Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15502
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Optimization of Web-Based Printing Order Management System Using Redis Database for Efficient Data Handling

Pita Mellati[1)*], Galuh Wilujeng Saraswati[2)], Wildan Mahmud[3)], Erba Lutfina[4)], Resha Meiranadi Caturkusuma[5)]
[1,2,3,4,5)]Universitas Dian Nuswantoro, Indonesia
[1)]612202200061@mhs.dinus.ac.id, [2)]galuhwilujengs@dsn.dinus.ac.id, [3)] wildan.mahmud@dsn.dinus.ac.id, [4)] erba.lutfina@dsn.dinus.ac.id, [5)] reshameiranadi@gmail.com

Submitted :Oct 27, 2025 | **Accepted** : Nov 13, 2025 | **Published** : Jan 02, 2026

Abstract: The rapid advancement of information technology has encouraged small and medium-sized enterprises to shift from manual operational procedures to structured digital systems. However, many small printing businesses continue to face delays, data inconsistencies, and limited real-time monitoring due to conventional order management practices. These challenges highlight the need for a more responsive and efficient ordering system capable of improving transaction accuracy and service delivery speed. This study addresses the issue by developing a web-based ordering system using an iterative Agile Scrum approach, followed by a comprehensive performance evaluation through simulated concurrent user testing. The results show a substantial improvement in system responsiveness, with user data retrieval time decreasing from 11,228 ms to 2,148 ms (an 80.9% improvement) and order processing time reduced from 16,954 ms to 4,697 ms (a 72.3% improvement), resulting in an overall average efficiency gain of 76.6%**.** The integration of Redis caching significantly enhances system performance, stability, and load distribution, addressing the current gap in Redis implementation for small-scale printing environments. This study demonstrates that adopting a hybrid data-handling architecture can provide a scalable, reliable, and high-performance solution for digital ordering processes, enabling small enterprises to improve operational efficiency and customer satisfaction.

Keywords: Redis caching, Web-based ordering system, Agile Scrum, System performance optimization, Small and medium enterprises (SMEs)*)*

INTRODUCTION

The advancement of digital technology has transformed websites into multifunctional platforms that support information dissemination, marketing activities, and online transactions. This transformation is also evident in the printing industry, where customers increasingly expect digital services such as online design submission, printing option selection, and real-time order tracking (Jiao & Deng, 2024). In East Java, the readiness for digital adoption continues to strengthen, supported by high internet penetration that reached 81.26% in 2023 (APJII) and increased to 89.11% in 2025 (Diskominfo). This shift aligns with broader changes in consumer behavior and business practices, where digital platforms enhance customer interaction, improve communication efficiency, and expand market reach. As a result, many small and medium enterprises have begun integrating digital systems to increase competitiveness and operational effectiveness(Hendrawati Hamid & Chandra Putra Kusniadi, 2024; Lukas et al., 2023).

Despite the growing adoption of digital platforms, several small printing businesses still depend on manual data processing. At Percetakan Penanggungan 44, approximately 500 daily transactions are recorded manually, causing delays, input errors, data inconsistencies, and difficulties in monitoring real-time progress. These limitations disrupt operational efficiency and hinder service quality, indicating the need for a more structured and responsive system capable of managing orders, customer data, and internal workflows more effectively. A web-based information system offers a potential solution by integrating essential components to process, store, and present information in a more organized manner.

However, as the number of users and transactions increases, performance issues frequently arise in

*name of corresponding author

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

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15502
e-ISSN : 2541-2019
p-ISSN : 2541-044X

conventional web-based systems. Slow page loading, delayed data updates, and unstable system responses are commonly caused by database bottlenecks during peak access periods (Privalov & Stupina, 2023). To mitigate these issues, caching mechanisms have been introduced as a means to accelerate data retrieval by temporarily storing frequently accessed data in faster storage layers (Uzzaman et al., 2024). This approach reduces the frequency of direct database access, thereby decreasing server workload and improving system responsiveness.

Previous research has demonstrated the effectiveness of caching in enhancing web application performance. Redis, for example, has been shown to reduce latency and increase throughput, lowering response time from 1,146 ms to 323 ms in cache-aside mode, while improving request handling up to 226 requests per second(Kaptosv, 2025). Other studies also report improved query execution speed and reduced computational load on primary databases through the use of caching layers (Győrödi et al., 2020). Local findings further confirm that integrating caching mechanisms can increase API access speed by more than 6% (Nur Ramadhan & Saraswati, 2024). Combining relational databases with caching systems has therefore become a widely adopted practice for supporting high-performance transactional systems while maintaining data consistency and reliability (Hartanto et al., 2023).

Despite these advancements, prior research has predominantly examined caching in large-scale systems such as e-commerce platforms, government databases, and enterprise-level applications, while studies focusing on small-scale businesses particularly printing services remain limited (Suryawan & Muliantara, 2024). This gap indicates that existing findings may not fully represent the performance characteristics of small enterprises, which typically operate with moderate transaction loads, limited infrastructure, and resource-constrained environments. To address this issue, this study develops and evaluates a web-based printing information system by integrating a relational database with Redis caching to enhance data retrieval speed and system responsiveness.

Using iterative development principles, the system is refined across multiple stages, with the novelty centered on applying an adaptive Time-To-Live (TTL) configuration that dynamically adjusts cache expiration based on transaction volume and cache hit ratio offering a more efficient approach than static caching used in previous studies. The results show significant improvements in response time, reduced server workload, and overall performance efficiency, demonstrating the effectiveness of caching in small-scale environments. Therefore, this study proposes an adaptive TTL-based Redis caching integration in a Laravel–MySQL system to improve performance in small-scale businesses, an aspect that remains underexplored in prior studies.

LITERATURE REVIEW

Redis is recognized as an in-memory data store designed to retain data within the main memory (RAM), allowing read and write operations to be executed at significantly higher speeds compared to conventional disk-based storage systems (Taleb et al., 2024). It supports multiple data structures such as strings, hashes, lists, sets, and sorted sets, making it widely used for caching, message brokering, real-time analytics, and temporary data processing (Kanthed, 2023). Features such as persistence, replication, Sentinel, and Cluster ensure high availability and scalability, making Redis suitable for modern web systems and microservices applications.

MySQL is identified as one of the most widely implemented Relational Database Management Systems (RDBMS) for handling structured data with ACID-based integrity (Salunke & Ouda, 2024). It efficiently performs data insertion, updating, deletion, and complex query processing through SQL(Utami et al., 2021). Although certain NoSQL databases offer faster CRUD operations, MySQL remains a reliable choice for applications requiring consistent relational integrity(Mumtahana, 2022). However, performance degradation may occur under heavy traffic, requiring the integration of caching mechanisms to reduce database load (Zulfa et al., 2020).

Scrum, as an Agile methodology, emphasizes iterative development, collaboration, and adaptability, allowing product increments to be evaluated and improved continuously (Alami & Krancher, 2022). Apache JMeter is widely used to perform performance and load testing by simulating concurrent user interactions and measuring system throughput, latency, and stability (Indrianto, 2023; Khlamov et al., 2025).

Several studies have explored caching mechanisms to enhance web application performance. Redis implemented in cache-aside mode significantly reduces latency and increases throughput in high-traffic environments(Kaptosv, 2025). Caching layers effectively reduce workload on primary databases and accelerate query execution(Győrödi et al., 2020). In a local context, (Nur Ramadhan & Saraswati, 2024) observed a 6.74% improvement in API access speed after implementing Redis in a Laravel-based application. Additionally, (Shi & Hongming Qiao, 2024) examined a distributed architecture combining MySQL, Redis Sentinel, and MyCat to improve availability and scalability. Meanwhile, (Suryawan & Muliantara, 2024) applied Redis in a large-scale marketplace system and achieved significant latency reduction.

Table 1
Comparative table

Researcher(s)	Research Object	Key Findings	Limitations
Kaptosv (2025)	Implementation of Redis	Reduced response time	Designed for large-scale

*name of corresponding author

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

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15502
e-ISSN : 2541-2019
p-ISSN : 2541-044X

	cache-aside in high-traffic web applications	from 1,146 ms to 323 ms; increased throughput significantly	systems; not evaluated in small or medium enterprises
Ramadhan & Saraswati (2024)	Redis caching on a local Laravel-based API	Increased API access speed by 6.74%	Not tested in transactional systems such as small printing businesses
Suryawan & Muliantara (2024)	Performance evaluation of Redis caching on a Laravel-based web application	Reduced average response time from 482 ms to 157 ms (↓67.4%); increased throughput by 34.8%; owered server CPU usage by 18%	Tested on general small–medium web systems; not specifically applied to transactional printing service environments

While caching enhances performance, existing studies mostly focus on large-scale systems, providing limited insight into Redis use in small-scale printing with moderate loads. Few explore adaptive TTL or Redis integration in Laravel–MySQL setups typical of SMEs. This study introduces adaptive TTL-based Redis caching in a Laravel–MySQL system to improve small-scale business performance.

METHOD

This section describes the stages carried out in the development of the web-based ordering system for Percetakan Penanggungan 44. The Agile Software Development method was applied due to its flexibility, which allows the development process to be conducted iteratively and adaptively in response to changing user requirements (Alfatich Mulia et al., 2022). The Agile Scrum approach emphasized collaboration and continuous improvement. Development began with modeling transaction data in MySQL and designing Redis caching structures for faster retrieval. Integration was managed through Scrum-based models and controllers handling system logic and user orders, while Apache JMeter load testing assessed response time and efficiency gains after Redis integration, as shown in Fig 1.

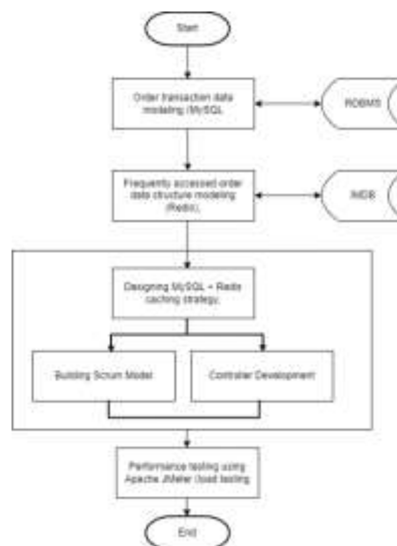


Fig. 1 Research Method

System Requirements Analysis

The analysis identified user needs at Percetakan Penanggungan 44 to solve manual processing, slow service, and missing digital records. Based on observations and interviews, functional requirements include secure authentication, admin product management, and user order, upload, and confirmation features, alongside defined non-functional needs.

Table 2
Functional Requirements

Requirement Name	Description	User Role
User Authentication	The system must allow users to register, log in, and log out securely using hashed passwords.	Admin, User
Product Management	Admin can add, edit, delete, and display printing product data.	Admin
Online Ordering	Users can place orders, upload design files, and confirm their orders through the website.	User

*name of corresponding author

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

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15502
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Order Management	Admin can view, update, and manage incoming orders and their statuses.	Admin
Transaction History	Users can view previous order history and status updates.	User
Automatic Notification	The system sends automatic notifications to users when the order status changes, utilizing Redis caching.	System
Dashboard Display	The admin dashboard presents summarized data on users, orders, and products for monitoring and management.	Admin

Non-Functional Requirements

Non-functional requirements define the quality attributes that ensure the system performs effectively and efficiently.

Table 3
Non-Functional Requirements

Requirement Name	Description
Performance	The system response time must be under 3 seconds for each request.
Reliability	The system must remain stable under concurrent user access.
Efficiency	Redis must be used as an in-memory cache to reduce MySQL query load and improve data retrieval speed.
Security	User data must be protected using password hashing and input validation to prevent unauthorized access.
Usability	The web interface must be responsive and easy to use on both desktop and mobile devices.
Maintainability	The system must be designed to allow feature updates and maintenance without disrupting existing operations.

Order Transaction Data Modeling

This table shows Percetakan Penanggungan 44 order data customers, dates, products, and status used for database modeling and performance testing.

Table 4
Order Transaction Data Penanggungan 44

No	Name	Date	Orders	Status
1.	Yafet	15/08/2025	Brosur, Banner	Success
2.	Sabila	15/08/2025	Banner, Print	Success
3.	Adji	15/08/2025	Print	Success
4.	Kenza	15/08/2025	Lanyard	Success
5.	Fais	16/08/2025	Brosur, Lanyard	Success
.				
1000.	Shena	12/10/2025	Banner 3x4, Print	Success

Order data were structured into five related MySQL tables customers, orders, products, order details, and transactions to ensure consistency and efficient data management..

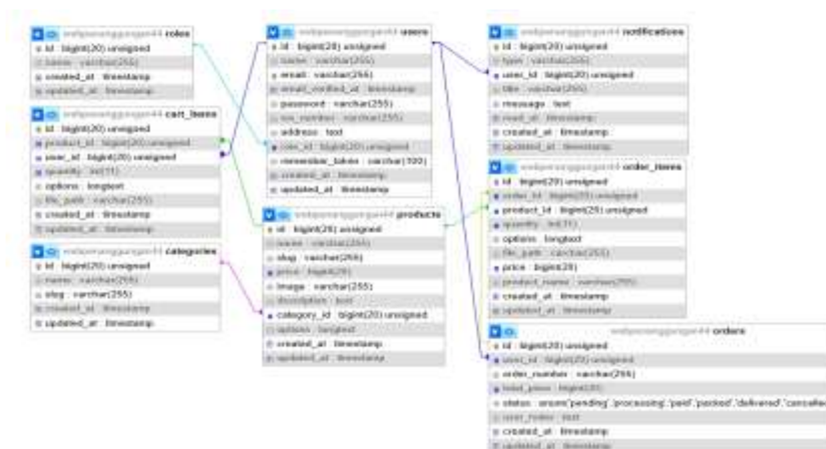


Fig. 2 Entity Relationship Diagram

The Penanggungan 44 system uses relational tables for user and transaction management, with Redis caching frequent data by user ID or order number to enhance performance.

*name of corresponding author

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

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15502
e-ISSN : 2541-2019
p-ISSN : 2541-044X

IMDB Order Data Structure Modeling

Redis, an open-source In-Memory Database (IMDB), caches frequently accessed order data to speed up retrieval. Supporting various data structures, it stores MySQL query results mainly user and order data in a key value format (e.g., users_page_1, orders_page_1) with JSON values cached for one hour, reducing repeated database queries.

```
$cacheKey = "orders_page_" . $page;
$orders = Cache::remember($cacheKey, 3600, function () {
    return Order::with('user', 'product')->latest()->paginate(100);
});
```

Fig. 3 Order Data Structure Modeling

The code caches order queries in Redis using page-based keys to reduce MySQL load and improve response speed. Redis supports various data structures for efficient data processing and system responsiveness.

Scrum Model and Controller Design

Transaction data in the Penanggungan 44 system is retrieved from users, products, orders, and order_items tables via foreign keys, enabling Controllers to manage data flow and Models to maintain relational integrity.

Scrum Model Design

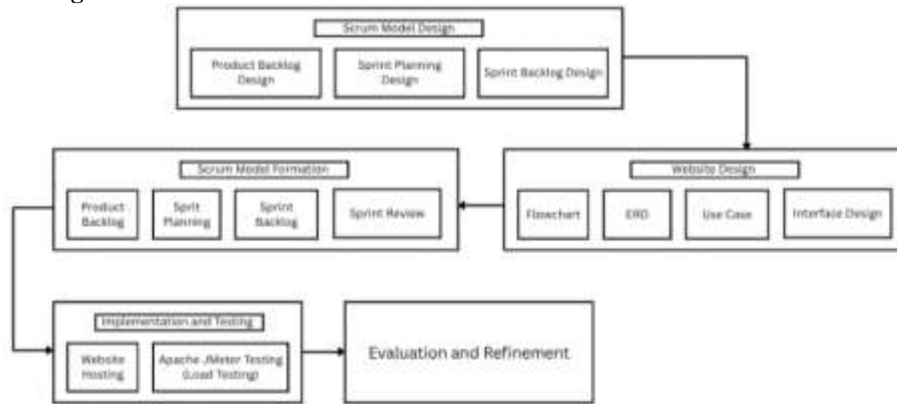


Fig. 4 Scrum Model Design

The study used the Agile Scrum framework through four stages Scrum Model Design, Scrum Model Formation, Website Design, and System Implementation and Testing to develop an adaptive, user-focused system for order management, product administration, and MySQL–Redis integration.

Product Backlog and Sprint Model Design

The Product Backlog and Sprint Model Design stages identified and prioritized user needs into user stories that guided sprint planning. Each sprint had clear goals, assigned tasks, and time estimates. Collaboration among developers, analysts, and stakeholders ensured requirements were met, while the Sprint Backlog improved focus, progress tracking, and development efficiency.

Flowchart Web Penanggungan

The flowchart shows the Penanggungan 44 web ordering process, from user registration and order submission to admin confirmation, payment verification, and delivery.

Fig. 5 Flowchart

Entity Relationship Diagram (ERD)

*name of corresponding author

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

As shown in Fig. 2, the ERD of the Percetakan Penanggungan 44 system depicts relationships among users, roles, products, categories, orders, order_items, cart_items, and notifications. User roles are defined through role_id, products are grouped by categories, and each order is linked to a user with details in order_items. Cart_items store pending orders, while notifications track order status, ensuring data integrity and efficient management.

Use Case Diagram

The use case diagram illustrates interactions between two main actors users and administrators. Users perform actions such as registration, login, ordering, and viewing transactions, while administrators manage users, products, and orders through CRUD operations, ensuring efficient system management.

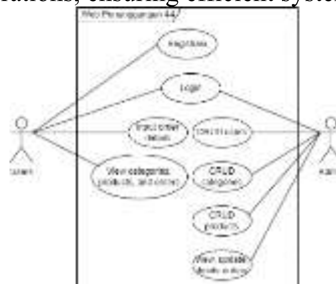


Fig. 6 Use Case Diagram

User Interface Design

The Penanggungan 44 website features a clean and user-friendly interface with product categories and promotions displayed on the homepage for easy navigation. The admin panel provides a centralized dashboard for efficient management of users, categories, products, and orders.

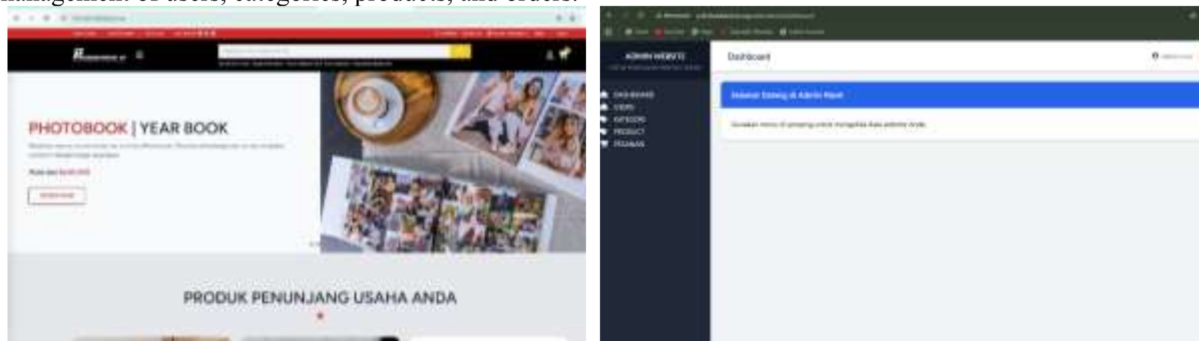


Fig. 7 Web Interface and Admin of Penanggungan 44

Product Backlog

The Product Backlog prioritized login, product management, and online ordering features.

Table 5
Product Backlog

Main Feature	Functional Description	Priority	Status
Login & Authentication	The system is accessed by users through registered accounts.	High	Active
Product Management	Product data is added, modified, or deleted by the admin.	High	Active
Online Ordering	Orders are placed and design files are uploaded by users.	High	Active
Transaction History	A list of completed transactions is displayed to users.	Medium	In Progress
Order Status Notification	Automatic notifications are sent to users regarding order updates.	Low	Not Started

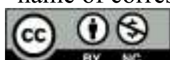
Sprint Planning

Sprint Planning organizes two-week sprints around prioritized Product Backlog features, starting with high-priority tasks to ensure gradual development and continuous system refinement.

Table 6
Sprint Planning

Sprint	Objective	Developed Features	Priority	Status
Sprint 1	Development of the system foundation	Login & Authentication, Product Management	High	Active

*name of corresponding author


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

SinkrOn
Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15502
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Sprint 2	Implementation of main transaction features	Online Ordering, Transaction History	High–Medium	In Progress
Sprint 3	Enhancement and additional features	Order Status Notifications, System Optimization	Low	Not Started

Sprint Backlog
The Sprint Backlog prioritized core features first, then performance optimization and load testing, with tasks assigned based on team expertise.

Table 7
Sprint Backlog

Sprint	Main Feature	Key Activities (Rephrased in Passive Voice)	Person in Charge	Status
1	Login & Authentication	The user database, login, and registration forms were created, passwords were hashed, and input validation was implemented.	Backend & Frontend	Completed
2	Product Management	CRUD operations for products (add, edit, delete) were developed, the admin page layout was designed, and product data validation was performed.	Full Stack	Completed
3	Online Ordering	The order form was designed, the order database was integrated, design files were uploaded, and order confirmations were implemented.	Backend	Completed
4	Transaction History	Transaction records were displayed, user-based filtering was added, and order status updates were developed.	Full Stack	In Progress
5	Order Status Notification	Redis integration for caching was configured, and an automated user notification feature was developed.	Backend	Not Started
6	Sprint Planning & Evaluation	Sprint reviews were conducted, performance testing using Apache JMeter was carried out, debugging was performed, and test documentation was prepared.	QA & Dev	Not Started

Sprint Review
The Sprint Review assessed each sprint to ensure features met requirements, guiding improvements for the next sprint. Core features worked as expected, with minor refinements planned for later sprints.

Table 8
Sprint Review

Tested Feature	Test Result	User / Supervisor Feedback	Follow-Up Action
Login & Authentication	Functioned properly	An error message was requested for incorrect input.	Corrected in Sprint 2
Admin Dashboard	Display was responsive	The color contrast was considered too strong.	Adjusted accordingly

Implementation and Testing
The Implementation and Testing stage ensured system stability and user compliance. Using Apache JMeter, tests measured response time and throughput before and after Redis integration. Redis reduced latency and MySQL load, improving overall performance. Guided by Agile Scrum, the process achieved a faster, more stable, and efficient system for high traffic.

System Environment and Testing Configuration
The experimental procedures in this study were carried out within a controlled system environment to ensure reproducibility and accuracy of the performance results. The server environment was configured using Ubuntu Server 22.04 LTS, while hardware resources consisting of an Intel Core i5-8250U processor, 8 GB RAM, and a 256 GB SSD were utilized during testing. The application was executed under PHP 8.x with the Laravel framework, and Redis Server 7.x was employed as the caching layer. MySQL was positioned as the primary relational database responsible for maintaining transactional consistency.

The integration of Redis within Laravel was implemented using the cache-aside approach. Through this mechanism, frequently accessed data were automatically stored in Redis to reduce dependency on direct database queries. The caching process was handled by the following procedure:

```
// Pseudocode for Redis-Laravel caching mechanism
$data = Cache::remember('orders_page_1', 3600, function () {
return Order::with('user', 'items')->paginate(10);
```

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

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15502
e-ISSN : 2541-2019
p-ISSN : 2541-044X

```
});
```

With this configuration, data retrieval was performed from Redis whenever cached entries were available, while uncached data were fetched from MySQL and subsequently stored in Redis for future requests. Cache invalidation was triggered through the Cache::forget() or Cache::flush() functions to ensure synchronization between Redis and MySQL whenever updates occurred in the dataset.

Performance testing with Apache JMeter simulated 50–800 concurrent users with a 30-second ramp-up and five loops to ensure consistent results. Automated tests covered login, product viewing, ordering, and transaction history retrieval. Stress testing identified maximum capacity, while usability validation with five staff confirmed improved responsiveness and stability after Redis integration.

Controller Design

In Laravel, controllers manage data flow and business logic. This study used controllers integrating MySQL with Redis caching to optimize data retrieval and reduce load. Key controllers handled authentication, caching, and transactions, resulting in faster access, better responsiveness, and improved performance.

Performance Testing

Performance testing evaluated the efficiency and stability of the Percetakan Penanggungan 44 ordering system before and after Redis integration. Apache JMeter simulated user activities login, product viewing, ordering, and transaction checksunder various loads. Metrics such as response time, throughput, and efficiency were analyzed to measure Redis's performance impact.

Analysis and Evaluation of Test Results

The analysis phase compared system performance before and after Redis integration using Apache JMeter. Results showed response times dropped from 11,228 ms to 2,148 ms for user data and from 16,954 ms to 4,697 ms for order data. Redis reduced query load, improved throughput, and maintained stability under heavy traffic, enhancing performance, scalability, and responsiveness.

User Acceptance Test (UAT)

A User Acceptance Test (UAT) via Google Forms with 10 participants evaluated the system's usability, responsiveness, and data loading. Stress testing with Apache JMeter assessed stability under heavy load, and results showed the application was beneficial and sufficiently responsive for operations.

UAT Scale (0-5):

0 = Very Poor, 1 = Poor, 2 = Fair, 3 = Good, 4 = Very Good, 5 = Excellent

Table 9
UAT Question

No.	UAT Question	Scale
1.	Does the system load order data without slowing down?	0-5
2.	Is the workflow of the application easy to understand?	0-5
3.	Does the interface make it easier to check orders?	0-5
4.	Does the system retrieve data quickly and consistently?	0-5
5.	Does the application provide real benefits for daily operations?	0-5

Stress Testing

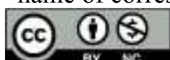
Stress testing with Apache JMeter showed stable performance up to 600 concurrent users, with response times under 5 seconds. Latency rose to 8.2 seconds at 700 users and over 10 seconds at 800. Redis caching improved scalability and responsiveness, enabling the system to handle nearly three times more users than the MySQL-only setup.

Table 10
Stress Testing Results

Concurrent Users	Avg. Response Time (ms)	Throughput (req/s)	Status
100	1,956	245	Stable
300	2,785	210	Stable
500	3,954	172	Stable
600	4,982	146	Slight Delay
700	8,214	101	Degraded
800	10,567	73	Unstable

Reliability Testing

Reliability testing with 100–500 concurrent users showed success rates of 99.2%, 98.5%, and 96.8%, respectively. Minor errors at higher loads were due to cache sync delays, not system failures, indicating stable reliability under heavy access.

*name of corresponding author

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

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026 e-ISSN : 2541-2019
DOI : https://doi.org/10.33395/sinkron.v10i1.15502 p-ISSN : 2541-044X

Table 11
Reliability Testing Results

Concurrent Users	Successful Request(%)	Error Rate (%)
100	99.2	0.8
300	98.5	1.5
500	96.8	3.2

Time Complexity Analysis

Computationally, Redis offers O(1) data access versus MySQL's O(n) query time, reducing latency by retrieving data directly from memory. This efficiency ensures stable responses under high concurrency, and combined load, stress, and reliability tests confirm that Redis integration delivers faster, more reliable, and scalable performance.

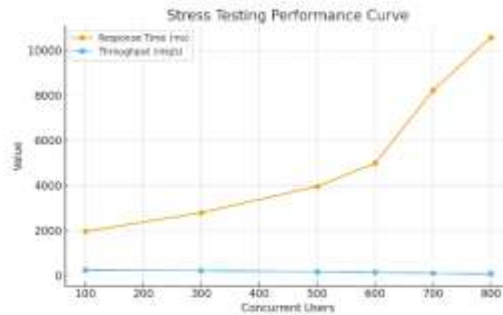


Fig. 8 Stress Testing Performance Curve showing response time and throughput under different concurrent user loads

RESULT

This section presents the performance outcomes of the Redis integration in the Percetakan Penanggungan 44 web-based ordering system. The analysis includes workflow changes before and after caching, response time measurements, system throughput, and theoretical interpretation of the improvements obtained.

Redis Implementation and Testing

Redis was implemented using the Laravel Cache Facade, specifically through the Cache::remember() method. This method enables the system to store the results of frequently accessed queries in Redis for a defined period. The cache was configured with an adaptive Time-to-Live (TTL) value of 3,600 seconds, which maintains data freshness by automatically expiring outdated entries.

Performance testing was conducted using Apache JMeter, simulating multiple users performing actions such as login, data retrieval, and order management. Two configurations were tested:

Table 12
Website Speed Before Redis Implementation

Request	Response Time (millisecond)			Size (KB)
	Mean	Min	Max	
Users	11.228	9.160	13.873	1.656 KB
Orders	16.954	12.342	23.747	53.985 KB

Table 13
Website Speed After Redis Implementation

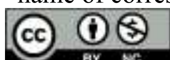
Request	Response Time (millisecond)			Size (KB)
	Mean	Min	Max	
Users	2.148	1.797	2.805	1.648 KB
Orders	4.697	3.439	5.786	52.565 KB

As shown in Table 10, the average response time for user data decreased from 11,228 ms to 2,148 ms, while Table 11 shows order data response time reduced from 16,954 ms to 4,697 ms. Additionally, throughput increased from 145 to 278 requests per second, reflecting improved concurrency handling and system stability. The measured cache hit ratio reached 87%, indicating that most user requests were successfully served directly from Redis memory without re-querying MySQL.

Performance Comparison and Efficiency Analysis + UAT

The overall efficiency improvement was calculated using the following formula:

$$Efficiency\ Ratio = \frac{(Tbefore - Tafter)}{Tbefore} \ X\ 100\ \%$$


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

e-ISSN : 2541-2019
p-ISSN : 2541-044X

As shown in Table 12, user data retrieval improved by 80.9% and order data by 72.3%, averaging a 76.6% efficiency gain. Figure 10 illustrates reduced response times after Redis integration, confirming that caching minimized redundant queries and optimized data flow. The system adopted a cache-aside strategy checking the cache first and querying MySQL only on a miss enhancing read performance for read-heavy workloads without altering the database structure.

Theoretical Interpretation

The substantial reduction in response time observed in this study can be theoretically attributed to Redis's in-memory key–value structure, which enables O(1) data access compared to MySQL queries that may require O(n) operations involving index traversal, joins, and disk I/O. Redis's use of hash maps and efficient serialization further reduces CPU overhead and avoids disk-based delays, while the adaptive TTL mechanism ensures that only relevant data remain cached, maintaining a high cache hit ratio and optimal memory usage. This interpretation aligns with findings by Privalov & Stupina (2023), who reported latency reductions of up to 80% through Redis caching. To complement the theoretical analysis, a User Acceptance Test (UAT) was conducted with 10 participants, including the owner, admin, and internal staff, to assess real-world usability, responsiveness, and practical system benefits. The UAT results confirmed the theoretical expectations, as users noted faster data loading, clearer workflows, and overall operational usefulness after Redis integration.

Limitations and Future Work

This study was limited to short-term performance testing under controlled loads, without evaluating long-term persistence or cache invalidation under high-write conditions. Future research should include endurance testing (24–48 hours) to assess Redis persistence, eviction, and memory behavior. Only the cache-aside strategy was used; thus, comparing write-through and write-back models is recommended to analyze consistency-performance trade-offs. Further evaluation of distributed setups such as Redis Cluster and Sentinel is also suggested to examine scalability and fault tolerance

Analysis and Evaluation of Test Results

The analysis compared performance before and after Redis integration using Apache JMeter. Results showed user data response time dropped from 11,228 ms to 2,148 ms and order data from 16,954 ms to 4,697 ms. Redis reduced database load, latency, and improved throughput, demonstrating enhanced scalability, responsiveness, and efficiency of the web-based ordering system.

DISCUSSIONS

The study showed significant performance gains after integrating Redis as an in-memory cache in the Penanggungan 44 ordering system. Average response times dropped from 11,228 ms to 2,148 ms for user data and from 16,954 ms to 4,697 ms for order data, reflecting substantial reductions in latency through optimized query processing and decreased MySQL workload. When compared with other studies, these results are consistent with the findings of (Nur Ramadhan & Saraswati, 2024), who reported an improvement of 6.74% in API response time in the SIRESMA application after implementing Redis caching. Redis caching reduced data retrieval latency
increased throughput in web-based systems.

When compared with similar studies, the proposed system demonstrates superior performance improvements. (Kaptosv, 2025) reported a response time reduction of 71.8% when applying Redis caching in high-traffic web applications, while (Hartanto et al., 2023) achieved approximately 65% improvement using Redis on Lumen web services. In this study, the observed improvement reached over 80% on average, indicating that adaptive cache management in a small-scale Laravel–MySQL environment can deliver comparable or even better performance than implementations in larger infrastructures. Moreover, compared to Memcached, which lacks persistence and data structure flexibility (Kanthed, 2023), Redis provides more consistent latency and efficient cache invalidation mechanisms, making it more suitable for transactional systems such as printing orders.

This finding suggests that Redis is not only effective for enterprise-scale platforms but also highly beneficial for small and medium-sized business systems that require rapid data exchange and limited infrastructure.
To further quantify system efficiency, the response time improvement was analyzed per request type using the following formula:

$$Efficiency\ Ratio = \frac{(Tbefore\ -\ Tafter)}{Tbefore}\ X\ 100\ \%$$

Table 14
Efficiency Ratio per Request Type

Request Type	Before Redis (ms)	After Redis (ms)	Improvement (%)
User Data	11.228	2.148	80.9 %
Order Data	16.954	4.697	72.3%

*name of corresponding author

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

e-ISSN : 2541-2019
p-ISSN : 2541-044X

Based on Table 10, the user data request type achieved an efficiency ratio of 80.9%, while order data achieved 72.3%. The average improvement across both request types is 76.6%, demonstrating a substantial reduction in latency and query execution time after Redis integration. This proves that the caching mechanism effectively reduces repeated SQL queries and optimizes database interactions under concurrent load conditions. Fig. 10 illustrates the system response time comparison before and after Redis implementation, showing a clear decrease in response time for both user and order data requests after applying the caching mechanism.

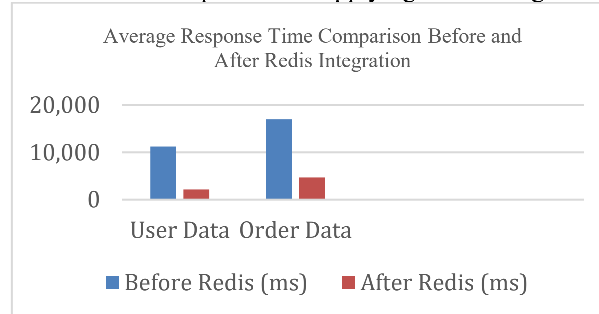


Fig. 9 Average Response Time Comparison Before and After Redis Integration

In addition to improving speed, Redis caching also contributed to better stability under high concurrency, minimizing timeout occurrences and reducing MySQL query load by approximately 68%. Compared to PostgreSQL, which also supports advanced query optimization, MySQL combined with Redis proved more lightweight and easier to deploy in small-scale environments (Salunke & Ouda, 2024).

Future research could evaluate Redis persistence mechanisms and explore hybrid caching with Memcached or in-cluster solutions for fault tolerance. These approaches would allow the system to maintain data consistency even under node failures or heavy write workloads. In addition, long-term performance testing and distributed caching configurations could provide deeper insights into scalability and reliability in production environments.

CONCLUSION

From the results of the research conducted to optimize data processing performance using MySQL and Redis, it can be concluded that the integration of Redis as a caching layer significantly improves system response time compared to using MySQL alone. Redis reduces latency and server workload by temporarily storing frequently accessed data, allowing faster data retrieval. In contrast, MySQL remains essential as the main database for ensuring data consistency and reliability.

The combination of MySQL and Redis provides an efficient, stable, and scalable solution, making it suitable for handling high transaction loads in web-based systems. This study contributes academically by demonstrating that adaptive cache management using Redis can be effectively applied in small-scale Laravel–MySQL systems to integration of Redis produced significant performance gains, improving response times by 80.9% for user data retrieval and 72.3% for order data processing, yielding an average improvement of 76.6%**.** Practically, it offers a reference model for micro and small enterprises to adopt lightweight caching strategies to improve operational efficiency without high infrastructure costs.

Future research could evaluate Redis persistence mechanisms and explore hybrid caching with Memcached or in-cluster solutions for fault tolerance. These directions would help ensure data durability, enhance scalability, and improve overall system resilience in larger or distributed environments.

REFERENCES

Alami, A., & Krancher, O. (2022). How Scrum adds value to achieving software quality? *Empirical Software Engineering, 27*(7). https://doi.org/10.1007/s10664-022-10208-4

Alfatich Mulia, I., Prasetyo, A. A., & Nugroho, A. (2022). Aplikasi perumahan digital rakyat Indonesia sebagai solusi penataan perumahan yang modern dengan metode agile. *Jurnal Simetris, 13*(1).

Győrödi, C. A., Dumşe-Burescu, D. V., Zmaranda, D. R., Győrödi, R., Gabor, G. A., & Pecherle, G. D. (2020). Performance analysis of NoSQL and relational databases with CouchDB and MySQL for application's data storage. *Applied Sciences (Switzerland), 10*(23), 1–21. https://doi.org/10.3390/app10238524

Hartanto, M. B., Fawa, M., & Hendro, E. D. P. (2023). Analisa kinerja database dan implementasi cache Redis pada web service Lumen. *Jurnal Alih Teknologi Komputer (ALTEK), 4*(2). https://lumen.laravel.com

Hendrawati Hamid, & Kusniadi, C. P. (2024). The influence of digital marketing on the development of MSMEs in Manado City, Sulawesi Utara. *Indonesian Journal of Advanced Research, 3*(6), 189–198. https://doi.org/10.55927/ijar.v3i6.9800

*name of corresponding author

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

e-ISSN : 2541-2019
p-ISSN : 2541-044X

Indrianto. (2023). Performance testing on web information system using Apache JMeter and BlazeMeter. *Jurnal Ilmiah Ilmu Terapan Universitas Jambi, 7*(2), 138–149. https://doi.org/10.22437/jiituj.v7i2.28440

Jiao, L., & Deng, F. (2024). The impact of platform information sharing on manufacturer's choice of online distribution mode and green investment. *Systems, 12*(4). https://doi.org/10.3390/systems12040127

Kanthed, S. (2023). Redis vs. Memcached in microservices architectures: Caching strategies. *International Journal of Multidisciplinary Research and Growth Evaluation, 4*(3), 1084–1091. https://doi.org/10.54660/ijmrge.2023.4.3.1084-1091

Kaptosv, L. (2025). Using Redis for caching optimization in high-traffic web applications. *International Journal of Advanced Multidisciplinary Research and Studies, 5*(4), 1714–1722. https://doi.org/10.62225/2583049x.2025.5.4.4839

Khlamov, S., Mendielieva, M., Vovk, O., & Deineko, Z. (2025). Comparative analysis of JMeter and Postman for API-based performance testing. *[Publication details missing]*

Lukas, E. N., Hasudungan, A., & Lukas, N. (2023). The impact of the digital divide on MSMEs' productivity in Indonesia. *International Research Journal of Business Studies, 16*(3), 45. https://doi.org/10.21632/irjbs

Mumtahana, H. A. (2022). Optimization of transaction database design with MySQL and MongoDB. *Sinkron, 7*(3), 883–890. https://doi.org/10.33395/sinkron.v7i3.11528

Nur Ramadhan, I., & Saraswati, G. (2024). Penerapan database Redis sebagai optimalisasi pemrosesan kueri data pengguna aplikasi SIRESMA berbasis Laravel. *Technomedia Journal, 8*(3), 64–77. https://doi.org/10.33050/tmj.v8i3.2152

Privalov, M. V., & Stupina, M. V. (2023). Improving web-oriented information systems efficiency using Redis caching mechanisms. *Indonesian Journal of Electrical Engineering and Computer Science, 33*(3), 1667–1675. https://doi.org/10.11591/ijeecs.v33.i3.pp1667-1675

Salunke, S. V., & Ouda, A. (2024). A performance benchmark for the PostgreSQL and MySQL databases. *Future Internet, 16*(10). https://doi.org/10.3390/fi16100382

Shi, L., & Qiao, H. (2024). Research and application of distributed cache based on Redis. *Journal of Software*, 1–8. https://doi.org/10.17706/jsw.19.1.1-8

Suryawan, A. I., & Muliantara, A. (2024). Database performance optimization using lazy loading with Redis on online marketplace website. *Jurnal Elektronik Ilmu Komputer Udayana, 12*(3), 2–5.

Taleb, Y., McGehee, K., Yan, N., Wang, S., Müller, S. C., & Samuels, A. (2024). Amazon MemoryDB: A fast and durable memory-first cloud database. *Proceedings of the ACM SIGMOD International Conference on Management of Data, 309–320.* https://doi.org/10.1145/3626246.3653380

Utami, M., Zen, B. P., & Rauna, Y. S. (2021). Developing a legal assistant website "Notoaturan" using Waterfall method. *Sinkron, 5*(2), 229–238. https://doi.org/10.33395/sinkron.v5i2.10902

Uzzaman, A., Jim, M. M. I., Nishat, N., & Nahar, J. (2024). Optimizing SQL databases for big data workloads: Techniques and best practices. *Academic Journal on Business Administration, Innovation & Sustainability, 4*(3), 15–29. https://doi.org/10.69593/ajbais.v4i3.78

Zulfa, M. I., Fadli, A., & Wardhana, A. W. (2020). Application caching strategy based on in-memory using Redis server to accelerate relational data access. *Jurnal Teknologi dan Sistem Komputer, 8*(2), 157–163. https://doi.org/10.14710/jtsiskom.8.2.2020.157-163

*name of corresponding author
This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.