid Application Development (RAD) method as shown in Figure 1. This research methods is an incremental engineering software development process model that emphasizes short and fast development cycles [15]. This method starts with analysis and quick design, prototype cycles, testing, and implementation. The prototype cycle has repetitions, namely, build, demonstrate, and refine.

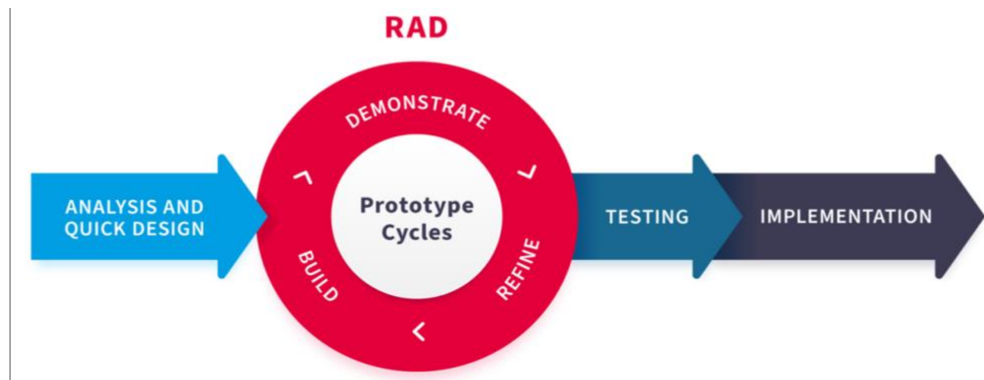

Figure 1. RAD method roadmap

1. Research procedure

   The research procedure begins with a literature study to understand the concept of queue layer and relevant message management technologies. After that, the CRM system is analyzed to identify bottleneck points in the message broadcast feature, which is the basis for designing the queue layer architecture. Implementation is carried out with technology that supports queue management as needed, then tested through large message delivery simulations using performance metrics such as latency, CPU, and RAM. The test results are then analyzed to assess the effectiveness of the queue layer in improving system efficiency. Furthermore, the research is compiled into a report that presents the entire process, results, and recommendations for further system development.

2. Materials used

   The materials used in this study consist of hardware and software. The hardware used is two VPS. The first VPS uses the Ubuntu 22.04 (LTS) x64 operating system, 16 GB of RAM, and an 80 GB hard disk. The second VPS uses the Ubuntu 22.04 (LTS) x64 operating system, 8 GB of RAM, and a 120 GB hard disk. The Golang programming language, Redis database, Docker, Apidog, and Asynq Library are the software used.

Analysis and quick design

Broadcast messages will be routed according to the specified message type to various API endpoints in the chat plugin service. Each plugin will have a token with a string data type for plugin identification that will be routed to the list of existing API endpoints.

The team members consist of Admin and Customer Support, who communicate with customers using the chat plugin. The team members have customer data in the form of names and contactable phone numbers. When CRM sends a message, the HTTP protocol will first be directed to the queue layer [16]. The message request will be processed into the queue and then directed individually to the CS plugin to be forwarded to the customer as shown in Figure 2.

The other company's team members use the same scheme, which allows CRM to make multiple HTTP requests to the chat plugin service. When the chat plugin receives too many HTTP requests, it experiences downtime, resulting in failure to deliver messages.

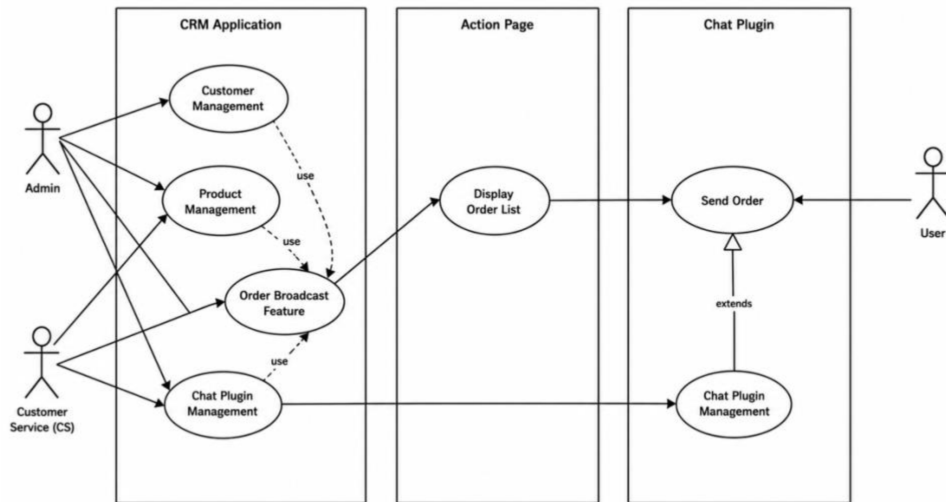

Figure 2. Broadcast message feature use case

Prototype cycles

Prototype Cycles start with building, demonstrating, and refining. In the build stage, the CRM application has an input form for setting broadcast messages, as shown in Figure 3. In this form, the user needs to enter some data: broadcast message name, which is the name for identification in the system; message format, which is the message that will be delivered to customers. The Selected Chat plugin will be used to send messages to customers. The customer data source will be used as the target for sending messages. Users can select customer data directly or customer data that has been grouped using Label or Segmentation. The Broadcast Settings will be where the user can set the date and time the broadcast message will be sent. Admin can allocate plugins, as shown in Figure 3. Customer Service can also make broadcast settings such as message templates, image attachments, send time settings and others. The message broadcast setting data will be stored in the database, which will then be triggered on the date and time that was set when setting the message. When triggered, this message broadcast feature will make a request to the chat plugin service using the HTTP protocol.

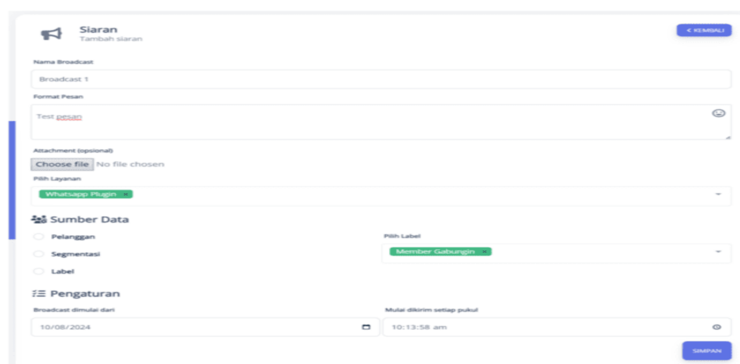

Figure 3. Broadcast Input form

At the demonstrating stage, when a request with HTTP protocol appears for the text message endpoint, the payload data will be parsed to retrieve the plugin token data, in this example, tokenplugin1. A Go programming language struct will be created based on

the payload, which becomes a Task. The plugin token will be the identifier for the task queue grouping. A check will be made to see if a worker for the plugin token already exists. If it does not exist, the worker for the token identifier will be added to the system, and the worker server will be restarted to process the queue of tasks. The example will be applied to HTTP requests with payloads containing other plugin tokens. Tokens as identifiers will be grouped [17].

At the refinement stage, the queue will start when the service queue layer receives an HTTP request from the CRM service [18]. Each chat plugin will have an identifier in the form of a token for mapping and a queue name. This design utilizes a library in the Go programming language called Asynq to process queues asynchronously. Async has two terms for handling queue data, namely Task and Worker. Task is a term for a struct in the Go programming language that becomes an enqueue object. Worker is a process in the system that processes queue data. The worker in the async library is a server connected to Redis to store queue status data [19].

Testing

System testing is carried out to ensure that the system design made is in accordance with the system design that has been previously built. This system test uses the Black Box Testing method to test software without having to pay attention to software details [20].

1. Functionality testing

   There are eight test points that are used as a reference for the successful implementation of the service queue layer in the CRM application. Functionality testing on the service queue layer in this case focuses on the resulting output. The test scenario contains test cases. Each test case will be given an ID in the form of a letter for test identification. The test results will explain whether the system developed is as expected or not.

   The first test is testing the server's liveliness with the Test ID code A. The system is able to receive HTTP requests through the port that has been prepared. This system uses a Golang programming language library called Fiber. The system can receive HTTP requests on port 8000. One of the protocol routes used is the route for the dashboard monitoring task and queue named asynqmon which can be accessed via the /monitoring route. For system development in a local environment, namely localhost.

   Test the text message API with test case code B on the service queue layer using payload. Testing will use a mock server on Apidog with JSON format [21]. API requests send messages with the HTTP POST method will generate a response. The test results show that this service is able to provide an HTTP response with a status of 200 or success.

   Testing sending image messages on test ID C using Apidog and mock server with payload. The payload will be sent using the POST method and JSON format. Test

requests on the API send image messages will result in a successful status. This indicates that the API for sending image messages is successfully running on the service queue layer [22].

Table 1. Black box testing results

ID Test	Test Step	Expected result	Test result
A	Access the server by opening the monitoring dashboard	The server can provide access for HTTP requests	The monitoring dashboard is successfully accessed
B	Send the send message payload on the send text message API	Server can respond without error messages	The payload was successfully sent and generated a status code of 200
C	Send the send message payload on the send image message API	Server can respond without error messages	The payload was successfully sent and generated a status code of 200
D	Send the send message payload on the send text message API	Server can process payload into tasks for enqueue	The payload was successfully converted into a task and enqueued.
E	Message clustering test	Server can process payload and process tasks based on tokens.	Payload successfully converted into tasks and grouped into queues
F	Message Processing Testing	Task can be processed and dequeue with success status.	The task in the queue runs and has a success status
G	Message delivery test	Service can interact with the chat plugin service and successfully send messages,	Message can be sent
H	Scalability & Portability Testing	Service can run in any environment (portable) and scalable.	Services can run in different environments and can be duplicated for scalability.

Test case D is a test to ensure that sending HTTP messages can be processed by the system and converted into a queue. The test results for enqueue show that an HTTP request with the token plugin tokentest can be processed by the worker and appears on the monitoring dashboard.

Message grouping in test case E is a grouping of messages that have been converted into queues based on the token from the plugin. In testing the message data can be grouped into queues according to the name of the chat plugin token.

Testing in test case F will use a script for testing with some data. The scenario of this test is to send the payload to the API send messages in a row using repetition or looping without pause. Testing using a testing script shows that messages can be grouped according to the plugin name as well as all queues can be processed to be sent to the mock server service plugin chat.

In Test Case G, this test will be carried out by sending a request to the send message API using Apidog with the chat plugin service active at the URL http://localhost:3000. Requests will use a payload and be sent to the text message API in the service queue layer. The message will then be processed by the chat plugin and will send a message to a chat server such as WhatsApp to the recipient.

Scalability and portability testing the source code of the service queue layer can be built using docker commands to create a portable system image [23]. The docker image can then be used on any host system as long as the system supports the docker engine. The results of functionality testing can be seen in Table 1.

2. Performance testing

Performance testing uses computer devices with Windows Operating System specifications, AMD Ryzen 5 2.1 GHz CPU, 8 GB RAM, Grafana and Prometheus monitoring applications.

Performance testing is done using HTTP request loops with three criteria The first criterion is HTTP Requests on the API send text messages service Plugin Chat for testing without Queue Layer and API send text messages through Queue Layer. The second criterion is that HTTP requests are made with a total of 10,000 requests, using 10 workers with 10 connections running together. The third criterion is that requests are made using only one client token on the Plugin Chat service.

Test results on service plugin chat without queue layer show an increase in RAM usage. RAM usage reached a total of 2.5 GB. When implementing the queue layer, RAM usage increases gradually according to the queue through the service queue layer. The Chat Plugin service RAM usage shows a total of 2.3 GB as shown in Figure 4.

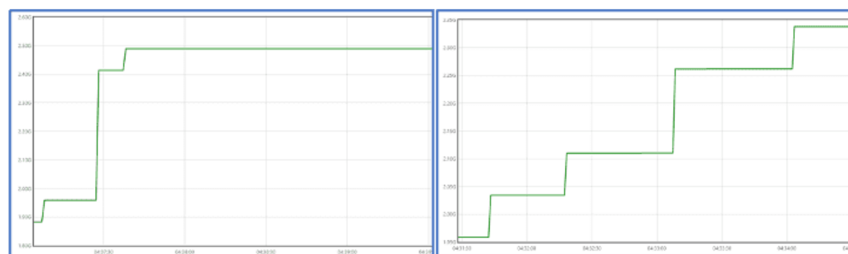

Figure 4. RAM Benchmark results

CPU usage without a service queue layer also experiences similar problems to RAM usage. The amount of usage on the CPU without the service queue layer increased significantly during performance testing. The increase in CPU usage is up to 72.5% as shown in Figure 5.

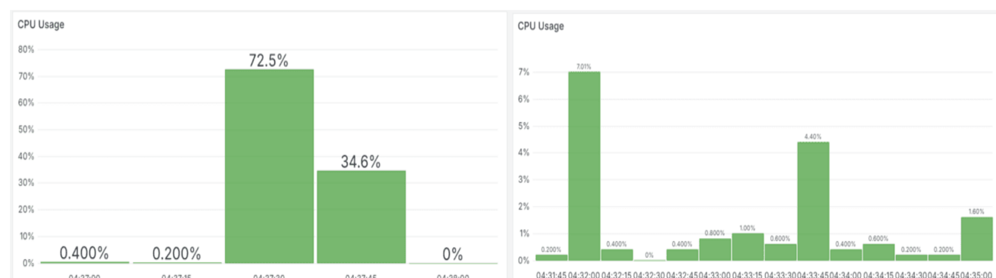

Figure 5. CPU Benchmark results

The results of testing using the black box method, performance testing and implementation show the success and achievement of the objectives of creating this service. The decrease in the use of RAM resources by 8% and CPU up to 90% on one

token is in accordance with the objectives of this study. The test results are summarized in Table 2.

Table 2. Black box testing summarized results

Criteria (10.000 request)	Without Queue Layer	With Queue Layer	Efficiency
CPU Usage	Highest 72.5%	Highest 7%	Decreased Usage >90%
RAM Usage	1.9 GB – 2.5 GB	1.9 GB – 2.3 GB	Decreased Usage 200MB or 8%
Message failed	25%	0%	100%

Implementation

Implementation of queue layer architecture as shown in Figure 6 using VPS with Ubuntu operating system specifications, 2 GB RAM, 25 GB Disk Space. Queue layer in this service will apply docker container technique with Service queue-layer and Service redis image design. In the Service queue-layer image used Image golang: bullseye, Forward Port 8080: 8000, and Network queue-network. In Service redis used Image bitnami/redis, Expose : 6379, Network queue-network [24].

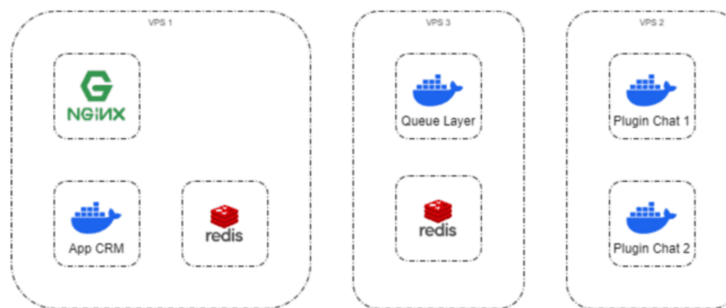

Figure 6. Queue layer architecture

Implementation was carried out on August 22, 2024 by redirecting direct message delivery to the chat plugin service to be changed to pass through the queue layer to be forwarded to the chat plugin service. The data obtained is data taken from the statistics of the Virtual Private Server (VPS) where the chat plugin service is running.

The impact of the implementation can be seen through the resource graph of the VPS from the observed chat plugin service. CPU and RAM monitoring is one of the easiest things to observe on the VPS dashboard. The implementation of this service resulted in a decrease in the spike in resource usage on the CPU and Memory as shown in Figure 7 and Figure 8.

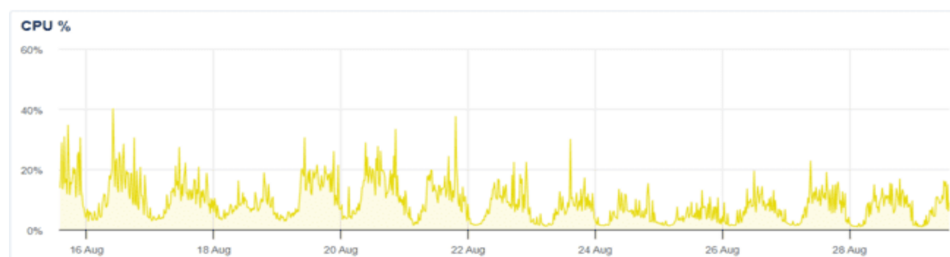

Figure 7. CPU usage onVPS

Figure 8. Memory usage on VPS

Results and Discussion

The results of this study show that the implementation of a queue layer on the CRM message broadcast feature is more efficient than traditional methods. CPU usage is reduced by more than 90% and RAM is reduced by up to 200 MB, reducing the risk of overload and message delivery failures during high volumes. With gradual message delivery, the queue layer ensures more stable and optimal performance on large-scale CRM compared to methods without queue management. Additional data analysis and a clearer presentation of the findings can improve the understanding of the benefits of this queue layer in ensuring more stable and optimal CRM performance at scale.

The implementation of queue layer in CRM message broadcast feature has significant impact in increasing the efficiency and reliability of message delivery in large scale. With message queue, the system can manage traffic load gradually, thus preventing overload that can cause delay or failure of delivery. This provides a more scalable solution for companies that continue to grow, because the system can still support the increasing message volume without requiring significant additional infrastructure. In practical terms, this ensures that messages, such as notifications or promotional information, are still delivered on time to customers even in high volume conditions.

From a managerial perspective, implementing a queue layer in CRM helps improve customer satisfaction through consistent and timely messaging, strengthening customer relationships, and encouraging loyalty. Better message traffic management also reduces the need for manual intervention from the IT team, freeing up time and operational costs for other, more strategic initiatives. In addition, the performance data generated from this queue system can be used for better data-based decision-making regarding development features, resource allocation, and effective communication strategies with customers.

Conclusion

This study has several limitations that need to be considered. The implementation of the Queue Layer only focuses on managing the message queue. The technology used is limited to Golang, Redis, Docker, Apidog, and Asynq Library. Performance testing is done in limited simulations with certain messages and user scenarios. Testing to measure the efficiency of CPU and RAM usage.

This research successfully implements the queue layer in the message broadcast feature of the CRM system. The results show that the use of queue layer can manage message flow more effectively, reduce the risk of overload, and ensure timely message delivery. The implementation of the queue layer has a positive impact on the performance of the message delivery service in the CRM application. The use of this layer allows a more even distribution of workload, thereby reducing system response time and minimizing delivery failures due to server overload. Some of the obstacles that arise during the implementation of the queue layer include the complexity of the system configuration, the need for infrastructure adjustments, and the potential for bottlenecks in message processing. The proposed solutions include system configuration optimization and regular monitoring. The result of the queue layer implementation in the CRM system is a decrease in CPU usage by more than 90% and a decrease in RAM usage by 200 MB.

References

1. Alawiyah I and Humairoh P 2017 the Impact of Customer Relationship Management on Company Performance in Three Segments Jurnal Ilmiah Ekonomi Bisnis 22
2. Al kharraz F and Seçim H 2023 Customer relationship management impact on customers' trust in the Palestinian telecommunications company Paltel during the Covid-19 era Electronic Commerce Research
3. Palupi E S 2022 Web-Based Customer Services Management Implementation For The Sales Division Jurnal Riset Informatika 5
4. Al-Twajre B A 2019 Performance Analysis of Messages Queue in the Different Actor System Implementation 2019 11th International Scientific and Practical Conference on Electronics and Information Technologies, ELIT 2019 - Proceedings
5. Menaouer B, Khalissa S, El Amine Belayachi M and Amine B 2021 The role of drop shipping in e-commerce: The Algerian case International Journal of e-Business Research 17
6. Van N N 2019 The cross-layer message-broadcasting mechanism for active road safety application based on dynamic priority Journal of Communications 14
7. Kristanto A A, Harjoseputro Y, Samodra J E and others 2020 Golang and New Simple Queue Implementation on Third Party Sandbox System Based on REST API Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi) 4
8. Menaouer B, Mohammed S and Nada M 2022 The Impact of Business Intelligence and Knowledge Management on Sustainability Performance in the Tourism Industry in Algeria Indonesian Journal of Sustainability Accounting and Management 6
9. Mostefaoui A, Merzoug M A, Haroun A, Nassar A and Dessables F 2022 Big data architecture for connected vehicles: Feedback and application examples from an automotive group Future Generation Computer Systems 134
10. Rintamäki V 2022 Designing message queue service in microservice architecture Master of Science Thesis (Tampere: Tampere University)
11. Tu Z 2023 Research on the Application of Layered Architecture in Computer Software Development Journal of Computing and Electronic Information Management 11
12. Park G, Jeon B and Lee G M 2023 QoS Implementation with Triple-Metric-Based Active Queue Management for Military Networks Electronics (Switzerland) 12
13. Joshi V, Shirur R and Shukla P 2023 Implementation of Multi-Channel Message Queues using QUIC 7th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2023 - Proceedings
14. Ardiansyah H and Fatwanto A 2022 Comparison of Memory usage between REST API in Javascript and Golang MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer 22
15. Nalendra A K 2021 Rapid Application Development (RAD) model method for creating an agricultural irrigation system based on internet of things IOP Conf Ser Mater Sci Eng 1098
16. Mujio Mukmin T, Rodhiah R, Wasino W, Michella Wijaya S, Febiyani Metty P, Lim C and Della Safitri R 2021 Marketing Strategy Based on CRM (Customer Relationship Management) at PT. Great Food

Prosperity in Tangerang City International Journal of Social Science Research and Review 4

17. Santos J, Verkerken M, Dhooge L, Wauters T, Volckaert B and De Turck F 2023 Performance Impact of Queue Sorting in Container-Based Application Scheduling 2023 19th International Conference on Network and Service Management, CNSM 2023
18. Brahami M 2020 The influences of knowledge management and customer relationship management to improve hotels performance: A case study in hotel sector Information Resources Management Journal 33
19. Kausar M A, Nasar M and Soosaimanickam A 2022 A Study of Performance and Comparison of NoSQL Databases: MongoDB, Cassandra, and Redis Using YCSB Indian J Sci Technol 15 1532–40
20. Kusuma A B and Hadinata N 2022 The Implementation of the Black Box Method for Testing Smart Hajj Application Ministry of Religion Journal of Information Systems and Informatics 4
21. Ehsan A, Abuhaliqa M A M E, Catal C and Mishra D 2022 RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions Applied Sciences (Switzerland) 12
22. Meshram S U 2021 Evolution of Modern Web Services – REST API with its Architecture and Design International Journal of Research in Engineering, Science and Management 4
23. Prasetyo S E 2021 Design and Implementation of Lightweight Virtualization Using Docker Container in Distributing Web Application with Experimental Methods JOURNAL OF INFORMATICS AND TELECOMMUNICATION ENGINEERING 4
24. Tutterow C and Saint-Jacques G 2019 Estimating Network Effects Using Naturally Occurring Peer Notification Queue Counterfactuals