Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

15th February 2026. Vol. 5. No. 2; e-ISSN: 2808-4519

Implementation of Prototyping Method in Developing a Web-Based Cos Management System Using Laravel

Yudistio Izza Al Farisi [1], Oman Komarudin[2], Intan Purmanasari[3]

[1,2,3]*Faculty of Computer Science, Singaperbangsa University, Karawang*
2110631170156@student.unsika.ac.id[1*]

Abstract

This study develops a web-based boarding house management system using the Prototyping method to address administrative issues at Kos D'Rosse, where data was previously managed manually through Google Form, Excel, and WhatsApp. The Prototyping model enabled iterative requirements gathering and user evaluation to refine system features. The system was built using the Laravel framework with an MVC architecture and includes modules for tenant management, room monitoring, payment processing, financial reporting, and a real- time dashboard. Blackbox testing confirmed that all features functioned according to user needs, while whitebox testing produced low Cyclomatic Complexity values, indicating simple and maintainable program logic. User Acceptance Testing (UAT) showed improvements in operational efficiency, data accuracy, and decision-making speed. The results demonstrate that the system integrates all management activities into a single platform, reduces administrative workload, and provides accurate, real-time information. Overall, the Prototyping approach and Laravel MVC support structured development and effective system performance.

Keywords: *Laravel, Prototyping, Web-based System, Boarding House Management, MVC*

1. Introduction

The development of information technology in the last two decades has driven significant digital transformation in various sectors, from education and health, transportation, business, and property. Innovations in hardware, software, and communication networks have enabled the automation of manual processes, increased data accuracy, expanded access to information, and delivered faster, more efficient, and more integrated systems, with the internet as the primary infrastructure supporting global connectivity[1]. One sector that has felt the impact of the development of information technology is temporary housing, particularly boarding houses. Boarding houses are often chosen by students and workers because of their flexible nature, with a relatively high turnover rate of residents[2]. Unlike apartments or rented houses, boarding houses are characterized by tenants only having the right to a private room, while public facilities such as the kitchen or living room are shared with other residents. The phenomenon of urbanization, the growth of the student population, and the mobility of the workforce between cities are increasingly driving the increasing demand for boarding houses in urban areas. This condition has led to the rapid growth of the boarding house industry in Indonesia and has become one of the most dynamic property sectors. A digital boarding house management system is an information system that supports structured boarding house management, including resident data, room status, payment schedules, and digital transaction records [3]. Web-based systems enable efficient interaction between owners, managers, and tenants through online registration features, room monitoring, rental contract management, and an integrated payment system with high accessibility from various locations [4]. In addition, a boarding house rental information system must be able to manage dynamic resident data, track tenant history, payment status, rental duration, and generate accurate reports for business evaluation, including managing room types with different rates [5]. The advantages of a digital boarding house management system include operational efficiency, data accuracy, real-time information access, and the ability to generate comprehensive reports, while enabling remote business monitoring, which is relevant for boarding houses with remote management and high operational complexity such as Kos D'Rosse [6].

As demand for boarding houses increases, boarding house management demands an efficient, transparent, and easily accessible system, particularly for recording resident data, room management, and payment transactions. Digitalization allows boarding house owners to search for rooms, communicate with tenants, and record and verify transactions online, easing the management burden while making things easier for tenants. D'Rosse Boarding House in Semarang, an exclusive women's boarding house with 100 rooms that has been consistently full since 2019, uses Google Forms, Excel, and WhatsApp for management. However, this manual method creates obstacles such as scattered data, the risk of input errors, slow payment verification, and non-real-time financial reports. This condition increases the administrative burden, complicating evaluation and planning, so an integrated web-based boarding house management system is needed to improve efficiency, accuracy, and speed of decision-making [7].

To address this, the Prototyping model in the Software Development Life Cycle (SDLC) can be applied, with the creation of a prototype as an initial representation of the system that is tested by users and improved iteratively [8]. This model allows system requirements that are not clear from the start to be adjusted, and minimizes the risk of specification errors. The Model–View–Controller (MVC) architectural pattern is also relevant, with the separation of the system into Model, View, and Controller to facilitate the management of data, user interfaces, and application flows [9], thus facilitating integration with the prototyping model.
The Laravel framework, which is based on PHP, supports the implementation of MVC with features such as Eloquent ORM, Blade Template Engine, Routing, Middleware, and Artisan CLI to simplify development, interaction with databases, and the creation of dynamic views. The use of Unified Modeling Language (UML) through various diagrams, as well as system testing using white box and black box testing [10], is important to ensure a flexible, structured, and actively user-involved boarding management system, overcoming the limitations found in previous research [11].
Based on these problems, an integrated and web-based boarding house management system is needed, so that it can combine all important functions ranging from tenant data recording, room management, to payment transactions into one platform. Therefore, this research is entitled "Implementation of Prototyping Model in Developing a Web-Based Boarding House Management System Using Laravel" This system is expected to provide a comprehensive solution to the digitalization of boarding house management, increase operational efficiency, maintain data accuracy, and improve the quality of service to residents.

2. Theoretical Basics

2.1. Prototyping Method in System Development

Prototyping is an approach to software development that involves creating an initial model (prototype) of the system being developed. This prototype serves to provide an initial overview of the system being built and allows users to provide direct feedback on the system's functionality and appearance. Unlike traditional approaches, which require the entire system to be completed before testing, prototyping allows for an iterative development process.

The primary advantage of the prototyping method is its ability to address uncertain user needs. A development process based on user feedback allows the system to be more tailored to their needs, thereby reducing the risk of development errors. In the context of developing a web-based boarding house management system, this method allows boarding house managers to identify and address existing operational issues through field-tested prototypes.

2.2. Laravel Framework and MVC Architecture

Laravel is a PHP framework widely used in web-based application development, known for its excellence in simplifying application development and maintenance. One of the main concepts in Laravel isModel-View-Controller (MVC), a software architecture pattern that separates an application into three main components:

a. Model: Responsible for application logic and interaction with the database.
b. View: Responsible for displaying data to users, namely the user interface (UI).
c. Controller: Connecting models and views, and handling application flow logic.

The MVC approach allows for more structured and easily maintained application development. The Laravel framework also includes features like the Eloquent ORM, Blade Template Engine, and Artisan CLI, which support efficient web-based application development. Using Laravel in developing a web-based boarding house management system simplifies the creation of dynamic user interfaces and the management of resident, room, and payment data.

2.3. Web-Based Boarding House Management System

A web-based boarding house management system is a software application that simplifies the management of a boarding house or temporary residence. This system is typically used to manage resident data, room status, payment schedules, and financial reports. Using a web-based system, boarding house owners and managers can access data in real time from any internet-connected device, simplifying operational management and oversight.

In the context of boarding house management, web-based applications offer various benefits, such as increased operational efficiency, reduced data entry errors, and ease of automated payment and room status management. The primary advantage of a web-based boarding house management system is its ability to centrally store data, reduce reliance on manual systems prone to human error, and increase transparency and accuracy of information.

2.4. System Testing: Blackbox and Whitebox

System testing is a critical step in software development, ensuring that the system functions properly according to established specifications. There are two types of testing commonly used in system development: blackbox testing and whitebox testing.

a. Black Box Testing: Testing is conducted with a focus on system functionality without looking at the internal program code. This testing assesses whether the system functions according to user requirements. For example, in a boarding house management system, testing might be conducted to ensure that the payment module is functioning properly, or that financial reports are generated correctly.
b. Whitebox Testing: Testing is performed by analyzing the application's internal logic, including code structure and program flow. The goal is to find potential bugs or errors in the code that are not visible in black-box testing. This testing is often performed using analysis techniques such as cyclomatic complexity, which measures the complexity of the logical flow within the program code.

Both types of testing are important to ensure that the application not only functions according to specifications, but also has efficient and maintainable code.

2.5. Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a visual modeling language used to describe the design and structure of software systems. UML provides a variety of diagrams that illustrate various aspects of a system, such as activity flows, interactions between objects, and class structures. Some types of UML diagrams used in developing web-based boarding house management systems include:

a. Use Case Diagram: Describes the interaction between the user and the system, as well as the functionality provided by the system.
b. Activity Diagram: Displays the workflow or sequence of activities in the system.
c. Class Diagram: Shows the class structure in the system and the relationships between classes.
d. Sequence Diagram: Describes the sequence of interactions between objects in a system based on time.

The use of UML helps the development team to understand the system requirements thoroughly and design the system in a more structured and organized manner.

3. Research Methods

3.1 Research Approach

This research uses a descriptive qualitative approach to develop and evaluate a web-based boarding house management system using the prototyping method in software development. This method was chosen because of its ability to directly respond to user needs through iteration and prototype testing. The developed prototype will be used to obtain user feedback and will be continuously refined.

Prototyping allows for rapid development cycles by actively involving users at every stage, which in turn improves the system's suitability to user needs. This research focuses on developing a system that can simplify boarding house management with features such as resident management, payments, financial reports, and a real-time dashboard.

3.2 Research Object

The object of this research is Kos D'Rosse, an exclusive women's boarding house in Semarang with 100 rooms and a high occupancy rate. This boarding house has been using a manual system for managing resident data, rooms, payments, and financial reports through platforms such as Google Forms, Excel, and WhatsApp. The existing system has many limitations, such as the risk of data input errors, information asymmetry, and delays in payment processing. Therefore, this research aims to develop an integrated web-based boarding house management system to improve operational efficiency and data accuracy.

3.3 Research Stages

This research follows systematic steps in developing a web-based system. The stages carried out in this research are as follows:

1. Requirements GatheringAt this stage, system requirements were gathered through interviews and observations with the management of D'Rosse Boarding House. Information gathering included identifying problems encountered with the manual

system and desired features for a web-based boarding house management system. The results of this requirements gathering served as the basis for designing a prototype of the system to be developed.

2. Prototype DesignBased on the gathered requirements, an initial prototype of a web-based boarding house management system was designed. This prototype included the user interface (UI), database structure, and key modules such as resident management, rooms, payments, financial reports, and a dashboard. This design used a Model-View-Controller (MVC) approach with the Laravel framework.
3. System DevelopmentDuring the development phase, the system was built using the Laravel framework, which implements an MVC architecture. This development process involved creating a database, user interface, and application logic to manage resident data, rooms, payments, and financial reports. The system also features a reporting module that allows managers to download reports in PDF or Excel format.
4. Prototype TestingAfter the prototype was developed, testing was conducted to ensure the system met user needs. The testing involved Kos D'Rosse management, who used the prototype in real-world conditions and provided feedback on functionality, speed, and ease of use. The management's feedback was used to refine and improve the prototype.
5. System Evaluation and ImprovementBased on the prototype testing results, an evaluation of the implemented features was conducted. User feedback was used to correct errors and improve the system's quality. This evaluation included analysis of system performance, data accuracy, and user satisfaction. The system was refined and refined through repeated iterations, in accordance with the basic principles of the prototyping method.
6. Blackbox and Whitebox Testing
    a. Blackbox TestingTesting is performed to test system functionality based on the input and output generated by the system. This testing involves testing all system modules, including resident management, rooms, payments, and reports, to ensure that all features function according to user requirements.
    b. Whitebox TestingTesting is performed to ensure that the application's internal logic is functioning properly and that there are no errors in the program code. This testing involves analyzing the program flow and evaluating the code using techniques such as cyclomatic complexity to measure the complexity of the program's logic.
7. User Acceptance Testing (UAT)After the system has been technically tested, user acceptance testing is conducted to measure the extent to which it meets the expectations and needs of boarding house managers. Managers are asked to use the system in their daily operations and provide ratings on ease of use, efficiency, and the quality of the system's functionality.

3.4. Tools and Materials

1. Laravel Framework: Used to develop PHP-based web applications with MVC architecture.
2. MySQL/PostgreSQL Database: Used to store resident data, rooms, payments, and financial reports.
3. UML (Unified Modeling Language): Used to model systems and describe diagrams such as use cases, activity diagrams, and class diagrams.
4. Testing Tools: Testing is performed using manual and automated testing tools to test the functionality and performance of the system.

3.5. Data Analysis

Data collected during the trial and evaluation phase will be analyzed qualitatively to assess whether the developed system meets its intended objectives. This analysis will include assessing system efficiency, user satisfaction, and the system's impact on boarding house operational management. Additionally, test data, such as black-box and white-box test results, will be used to evaluate the system's technical quality.

4. Results and Discussion

4.4. Initial Requirements

Based on the Initial Requirements stage, system requirements gathering began with direct communication through interviews with the management of Kos D'Rosse. Interview results revealed that current resident data management is still scattered across Google Forms, Excel, and WhatsApp, making the manual recording process prone to errors, data duplication, and asynchronous data. Checking room availability is time-consuming because it requires manually opening and marking data, while the payment process involves several verification stages that are time-consuming and risky for errors. Financial reports are also slow and inaccurate because data is scattered across various platforms. The existing partial digitization actually increases the administrative burden, especially for large-capacity boarding houses like Kos D'Rosse. This situation emphasizes the need for an integrated web-based boarding house management system that can combine resident, room, and payment data in a single platform, making operational processes more efficient, accurate, and supporting rapid decision-making.

Based on interviews, the main obstacles in managing Kos D'Rosse are the lack of an integrated system, resulting in scattered resident data across Google Forms, Excel, and WhatsApp, error-prone manual input processes, slow room checks, inefficient payments, and non-real-time financial reports. To address these issues, it is proposed to develop a web-based system with tenant management modules (online forms and automatic verification), room management (real-time with resident history), payment modules (auto-upload proof and notifications), financial report modules (automatic and can be exported to PDF), and a dashboard that displays data summaries, occupancy

statistics, and monitoring of late payments. Thus, this system is expected to improve operational efficiency, minimize errors, and support fast, data-driven decision-making.

4.5. Design

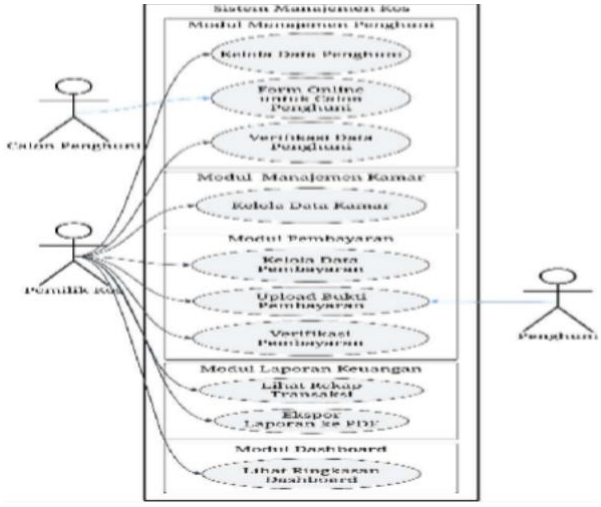


Fig. 1: Use Case Diagram from the results of problem identification

*Use case diagram*The D'Rosse Kos management system showcases key workflows that simplify boarding house management. Boarding house owners can manage resident data, verify prospective residents, manage rooms with real-time information, record and verify payments, and monitor and export financial reports to PDF. Prospective residents can register independently through an online form. The system provides automatic notifications and confirmation of each action, displaying important data such as the number of residents, occupancy rate, room status, and late payments. This makes the entire administrative process more efficient, reduces manual errors, and supports data-driven decision-making.

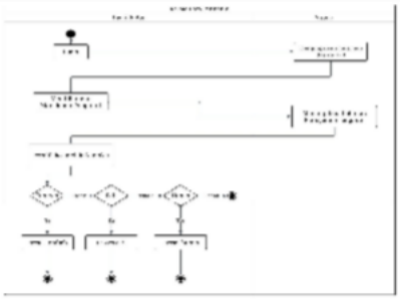


Fig. 2: Resident data management activity

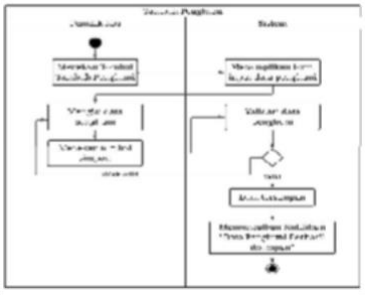


Fig. 3: Activity of adding residents

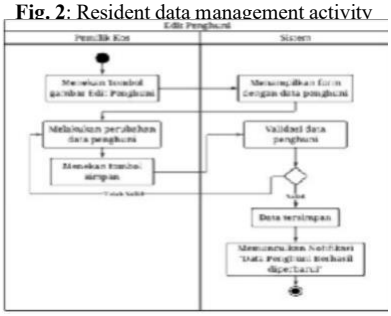


Fig. 4: Resident edit action activity

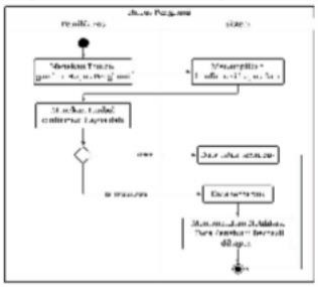
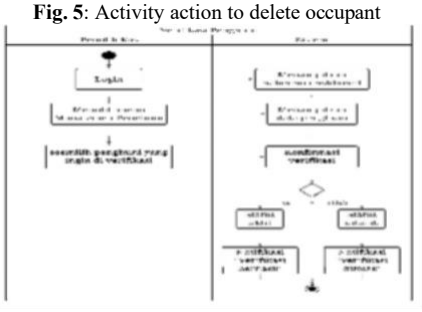


Fig. 5: Activity action to delete occupant



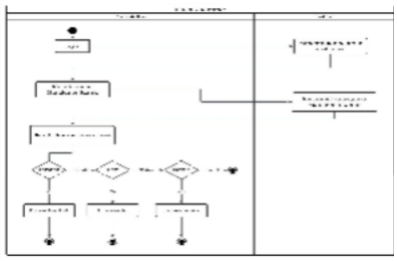


Fig. 8: Room data management activity

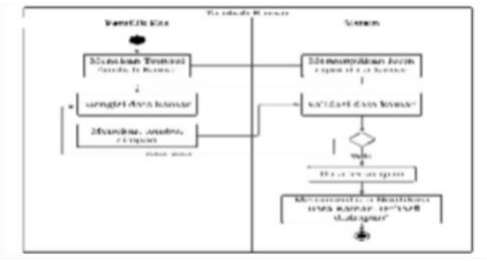


Fig. 9: Add room action activity

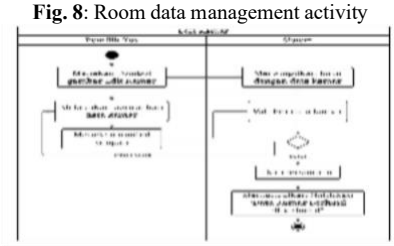


Fig. 10: Room edit action activity

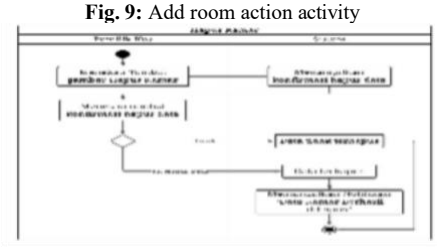


Fig. 11: Delete room action activity

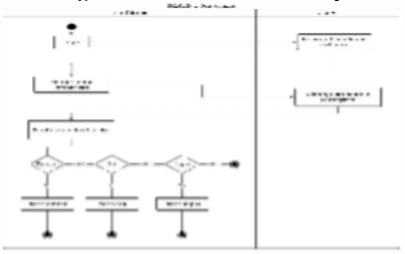


Fig. 12: Payment data management activity

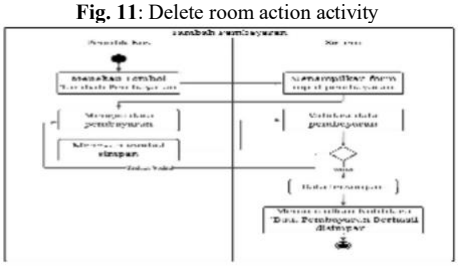


Fig. 13: Action activity adding payment

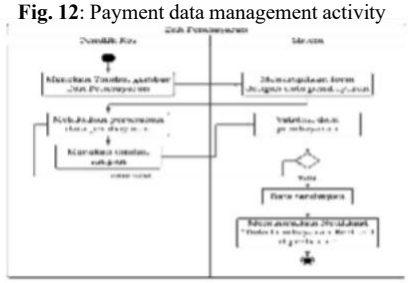


Fig. 14: Payment edit action activity

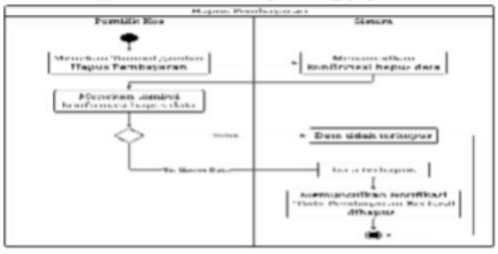


Fig. 15: Delete payment action activity

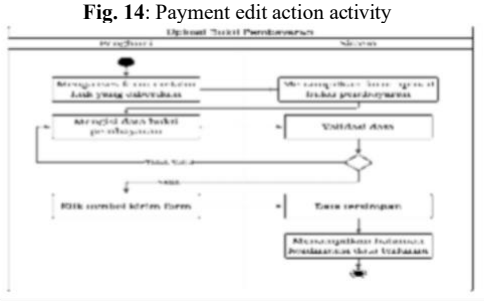


Fig 16. Activity upload proof of payment

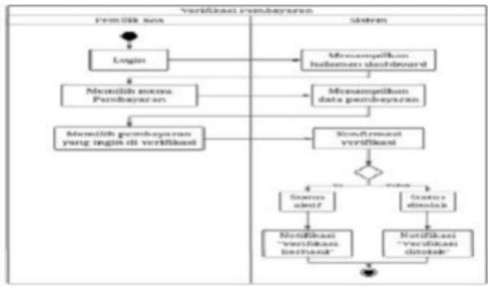


Fig 17. Payment verification activity

Fig. 18: Activity view transaction summary

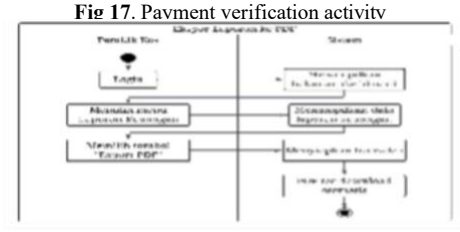


Fig. 19: Activity export report to PDF

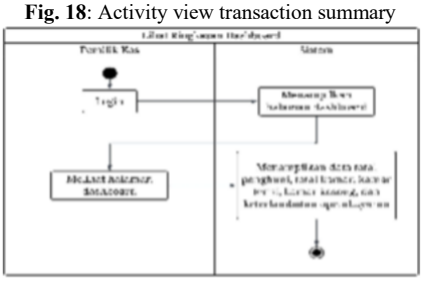


Fig. 20: Activity view dashboard summary

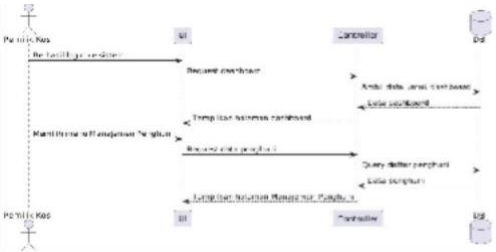


Fig. 21: Resident data management sequence

Fig. 22: Sequence of actions to add occupants

Fig. 23: Resident edit action sequence

Fig. 24: Sequence of actions to delete an occupant

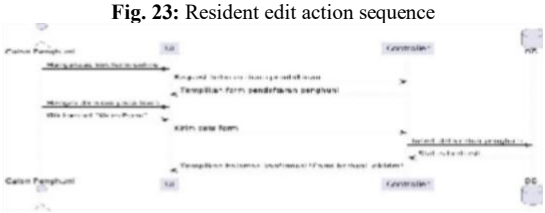


Fig. 25: Online form sequence for prospective residents

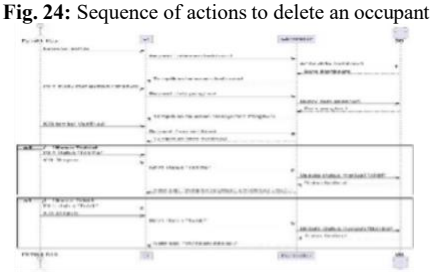


Fig. 26: Resident data verification sequence

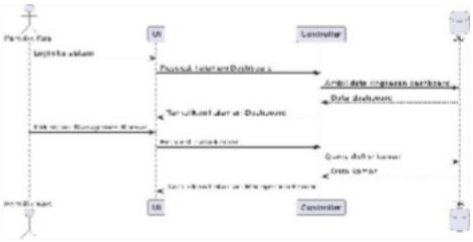


Fig. 27: Room data management sequence

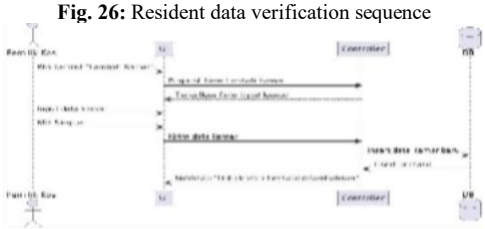


Fig. 28: Sequence of actions to add a room

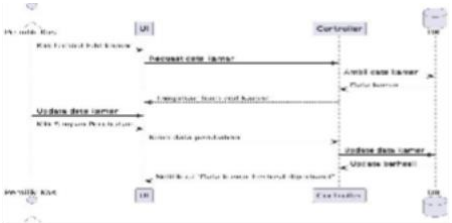


Fig. 29: Room editing action sequence

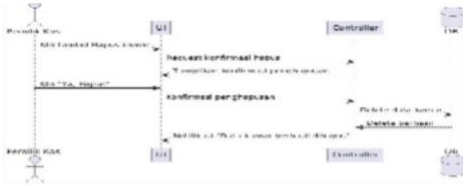


Fig. 30: Sequence of actions to delete a room

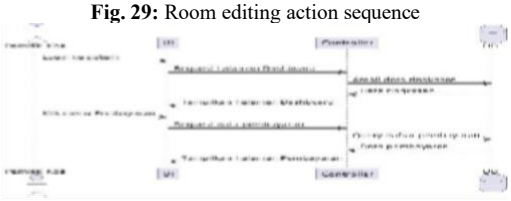


Fig. 31: Payment data management sequence

Fig. 32: Sequence of actions to add payment

Fig. 33: Payment edit action sequence

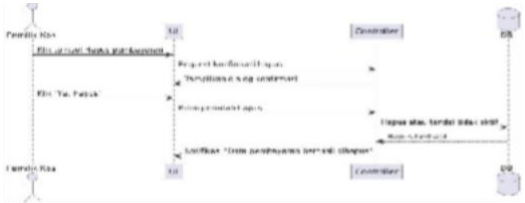


Fig. 34: Sequence of actions to delete payment

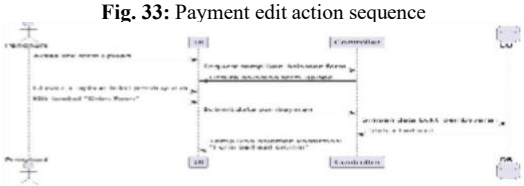


Fig. 35: Sequence for uploading proof of payment

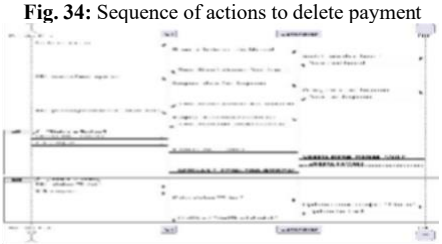


Fig. 36: Payment verification sequence

Fig. 37: Sequence view transaction summary

Fig. 38: Report export sequence to PDF

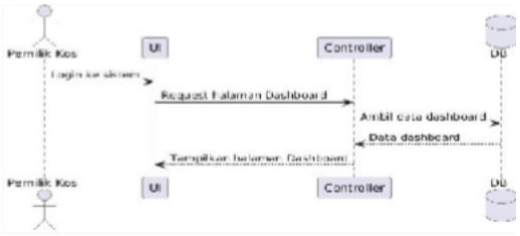


Fig. 39: Sequence view dashboard summary

Fig. 40: Class diagram of the boarding house management system

*Class diagram*The Kos D'Rosse management system describes the organized structure of all entities and the relationships between classes. The Building and Floor classes represent buildings and floors, allowing the creation, updating, and deletion of location data. The RoomType and Room classes manage room type, number, status, and occupant history. The Occupant class stores identity, contact information, status, and rental dates, and handles verification, documentation, and notifications. The Payment, PaymentProof, and PaymentVerification classes manage transactions, store payment receipts, and record verification results. The OccupantVerification class records the history of the occupant verification process, while the FinancialReport generates financial summaries and reports that can be exported to PDF or Excel. The Dashboard class presents statistical summaries, including the number of rooms, occupants, vacant rooms, and due dates. Overall, this design ensures efficient management of rooms, occupants, and payments and supports data-driven decision-making.

4.6. Prototyping

In this section, the prototype focuses on common interface elements that are used consistently across modules. This common interface includes components such as notifications, action confirmations, action buttons (add, edit, delete), and sidebar navigation.

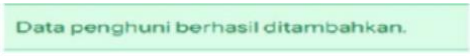


Fig. 41: General view of notifications

Fig. 42: General view of action confirmation

Fig. 43: General view of the add action

Fig. 44: General view of edit and delete actions

Fig. 45: General view of navigation

Fig. 46: Resident data management page

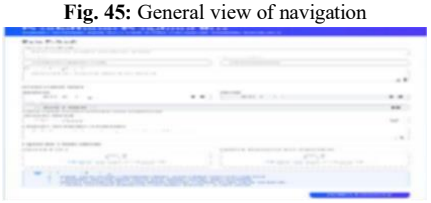


Fig. 47: Online form page for prospective residents

Fig. 48: Resident data verification page

Fig. 49: Room data management page

Fig. 50: Payment data management page

Fig. 51: Proof of payment upload page

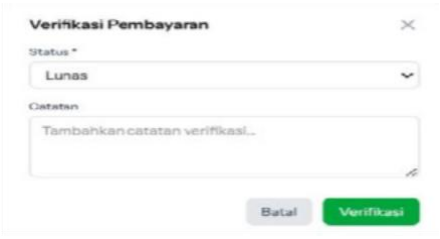


Fig. 52: Payment verification page

Fig.53: Transaction summary view page

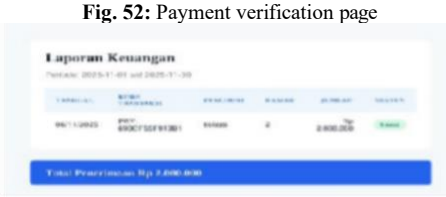


Fig. 54: Export report to PDF page

Fig .55: View dashboard summary page

4.7. Customer Evaluation

Table 2:. Comparison of results before and after evaluation of the general display of action confirmation

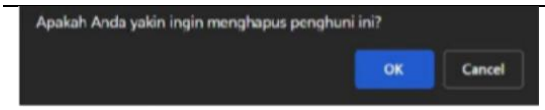
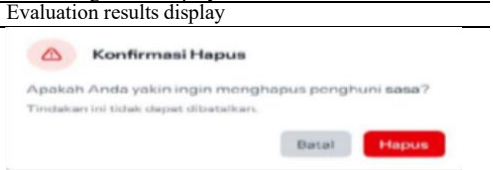
Initial view	Evaluation results display
	

Table 3: Comparison before and after evaluation of export pages to PDF

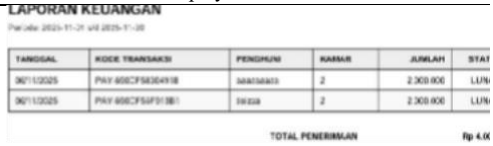
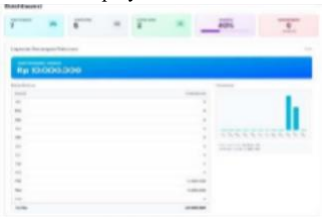
Initial view	Evaluation results display
	

Table 4: Comparison before and after evaluation of the dashboard summary view page

Initial view	Evaluation results display
	

4.8. Development

The figure illustrates the technology architecture used in developing the e-booking system for boarding houses. This system is built using the Laravel framework, which consists of three main components: Model, View, and Controller (MVC). The Model component (Eloquent ORM) serves as a link between the application and the PostgreSQL database, managing and retrieving data as requested by the system. The Controller (Livewire) acts as the application logic controller, managing the data flow between the model and the view, while the View (Blade and Tailwind CSS) serves to display the user interface with a responsive and interactive design. These three components interact with each other within the Laravel framework, creating an efficient workflow between data storage, logic processing, and displaying the view to the user.

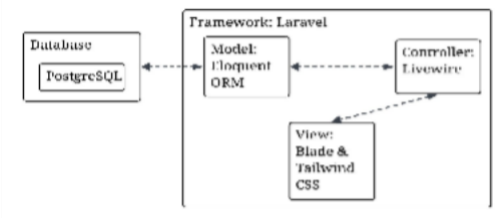


Fig. 56: Model View Controller system flow

Fig 57: Code snippet of the getHuman() function

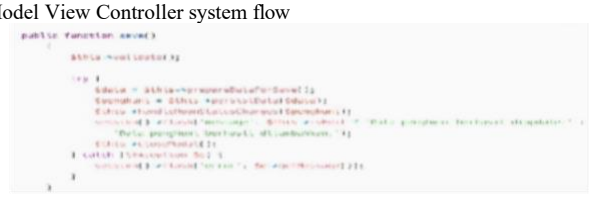


Fig. 58: Code snippet for the save() function

Fig. 59: Code snippet of the edit() function

Fig. 60: Code snippet of the delete() function

Fig. 61: Code snippet of the verify() function

Fig. 62: Code snippet of the submit() function

Fig. 63: Code snippet of the getRooms() function

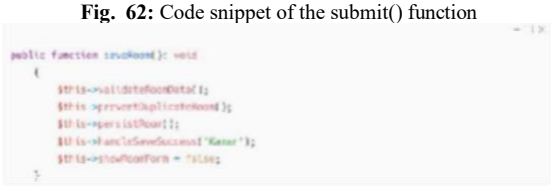


Fig. 64: Code snippet of the saveRoom() function

Fig. 65. Code snippet of the editRoom() function

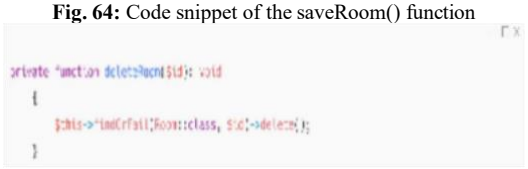


Fig. 66: Code snippet of the deleteRoom() function

Fig. 67: Code snippet of the getPembayaranQuery() function

Fig. 68: Code snippet of the save() function

Fig. 69: Code snippet of the edit() function

Fig. 70: Code snippet of the delete() function

Fig. 71: Code snippet of the verify() function

Fig. 72: Code snippet of the submit() function

Fig. 73: Code snippet of the baseQuery() function

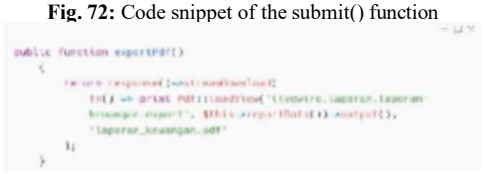


Fig. 74: Code snippet of the exportPDF() function

```php
private function prepareViewData(): array
    {
        $totalKamar = $this->getTotalKamar();
        $kamarTerisi = $this->getKamarTerisi();
        $laporan = $this->getLaporanKeuanganTahunan();

        return [
            'totalPenghuni' => $this->getTotalPenghuni(),
            'totalKamar' => $totalKamar,
            'kamarTerisi' => $kamarTerisi,
            'okupansi' => $this->calculateOkupansi($totalKamar, $kamarTerisi),
            'telatBayar' => $this->getTelatBayar(),

            'dataBulan' => $laporan['data'],
            'totalTahunan' => $laporan['total'],
        ];
    }
```

Fig. 75: Code snippet of the prepareViewData() function

4.9. Testing

Testing is a crucial stage in system development to ensure that the application being built meets both functional and non-functional requirements. The testing process is carried out to detect errors (bugs), ensure system stability, improve security, and evaluate application performance and responsiveness. In this study, testing was conducted using the Whitebox and Blackbox approaches. Whitebox testing emphasizes examining internal logic, program flow, and code coverage, while Blackbox testing focuses on application functionality from a user perspective without looking at the internal structure.

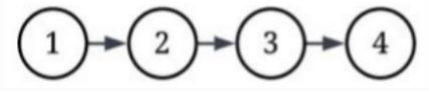


Fig. 76: Flowgraph getHuman()

Based on the whitebox testing analysis on the getPenghunis() function using the Basis Path method, the flowgraph in Figure X has 4 nodes and 3 edges, resulting in a Cyclomatic Complexity (CC) value = 1.

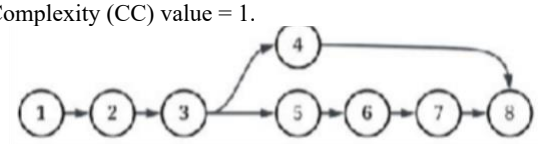


Fig. 77: Flowgraph save()

Based on the whitebox testing analysis on the save() function using the Basis Path method, the flowgraph in Figure X has 8 nodes and 8 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

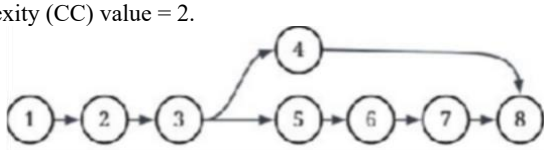


Fig. 78: Flowgraph edit()

Based on the whitebox testing analysis on the edit() function using the Basis Path method, the flowgraph in Figure X has 8 nodes and 8 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

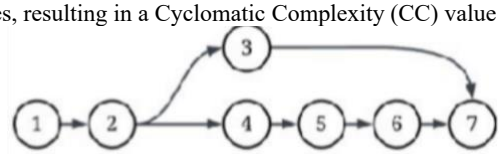


Fig. 79: Flowgraph delete()

Based on the whitebox testing analysis on the delete() function using the Basis Path method, the flowgraph in Figure X has 7 nodes and 7 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

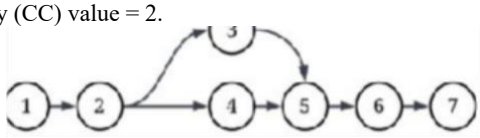


Fig. 80: Flowgraph verify()

Based on the whitebox testing analysis on the verify() function using the Basis Path method, the flowgraph in Figure X has 7 nodes and 7 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

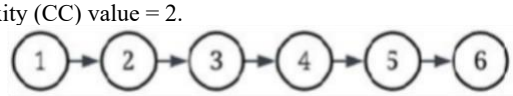


Fig. 81: Flowgraph submit()

Based on the whitebox testing analysis on the submit() function using the Basis Path method, the flowgraph in Figure X has 6 nodes and 5 edges, resulting in a Cyclomatic Complexity (CC) value = 1.

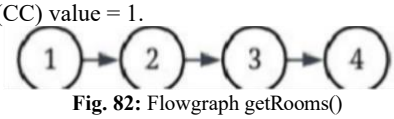


Fig. 82: Flowgraph getRooms()

Based on the whitebox testing analysis on the getPenghunis() function using the Basis Path method, the flowgraph in Figure X has 4 nodes and 3 edges, resulting in a Cyclomatic Complexity (CC) value = 1.

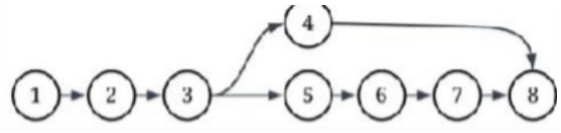


Fig. 83: Flowgraph saveRoom()

Based on the whitebox testing analysis on the saveRoom() function using the Basis Path method, the flowgraph in Figure X has 8 nodes and 8 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

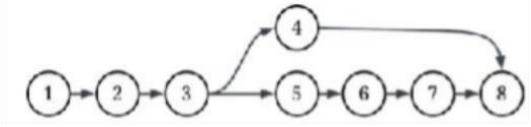


Fig. 84: Flowgraph editRoom()

Based on the whitebox testing analysis on the edit() function using the Basis Path method, the flowgraph in Figure X has 8 nodes and 8 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

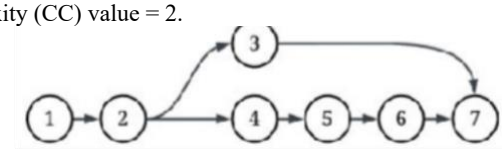


Fig. 85: Flowgraph delete()

Based on the whitebox testing analysis on the delete() function using the Basis Path method, the flowgraph in Figure X has 7 nodes and 7 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

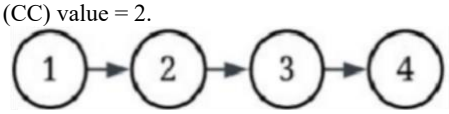


Fig. 86: Flowgraph getPembayaranQuery()

Based on the whitebox testing analysis on the getPembayaranQuery() function using the Basis Path method, the flowgraph in Figure X has 4 nodes and 3 edges, resulting in a Cyclomatic Complexity (CC) value = 1.

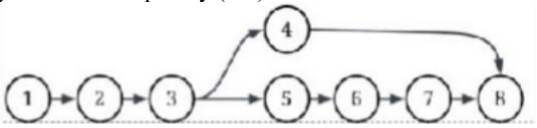


Fig. 87: Flowgraph save()

Based on the whitebox testing analysis on the save() function using the Basis Path method, the flowgraph in Figure X has 8 nodes and 8 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

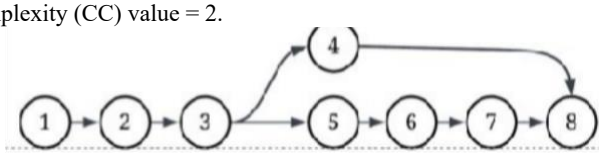


Fig. 88: Flowgraph edit()

Based on the whitebox testing analysis on the edit() function using the Basis Path method, the flowgraph in Figure X has 8 nodes and 8 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

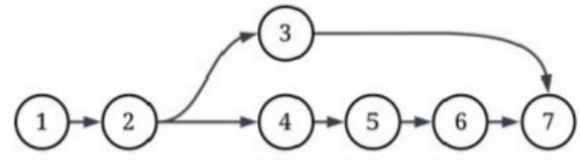


Fig. 89: Flowgraph delete()

Based on the whitebox testing analysis on the delete() function using the Basis Path method, the flowgraph in Figure X has 7 nodes and 7 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

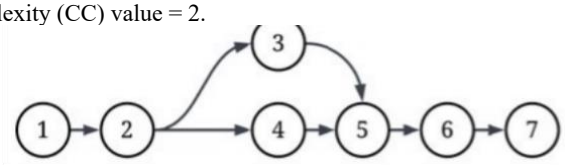


Fig. 90: Flowgraph verify()

Based on the whitebox testing analysis on the verify() function using the Basis Path method, the flowgraph in Figure X has 7 nodes and 7 edges, resulting in a Cyclomatic Complexity (CC) value = 2.

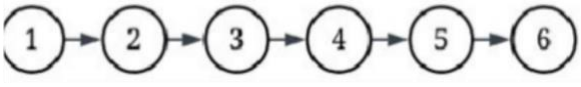


Fig. 91: Flowgraph submit()

Based on the whitebox testing analysis on the submit() function using the Basis Path method, the flowgraph in Figure X has 6 nodes and 5 edges, resulting in a Cyclomatic Complexity (CC) value = 1.

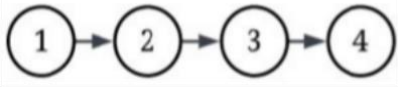


Fig. 92: Flowgraph baseQuery()

Based on the whitebox testing analysis on the baseQuery() function using the Basis Path method, the flowgraph in Figure X has 4 nodes and 3 edges, resulting in a Cyclomatic Complexity (CC) value = 1.

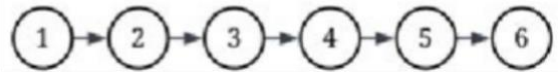


Fig.93: Flowgraph exportPdf()

Based on the whitebox testing analysis on the exportPdf() function using the Basis Path method, the flowgraph in Figure X has 4 nodes and 3 edges, resulting in a Cyclomatic Complexity (CC) value = 1.

The blackbox testing results on the D'Rosse Boarding House management system show that all modules function as expected and meet user needs. The Resident Management Module successfully displays, filters, adds, edits, and deletes resident data with proper input validation, including status and room verification. The Prospective Resident Registration Module is able to process online forms with complete validation for name, email, mobile number, address, and ID card files. The Room Management Module ensures the process of adding, editing, and deleting rooms runs safely, preventing duplication, and rejecting actions when the room is occupied. The Payment Module is fully functional from recording, editing, deleting, to payment verification, including validation of the nominal amount, date, and proof of payment files. The Financial Report Module displays transactions according to filters, is able to export PDFs, and handles empty data conditions. The Dashboard Module displays a real-time summary of information, including the number of residents, rooms, occupancy levels, and late payments. The UAT evaluation shows that the system improves operational efficiency, reduces manual errors, provides accurate information, simplifies verification, and supports fast and precise decision-making, resulting in a better user experience compared to previous methods.

4.10. Discussion

Previously, boarding house data management was done manually using books and Excel, which caused various problems such as errors in recording resident, room, and payment data, difficulty monitoring room and occupancy status, and inefficient financial reporting processes. To address these issues, a web-based boarding house management system was developed using a prototyping method so that feature evaluation and refinement could directly involve users. This system includes Resident Management for registration, editing, and verification with structured data validation; Room Management for digital recording and monitoring of room status; a Payment Module that handles recording, uploading evidence, and real-time verification; Financial Reports that automatically present a summary of monthly and annual income; and a Dashboard that displays total residents, occupied rooms, and late payments.

Implementation using the Laravel framework with an MVC architecture provides modularity, maintainability, and stability, demonstrated by core function testing with low Cyclomatic Complexity (1–6). Thus, the system successfully integrates previously scattered data, improves operational efficiency, minimizes recording errors, and supports more accurate and responsive decision-making to user needs.

5. Conclusion

Based on the results of needs analysis, design, implementation, and testing, the developed web-based boarding house management system successfully addressed the main issues previously experienced by the D'Rosse Boarding House Owner. This system integrates room management, occupancy, transactions, and reporting in one easy-to-use platform.with support for automation features, data validation, and responsive displays that improve daily operational efficiency. Design using the prototyping method allows for feature refinement through iterative evaluation, while the MVC architecture in Laravel separates logic, displays, and data processing for more stable integration between modules. As a result, managing residents, room status, payments, and financial reports is faster, more accurate, and more structured, reducing administrative burdens and the risk of input errors, while providing an informative dashboard that simplifies monitoring and decision-making, thus improving the overall system's efficiency, accuracy, and reliability of data in daily boarding house management.

Based on the results of this study, recommendations are for boarding house owners and managers to consistently utilize the system to ensure all operational data is properly recorded and minimize the risk of information loss. Regular use of the dashboard, financial reports, and room status features can facilitate faster and more accurate decision-making. For further development, this research could be expanded by adding automatic notification features via WhatsApp or email, digital payment integration, and a boarding house inventory management module. Furthermore, further research could consider a more in-depth user experience analysis and the development of a mobile application to enhance system accessibility and flexibility.

References

[1] Ramdany, SW, Kaidar, SA, Aguchino, B., Putri, CAA, & Anggie, R. (2024). Application of UML class diagrams in designing web-based library information systems. Journal of Industrial and Engineering System,
[2] 5(1).https://ejurnal.ubharajaya.ac.id/index.php/JIES/article/view/2275/1655
[3] Narulita, S., Nugroho, A., & Abdillah, MZ (2024). Unified Modeling Language (UML) diagram for designing research and community service management information systems (SIMLITABMAS). BRIDGE: Journal of Information Systems and Telecommunications Publication, 2(3), 244–256.https://journal.aptii.or.id/index.php/Bridge/article/view/174/281
[4] Hasridayyana, I., Ahmad, L., & Junaidi, R. (2023). Boarding house management information system using content management system method in Banda Aceh City. Computer Systems Journal (SISKOM), 3(2), 84–94.https://journal.stmiki.ac.id/index.php/siskom/article/view/797
[5] Suminten, S., Sintawati, & Indrarti. (2023). Designing a boarding house rental information system through a web-based application.
[6] Teknika, 17(2).https://jurnal.polsri.ac.id/index.php/teknika/article/view/8135
[7] Rahmatya, MD, Simangunsong, DES, & Wicaksono, MF (2022). e-Kos as a boarding house management information system at Mazasi's House. Journal of Technology and Information (JATI), 12(2), 176–190.https://doi.org/10.34010/jati.v12i2.8027

[8] Ghanghro, S.A., Sawand, M.A., Channa, W.A., Kashif, U.A., Tunio, M.H., Kumar, K., & Nooruddin, N. (2021). Comparative analysis of software process models in software development. International Journal of Advanced Trends in Computers Science
[9] an Engineering, 10(3).https://doi.org/10.30534/ijatcse/2021/1521032021
[10] Ulfa, M., Suryayusra, S., & Hardini, S. (2020). Application of model view controller (MVC) for designing the Ruang Buku Indonesia system. CESS (Journal of Computer Engineering System and Science, 5(1),
[11] 53.https://jurnal.unimed.ac.id/2012/index.php/cess/article/view/15403
[12] Praniffa, AC, Syahri, A., Sandes, F., Fariha, U., & Giansyah, QA (2023). Black box and white box testing of a web-based parking information system. Journal of Testing and Implementation of Information Systems, 1(1), 1–16.https://www.journal.al- matani.com/index.php/jtisi/article/view/321
[13] Fitriah, & Tining Haryanti. (2022). Designing a boarding house rental website information system (E-Kos) studies c a s e Surabaya. SinarFe7.https://journal.fortei7.org/index.php/sinarFe7/article/view/370
[14] Arimbi, YD, Kartinah, D., & Della, ANW (nd). Design of Malika Women's Boarding House Information System Based on Laravel & MySQL. Multidisciplinary Scientific Journal.https://www.semanticscholar.org/paper/RANCANGAN-SISTEM- INFORMATION-KOST-PUTRI-MALIKA-DAN-Arimbi-Kartinah/f312e225f610e78a9dc366f07cf213bbbb2ad4f1