Contents lists available at JESA

Journal of Engineering and Science Application

journal homepage: https://jesa.aks.or.id/index.php/jesa/index

Web-Based Point-of-Sale System with Inventory and Membership Management for KODAGAMA

Ridwan Cahya Gumilar, Ananda Syafira, Aditia Putra, Wasis Haryono

Informatics Engineering Study Program, Faculty of Computer Science, Universitas Pamulang, South Tangerang, Indonesia

Article Info

Keywords:

Black-Box Testing
Cooperative Retail
Inventory Management
Membership
Point Of Sale
White-Box Testing

ABSTRACT

The KODAGAMA cooperative retail unit previously maintained inventory records separately from sales, requiring staff to reconcile products sold with stock balances. This study presents a web-based point-of-sale system that integrates product management, sales transactions, inventory, membership benefits, and sales reporting within a shared application. Development followed the Waterfall stages of requirements analysis, design, implementation, and verification, with requirements identified through observation, interviews, and review of existing records. The Laravel application records invoices and line items, supports cash and noncash payment methods, and associates eligible purchases with member points and discounts. Evaluation combined documented black-box test cases from the work-placement report with manual white-box tracing of selected source paths. Of 172 black-box cases, 170 were recorded as meeting the expected outputs and two did not, both concerning options on an earlier sign-in interface. All 15 selected manual traces matched their expected outcomes for transaction processing, payment confirmation, member points, and reporting. These traces were simulations of code paths rather than execution in the application or database. Source inspection identified a risk in noncash confirmation because payment status and stock updates were not enclosed in one explicit database transaction. Further verification should examine repeat confirmations, stock changes before settlement, and interrupted writes using paired invoice and inventory records. The findings describe the documented functional behavior and selected implementation risks of the system; improvements in operational time and stock accuracy were not measured.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Ananda Syafira
Program Studi Teknik Informatika, Fakultas Ilmu Komputer
Universitas Pamulang
Tangerang Selatan, Indonesia
Email: anandasyafira878@gmail.com

1. INTRODUCTION

The retail unit of KODAGAMA, a consumer cooperative in Jakarta, maintained spreadsheet inventory records separately from sales. Staff had to reconcile quantities sold against stock balances; a payment record did not identify the products sold or the quantity of each item. The cooperative also needed a way to associate eligible purchases with member benefits without making membership compulsory at

checkout. Retail research has documented discrepancies between recorded and physical stock and investigated tools that make such discrepancies easier to detect [1], [2]. These studies explain why traceable records matter, although the frequency of inventory discrepancies at KODAGAMA was not measured.

Previous web systems have consolidated product, sales, and inventory data for stationery retailers [3], [4] and connected cashier functions to reporting in coffee and general retail settings [5], [6]. Other applications cover pharmacy transactions [7], link perfume sales to inventory monitoring [8], and separate cashier, administrator, and owner tasks [9]. These studies establish how a shared application can organize retail records. The specific problem at KODAGAMA is determining when the same sale should affect stock and member benefits when a cash payment is settled at checkout or a noncash payment is confirmed after an invoice has been issued.

This study presents a web-based point-of-sale (POS) application for the cooperative and evaluates its functional behavior and selected code paths. The contribution is a concrete account of how invoice status, inventory deduction, member benefits, and sales reporting are connected, together with evidence of which behaviors were recorded and which still require execution-based verification. The scope is one retail unit. No operational performance comparison or causal claim about the effectiveness of the application is made.

The evaluation addresses three questions. First, how are the product, invoice, inventory, and membership records connected during a sale? Second, which expected behaviors are supported by the documented functional cases and selected code traces? Third, which conditions still need verification when payment confirmation occurs after invoice creation? These questions keep the analysis focused on the cooperative's transaction workflow. A correctly displayed total is useful evidence about calculation, but it does not by itself establish that the corresponding inventory and membership updates were completed.

The practical contribution is the description of an integrated workflow in which a sale can be followed from product selection to its payment state and reporting. The evaluation contribution is the separation of recorded interface outcomes from manually inferred code outcomes. This distinction makes the reported results interpretable without treating a screen capture, a test verdict, and an executed database check as equivalent evidence.

2. RESEARCH METHOD

2.1. Case setting and development

The case is the retail unit of Koperasi Konsumen Juanda Tiga Lima (KODAGAMA). The work-placement report documents observation, interviews, and review of existing records to identify requirements for products, sales, stock, membership, and reporting. The evaluation uses the report's black-box records, interface captures, and the project source at commit 97843f3. Keeping these sources distinct allows the reader to distinguish what was recorded through the interface from what was inferred from code, consistent with guidance for software engineering case studies [10].

The project followed the Waterfall stages of requirements analysis, design, implementation, and verification. Staff workflows and data dependencies were mapped before the corresponding modules were implemented. Royce's discussion of late-stage testing risks informed the use of review points before verification [11]. Unified Modeling Language conventions were used to describe roles and the order of the main transaction actions [12].

The requirements were organized around the records used during daily retail work. Product identification and classification support selection at checkout; price and quantity determine the sale amount; payment method and status determine the settlement workflow; and an optional member selection connects a purchase to benefits. Reporting brings these records together for a chosen period. This organization provides a basis for reviewing integration: each module has a local responsibility, while the invoice links the actions that belong to the same sale.

Within the Waterfall sequence, verification therefore needs to revisit the relationships established during analysis and design. A product search can be checked as an individual function, but a completed sale also requires a consistent relationship between its selected products, recorded quantities, amount due, and payment state. The study examines both levels through the available test records. It does not assume that completing a development stage proves that every cross-module condition has been verified.

2.2. System design and settlement rule

The Laravel application stores the sale as an invoice and line items containing product identifiers, quantities, and unit prices. Each invoice records its payment method and status. Product records hold the current balance. If a member is selected, an optional identifier links the sale to the membership record; sales can also be made without a member. The implementation stores general discounts, level-based discounts, and discounts obtained by redeeming points as separate transaction amounts. Eligible spending accrues points and updates the member's level under the configured rules. Laravel provides transaction facilities for committing or rolling back related writes [13].

Figure 1 shows the intended relationship between settlement and stock. Cash is settled at checkout when the amount tendered is sufficient. An invoice for a bank transfer or a payment under the Indonesian Quick Response Code standard (QRIS) begins as pending [14]. The operator verifies receipt and confirms the transaction; the application does not automatically receive a settlement notification in the reviewed implementation. Stock is checked at invoice creation, but a pending invoice does not reserve units. The balance may therefore change before the operator confirms payment.

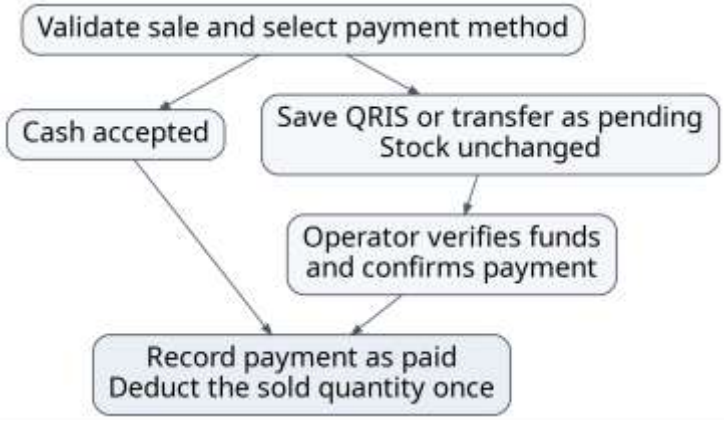


Figure 1. Intended relationship between payment settlement and stock deduction.

For a single isolated sale of quantity q and opening balance S_0, a pending invoice should leave stock at S_0. First settlement should leave stock at $S_0 - q$; a rejected operation or a repeat confirmation should not deduct stock again. Related settlement and inventory writes should complete together or roll back together [15], [16]. This rule defines the expected behavior against which the relevant paths are reviewed.

The application can be understood at three levels: the interface where staff select products and enter payment information, the application logic that validates and processes those inputs, and the stored records that retain the invoice and its associated data. The interface presents a proposed sale; the application logic decides whether the request is acceptable; the stored records provide the basis for later retrieval and reporting. Reviewing these levels separately helps locate a discrepancy. An incorrect total may concern calculation, whereas a paid invoice with an unchanged balance concerns the persistence of related updates.

Membership is optional within this transaction model. A sale without a member must still retain its products, quantities, total, and payment state. When a member is selected, the purchase also needs a consistent link to the benefit calculation. This adds a dependency to checkout rather than replacing the ordinary sale record. Consequently, the absence of a member should not be interpreted as an incomplete transaction, and a valid member selection should not bypass the ordinary stock and payment checks.

2.3. Functional testing and manual code tracing

Black-box testing compares visible or otherwise observable outputs with the expected result without depending on the internal code structure. The report documents cases for normal behavior, rejection of invalid input, and boundary conditions. Test-design guidance distinguishes equivalence partitioning and boundary-value analysis from techniques based on code structure [17], [18]. Quantity values immediately below, at, and above available stock are useful because the application changes its decision at that boundary [19], [20]. Related sales and inventory studies also illustrate how these techniques can turn input rules into representative cases [21], [22].

White-box evaluation in this study is a manual trace of 15 selected paths in the source. The trace identifies each decision, the case input, and the outcome inferred from the path. A matching trace includes an expected rejection, such as insufficient cash or an invalid point increment. The exercise does not run the application or its database. The documented black-box case verdicts and the manual trace outcomes are therefore recorded separately. No branch coverage or combined pass rate is calculated.

2.4. Evidence interpretation and evaluation criteria

The unit of functional evaluation is an individual test case: an input or action, an expected output, and its recorded outcome. Cases are grouped by system area to summarize the documented results. The unit of white-box evaluation is a selected code path, with the relevant decision and expected consequence traced manually. Because these units answer different questions, their totals are not combined. Fifteen matching traces cannot be added to the functional cases to produce an overall application success rate.

Three criteria guide interpretation. Calculation consistency concerns whether quantities, discounts, and tendered amounts produce the expected payable amount and change. State consistency concerns whether an invoice's payment status agrees with the inventory effects expected at that stage. Traceability concerns whether the products, member selection, and payment information can be related to the same invoice. These criteria organize the discussion of the supplied evidence; they are not additional test results or a claim of complete coverage.

For boundary cases, the expected result must include both acceptance and rejection behavior. A request above available stock should be rejected even if an ordinary sale succeeds. Likewise, an invalid point redemption should not produce a discount simply because a valid redemption works. The report's verdicts and the selected traces are assessed against their own expected outcomes. Where the underlying record lacks a detailed observation, the interpretation remains limited to the recorded verdict rather than reconstructing an unobserved application response.

3. RESULTS AND DISCUSSION

3.1. Black-box test summary

Table 1 summarizes the case verdicts transcribed from the work-placement report. Of 172 cases, 170 were recorded as meeting the expected output and two did not. Both mismatches concerned earlier sign-in options: "Remember me" and password recovery. The report states that password recovery was subsequently removed from the November interface. This change does not count as a passing retest. Some case records provide a verdict without detailed output or independent evidence, so the counts describe the original records rather than a verified pass rate for the deployed build.

Table 1. Documented black-box verdicts by system area

System area	Cases	Matched	Did not match
Sign-in	15	13	2
Administrator dashboard	14	14	0
Membership	9	9	0
Product types	8	8	0
Categories	9	9	0
Administrator product management	14	14	0
Administrator transactions	15	15	0
Administrator sales reports	15	15	0
User management	9	9	0
Search	10	10	0
Cashier checkout and basket	22	22	0
Cashier product list	12	12	0
Cashier sales reports	20	20	0
Total	**172**	**170**	**2**

A captured cash transaction provides an example of visible behavior. The detail page shows one item each of instant noodles, coffee, and bottled water at IDR 3,000, IDR 5,000, and IDR 3,000, respectively, with an IDR 11,000 total and successful payment confirmation (Figure 2). A separate checkout capture at the same total shows IDR 9,000 cange for IDR 20,000 tendered. Neither capture contains paired opening and closing stock balances.



Figure 2. Transaction detail showing the total and successful cash payment confirmation.

3.2. Representative black-box test cases

Table 2 gives the inputs, expected behavior, and recorded outcomes for selected black-box cases. It includes both sign-in mismatches and cases from membership, the cashier product list, checkout, inventory, and reporting. “Matched” reproduces the original verdict. Where the report provides only a verdict or a brief observation, the table does not infer a more detailed result.

Table 2. Selected black-box test cases and recorded outcomes

Test case	Input or action	Expected result	Recorded result	Verdict
Incorrect password	Enter an incorrect password.	The application rejects sign-in.	Matched the expected outcome, as recorded.	Matched
Remember me	Select “Remember me” on the earlier interface.	The application applies the sign-in preference.	Outcome did not match; details were not recorded.	Did not match
Password recovery	Select the password recovery link on the earlier interface.	The recovery page opens.	Outcome did not match; details were not recorded.	Did not match
Duplicate member ID	Enter a previously registered membership identifier (NIP).	The application rejects the duplicate.	Matched the expected outcome, as recorded.	Matched
Cashier product search	Enter “Indomie” in the product search field.	The application lists Indomie Goreng.	The test record confirms that the product was found.	Matched
Percentage discount	Apply a 10% discount at checkout.	The application reduces the amount due by 10%.	The recorded amount due changed as expected.	Matched
Cash change	For an IDR 15,000 sale, enter IDR 20,000 tendered.	The application calculates IDR 5,000 change.	The record reports correct change.	Matched
Insufficient cash	For an IDR 15,000 sale, enter IDR 10,000 tendered.	The application prevents processing or warns that the amount is insufficient.	The application displayed a validation response.	Matched
Payment confirmation	Select Confirm Payment.	The application updates the payment status.	Marked as matching; the final status value was not recorded.	Matched

Test case	Input or action	Expected result	Recorded result	Verdict
Stock deduction	Complete a sale of two units of Aqua.	Stock decreases by two units after the sale succeeds.	The record notes an automatic stock update; it does not give paired balances.	Matched
Insufficient stock	Request more units than are available.	The application prevents the requested quantity from being added.	The record reports that stock validation worked.	Matched
Sales report	Open the sales total for 16 Sep–15 Oct 2026.	The report totals paid transactions for the chosen period.	The displayed total was IDR 1,620,600.	Matched

The detailed records show what was observed or reported at the interface. For example, the change calculation and stock validation were recorded as working, whereas the stock deduction case does not report opening and closing balances. The displayed sales total is also an interface observation; the report does not provide an independent reconciliation of every underlying invoice for that case.

3.3. White-box trace results

All 15 selected traces matched their expected outputs in the manual simulation (Table 3). The transaction trace covered stock rejection, percentage and capped nominal discounts, insufficient cash, creation of a pending QRIS invoice, and confirmation behavior. For example, the trace for three items totaling IDR 18,000 applied a 10% discount and reduced the amount due to IDR 16,200; tendering IDR 20,000 produced IDR 3,800 change.

Table 3. Manually traced white-box paths

Trace IDs	Function	Branches traced	Trace outcome
WB01–WB05	Order creation	Stock and tender checks, percentage and capped nominal discounts, and a pending QRIS invoice.	5/5 match
WB06–WB07	Payment confirmation	A paid invoice stops before a second write; an eligible pending invoice updates status and stock.	2/2 match
WB08–WB12	Membership rules	Reject points below the minimum, outside the increment, or above the balance; apply a valid redemption and carry forward unused spending.	5/5 match
WB13–WB15	Report trends	Handle a zero prior value and calculate the direction and absolute change for a nonzero prior value.	3/3 match

The membership traces separate three rejection conditions for point redemption: a request below the minimum, a request outside the allowed increment, and a request exceeding the balance. A valid 20-point example used the active constant of IDR 100 per point in the reviewed source and produced IDR 2,000 off. A separate trace showed how a balance carried over from earlier purchases can yield one additional point and a level change. The report traces also checked the branch that handles a zero prior-period value before division.

3.4. Risks identified in confirmation logic

Source inspection showed that cash checkout groups invoice, line-item, and inventory writes inside a database transaction. The reviewed noncash confirmation handler instead updated the invoice and then deducted stock, without an explicit enclosing database transaction or a renewed stock check. If stock changes after invoice creation, the earlier stock check may no longer be sufficient. If an inventory write fails after the invoice has been marked paid, payment and inventory records may diverge. The guard for a previously paid invoice can stop an ordinary repeat after the first request completes, but two simultaneous requests could both read the pending state. These are risks inferred from the source, not incidents observed during field operation.

A follow-up change would verify that the invoice is still pending, check available stock, and update both records in one protected database transaction. Research on isolation explains why the result of concurrent access cannot be established from sequential execution alone [23]. Row-locking reads offer one implementation mechanism for protecting the required records [24]. The next checks should repeat confirmation sequentially and concurrently, sell remaining stock before a pending confirmation, interrupt a write between the status and stock updates, and submit two sales for the same last unit. Each check should retain invoice and stock records from the same run. These checks address risks found in the code and have no recorded execution results in this study.

3.5. Integration of transaction records

The main integration point is the invoice. Product management supplies the items available for selection, while each sale line records the selected product, quantity, and unit price. The invoice collects these lines under one payment method and status. Membership adds an optional relationship, and the reporting function selects transactions for a defined period. Table 4 summarizes the logical responsibilities of these records. It describes their role in interpreting the application rather than introducing additional physical database tables.

Table 4. Logical records and their role in transaction traceability

Record or output	Relationship in the workflow	Contribution to interpretation
Product	Referenced by a sale line	Identifies the item and its available stock.
Invoice	Groups sale lines and payment information	Connects the amount due to the payment method and status.
Sale line	Connects a product to an invoice	Retains quantity and unit price for the purchase.
Member	Optionally associated with a sale	Links an eligible purchase to points and discounts.
Sales report	Derived from transactions within a selected period	Presents the paid transactions included in the period.

This structure explains why module-level success should be interpreted together with the invoice state. A product list can display a stock value, and a transaction detail page can display a successful payment, without the two captures establishing the change caused by that particular invoice. The IDR 11,000 cash example demonstrates a visible transaction total and confirmation. To establish the inventory effect, the same run would also need its opening balance, quantity sold, and closing balance for each affected product. The present captures do not supply that paired comparison.

3.6. Interpretation of calculations and membership rules

The manually traced discount example provides a transparent arithmetic reference. Two items priced at IDR 5,000 and one item priced at IDR 8,000 produce an IDR 18,000 subtotal. A 10% discount is IDR 1,800, leaving IDR 16,200 payable. With IDR 20,000 tendered, the expected change is IDR 3,800. These values explain the trace outcome; they are distinct from the captured IDR 11,000 sale. Keeping the examples separate avoids attributing one calculation to a different transaction.

Percentage and nominal discounts also need different boundary interpretations. The reviewed percentage expression limits the percentage used in calculation to 100, while the nominal expression limits the deduction to the subtotal. Those limits address how much can be deducted within that calculation. They do not establish the behavior of every combination of general, membership-level, and point-redemption discounts. The report describes separate amounts for these discount components, so combined cases would need an explicit expected order of calculation and a check of the final amount recorded.

For membership, the 20-point trace produces an IDR 2,000 discount using the reviewed value of IDR 100 per point. The rejection traces examine requests below the minimum, outside the permitted increment, and above the available balance. These are distinct business conditions: an available balance does not make an invalid increment acceptable. A complete execution check would also compare the member balance before and after the corresponding sale and relate the change to its invoice. The matching manual traces establish the selected decision outcomes, while the timing and persistence of the updates require evidence from an actual run.

3.7. Implications for operation and follow-up verification

For daily operation, the distinction between an issued invoice and a paid sale is central. An invoice documents a proposed purchase, while payment confirmation determines whether the sale has reached the settled state described in the workflow. Staff reviewing transaction history should therefore interpret the status together with the method and total. Selecting QRIS or transfer identifies the payment method; in the reviewed implementation, the operator still verifies receipt. The existence of that selection does not demonstrate an automatic connection to a payment provider.

Reporting has a corresponding dependency. The expected sales report includes paid transactions within the selected period, so its total should be compared with the same set of invoices and the same date boundaries. Pending invoices should not be treated as settled revenue in this comparison. A displayed total is useful for checking the interface, but independent reconciliation would require listing the included invoice identifiers and summing their final amounts. The supplied report record contains a displayed figure, rather than that complete reconciliation.

The first follow-up priority is the confirmation path because one action affects both payment state and inventory. A controlled check should record the starting invoice status and product balance, confirm the payment, and inspect both resulting records. Repeating the action should leave the first completed deduction unchanged. A separate concurrent check is needed because a sequential repeat does not reproduce two requests reading the same pending state. Interrupted-write checks should determine whether the related changes are either completed together or rolled back together, consistent with the intended transaction boundary [15], [16].

The next priority is continuity between the tested version and the version used by staff. Removing an earlier password-recovery option changes the interface but does not retrospectively change its recorded verdict. A later verification record should identify the build and the behavior retested. The same principle applies to confirmation and membership changes: a corrected implementation needs its own execution evidence. This approach preserves the usefulness of the existing black-box results while making subsequent corrections distinguishable from the original evaluation.

3.8. Limits of the evaluation

The case covers one cooperative and one source snapshot. The functional records do not verify later deployments. Manual traces do not establish persisted changes, code coverage, or concurrent behavior. Usability, recoverability, and performance require separate criteria [25]. Changes in staff time and stock accuracy were not measured.

The available cases also constrain interpretation of coverage. Counts by module indicate where cases were documented, but they do not measure the proportion of possible inputs, branches, or user sequences examined. The cashier checkout area contains more recorded cases than membership, for example, without proving that one area is more reliable. The study supports a bounded account of the selected functions and paths. Broader conclusions would require defined coverage targets and retained execution evidence for those targets.

4. CONCLUSION

The KODAGAMA POS links stock and member benefits to the sale and its payment state. The report records 170 of 172 black-box cases as matching expectations, with two sign-in mismatches. All 15 selected white-box traces matched in manual simulation. The next verification should compare paired invoice and stock records on the target build after duplicate requests, competing sales, and interrupted writes.

REFERENCES

1] N. DeHoratius and A. Raman, “Inventory record inaccuracy: An empirical analysis,” Management Science, vol. 54, no. 4, pp. 627–641, 2008, doi: 10.1287/mnsc.1070.0789.

[2] B. C. Hardgrave, J. A. Aloysius, and S. Goyal, “RFID-enabled visibility and retail inventory record inaccuracy: Experiments in the field,” Production and Operations Management, vol. 22, no. 4, pp. 843–856, 2013, doi: 10.1111/poms.12010.

[3] M. R. Hasibuan, F. Gultom, and M. B. K. Nasution, “PHP-based stationery sales information system at Ufaira Printing Sigambal, Labuhanbatu,” Journal of Engineering and Science Application, vol. 3, no. 2, pp. 190–197, 2026, doi: 10.69693/jesa.v3i2.57.

[4] C. C. Utami, R. I. Desanti, and F. A. Halim, “The development of web-based sales and inventory system for a stationary store,” Ultima InfoSys, vol. 14, no. 2, pp. 52–57, 2023, doi: 10.31937/si.v14i2.3358.

[5] F. Azizah, L. Novyanti, N. O. Amri, A. Oktaviani, and R. Nurfalah, “Perancangan aplikasi Point of Sale Pandita Coffee berbasis web dengan kombinasi model SDLC Waterfall,” Jurnal Sistem Informasi Kaputama, vol. 6, no. 1, pp. 69–74, 2022, doi: 10.59697/jsik.v6i1.184.

[6] S. Suprianto, M. Fadlan, and D. Prayogi, “Perancangan aplikasi Point of Sale berbasis web pada Toko Project Salfa Tarakan,” Sebatik, vol. 25, no. 2, pp. 624–631, 2021, doi: 10.46984/sebatik.v25i2.1519.

[7] I. D. G. A. Pandawana, B. P. P. Putra, I. P. G. Budayasa, I. D. M. A. B. Joni, and P. A. Permata Sari, “Pengembangan Point of Sale berbasis web untuk pengelolaan transaksi apotek,” Jurnal Ilmiah Teknik Universitas Mahasaraswati Denpasar, vol. 6, no. 1, pp. 59–64, 2026, doi: 10.36733/jitumas.v6i1.14027.

[8] H. Aliazki Pratama and B. Yulisa Geni, “Web-based POS system to support perfume sales transactions using the Waterfall model,”

Journal of Information Systems and Technology Research, vol. 5, no. 3, pp. 343–355, 2026, doi: 10.55537/jistr.v5i3.1535.
[9] A. A. Amalia, E. Supriyati, and T. Listyorini, "Design of a web-based Point of Sale system for transaction recording at Sumber Rejeki Store," SAGA: Journal of Technology and Information System, vol. 3, no. 3, 2025, doi: 10.58905/saga.v2025i3.547.
[10] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical Software Engineering, vol. 14, pp. 131–164, 2009, doi: 10.1007/s10664-008-9102-8.
[11] W. W. Royce, "Managing the development of large software systems," in Proc. IEEE WESCON, 1970, pp. 1–9. [Online]. Available: https://cse.msu.edu/~cse435/Homework/HW3/royce1970.pdf
[12] Object Management Group, OMG Unified Modeling Language, ver. 2.5.1, 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/
[13] Laravel, "Database: Getting started," Laravel 12.x Documentation. [Online]. Available: https://laravel.com/docs/12.x/database
[14] Bank Indonesia, "Quick Response Code Indonesian Standard (QRIS)." [Online]. Available: https://www.bi.go.id/id/fungsi-utama/sistem-pembayaran/ritel/kanal-layanan/QRIS/default.aspx
[15] J. Gray, "The transaction concept: Virtues and limitations," in Proc. 7th Int. Conf. Very Large Data Bases, 1981, pp. 144–154. [Online]. Available: https://www.vldb.org/conf/1981/P144.PDF
[16] Oracle, "START TRANSACTION, COMMIT, and ROLLBACK statements," MySQL 8.4 Reference Manual. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/commit.html
[17] ISTQB, Certified Tester Foundation Level Syllabus, ver. 4.0.1, 2024. [Online]. Available: https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf
[18] Software and systems engineering—Software testing—Part 4: Test techniques, ISO/IEC/IEEE 29119-4:2021, 2021.
[19] F. Dobslaw, R. Feldt, and F. Gomes de Oliveira Neto, "Automated black-box boundary value detection," PeerJ Computer Science, vol. 9, Art. no. e1625, 2023, doi: 10.7717/peerj-cs.1625.
[20] A. Z. Rafif et al., "Functional testing of web-based input form using boundary value analysis technique: A case study on TemanTernak," Jurnal Teknologi dan Open Source, vol. 8, no. 2, pp. 1064–1070, 2025, doi: 10.36378/jtos.v8i2.5084.
[21] A. Samdono, A. P. Sari, and F. P. Aditiawan, "Pengujian Black Box pada sistem informasi stok dan penjualan berbasis website menggunakan metode equivalence partitioning: Studi kasus CV. Algani Karya Mandiri," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 8, no. 1, 2024, doi: 10.36040/jati.v8i1.8893.
[22] M. Sutomo and F. E. B. Simamora, "Pengujian black box aplikasi inventaris dengan Boundary Value Analysis di PT Duo Roda Motor," Device: Journal of Information System, Computer Science and Information Technology, vol. 5, no. 1, pp. 202–216, 2024, doi: 10.46576/device.v5i1.4557.
[23] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," in Proc. ACM SIGMOD Int. Conf. Management of Data, 1995, pp. 1–10, doi: 10.1145/223784.223785.
[24] Oracle, "Locking reads," MySQL 8.4 Reference Manual. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/innodb-locking-reads.html
[25] Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—Product quality model, ISO/IEC 25010:2023, 2023.