The Application of Enterprise Resource Planning on a Single Board Computer

1st Eldwin Pradipta
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
eldwinpradipta670@gmail.com

2nd Rinaldi
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
rinaldi@informatika.org

***Abstract*—Small environmental testing laboratories may rely on disconnected spreadsheets to coordinate commercial and testing workflows, creating difficulties in data consistency and process traceability. This paper reports an empirical design-and-deployment case study of Senling, a custom enterprise resource planning (ERP) system developed for Sentral Sistem Laboratory (SSLab) in Indonesia and hosted on an Orange Pi 5 Pro. Senling uses a modular-monolith architecture implemented with Laravel, Filament, and PostgreSQL as a four-container Docker Compose stack. Its requirements record evolved from 31 designed use cases into an as-built inventory of 39, while the tested and operationally used subsets are reported separately. All 76 automated test cases, organized into 21 test units, passed, and part of the sales flow entered operational use during the study. Five internal users reported mean scores of 3.98 out of 4 for usability and 4.00 out of 4 for perceived responsiveness; these results are treated only as descriptive perceptions because of the sample size. During approximately 39 minutes of light real use involving quotation preparation, the 8-GB device recorded 7.0% mean CPU utilization, 13.9% peak CPU utilization, and 2.7 GB mean memory use; deployment reproducibility was demonstrated only through a clean build on WSL x86-64. The results support initial feasibility under the observed development and sandbox conditions but do not establish production capacity, comparative hardware performance, or generalizability beyond this case.**

***Index Terms*—enterprise resource planning, single-board computer, modular monolith, on-premise deployment, testing laboratory**

I. Introduction

Enterprise resource planning (ERP) systems integrate an organization's business processes around a shared database, replacing fragmented, per-department record keeping [1]. While ERP adoption is well established in large enterprises, small and medium-sized organizations face a persistent gap: their processes are often too domain-specific for generic modules, yet their budgets rule out enterprise-class licensing and consulting.

Commercial environmental testing laboratories illustrate this gap. Their operational chain — quotation, sales order, sample collection and reception, test scheduling and execution, supervisory review, and certificate issuance — must satisfy the traceability demands of ISO/IEC 17025 accreditation [2]. In practice, laboratories at this scale frequently coordinate the entire chain through disconnected spreadsheets, which invites data inconsistency and makes traceability laborious. Laboratory information management systems (LIMS) address the analytical core of this work [3], but leave the surrounding commercial process (quotation, ordering, invoicing hand-off) outside their scope, while off-the-shelf ERP products such as Odoo cover the commercial process generically and require deep domain customization before they fit laboratory workflows [4].

Separately, commodity ARM single-board computers (SBCs) have become inexpensive, low-power alternatives to conventional servers [5], [6]. Whether such a device can host a custom, domain-specific ERP for a small organization — not merely run a generic ERP package as an installation exercise — is comparatively unexamined.

This paper reports an empirical design-and-deployment case study conducted at Sentral Sistem Laboratory (SSLab), an Indonesian environmental testing laboratory established in 2025 as part of a group that also operates a KAN-accredited calibration unit; SSLab itself is KAN-accredited for part of its test parameters. This context matters: a newly established laboratory on a limited budget, at the time of this study, built and ran Senling with a two-person development team rather than a dedicated IT department, yet faces the same traceability expectations from its accreditation body as a larger organization. The contribution is threefold. First, the paper documents how the operational requirements of a small testing laboratory were translated into a custom ERP, Senling, with explicit domain boundaries and a modular-monolith architecture. Second, it reports the design-and-deployment rationale for running Senling as a single deployable application on an ARM SBC, the Orange Pi 5 Pro, given this small development team and budget. Third, it provides bounded empirical evidence from functional testing, partial operational use, internal user perception, and whole-system resource monitoring, establishing initial feasibility under the observed light-load conditions rather than production capacity or hardware superiority.

II. Related Work

A. ERP for Small Organizations

Monk and Wagner frame ERP as the integration of functional areas over one shared data store [1]. For small organizations, the customization-versus-adoption trade-off is well documented: generic modules presume generic processes, and

the effort of bending them to a specialized domain can approach that of purpose-building [4]. In the case examined here, mapping the laboratory's sample-registration and certificate-issuance process onto Odoo's generic modules was judged to require domain-level customization comparable in effort to a from-scratch build [4].

Domain-specific systems for laboratories exist as LIMS [3], but these focus on sample analysis rather than the commercial chain (quotation, ordering, invoicing hand-off) that surrounds it. A testing laboratory of the kind studied here needs both halves connected, motivating a custom system spanning the commercial and analytical process rather than adopting a generic ERP or a LIMS alone.

B. Modular Monolith Versus Microservices

Splitting an application into independently deployable microservices trades operational simplicity for independent scalability, and is generally advised only once network-boundary coordination is repaid by workload heterogeneity across services [12]. For a small laboratory this cost is hard to justify: each service is another deployment unit, another failure mode, and another surface to monitor, all falling on the one or two people already responsible for the rest of the stack. A modular monolith keeps domain boundaries explicit in the schema and code while deploying as a single artifact, preserving the option to split later without paying that cost up front — making the choice contingent on team size and operating context as much as on expected load.

C. Single-Board Computers as Servers

SBCs have been studied as portable, low-cost servers. Arief et al. evaluated an SBC-based portable server for classroom activity [5]; Álvarez Ariza and Baez survey the role of SBCs in engineering education [6]. Putra and Sujana showed a load-balanced Raspberry Pi cluster can improve response time over a single node for static HTTP traffic [9]. Blem et al. found ARM/x86 performance-per-watt differences stem more from design targets than the instruction set itself [10], and Aroca and Gonçalves reported favorable ARM power efficiency in server workloads [11].

D. ERP on SBCs and Positioning

Reports of ERP on SBCs are mostly community installation guides, e.g., Odoo on a Raspberry Pi [7] and ERPNext on a Raspberry Pi 5 [8]; these confirm installation is possible but say little about sustained use by a real organization or resource behavior under real load. Taken together, the cited literature offers limited evidence connecting a domain-specific ERP design with observations from actual organizational use on SBC hardware. This study addresses that narrower intersection through a single empirical case, without attempting a comparative hardware evaluation or a general performance claim.

III. Case and System Design

A. Case and Requirements

SSLab's pre-existing process was manual: quotations, sample chains of custody, and schedules lived in separate spreadsheets, risking inconsistent records and making ISO/IEC 17025 traceability harder to demonstrate. Analysts and sales staff duplicated data entry across files, sample status was tracked informally through messaging channels, and reconstructing a sample's history for an audit required manually cross-referencing several spreadsheets. These symptoms, rather than any single catastrophic failure, motivated the case for an integrated system.

From an analysis of this condition, six functional requirements (FR) and six non-functional requirements (NFR) were formulated along the operational chain, summarized in Table I. FR-1 through FR-4 map onto the process stages in Fig. 1; FR-5 (inventory) supports testing indirectly; FR-6 (finance) is deliberately narrow, as discussed in Section III-D. The NFRs derive from accreditation demands (NFR-1), the small team and its users (NFR-2–4), and the target hardware (NFR-5–6). The requirements are stated generically for laboratories of this class; their instantiation is specific to SSLab.

TABLE I
Functional and non-functional requirements

ID	Requirement
FR-1	Quotation and sales order
FR-2	Sample collection and reception
FR-3	Test scheduling and execution
FR-4	Supervisory review and certification
FR-5	Inventory (reagents and equipment)
FR-6	Finance
NFR-1	Data integrity and consistency
NFR-2	Role-based access control
NFR-3	Usability
NFR-4	Responsiveness
NFR-5	Deployment portability
NFR-6	Frugal resource use

B. Business Process

The as-built process (Fig. 1) runs from quotation through sample collection, reception, testing, supervisory review, and certificate issuance, with payment handled outside Senling by a shared finance application. This flow-level view drove the use-case and screen decomposition described in Section IV.

Fig. 1. Level-0 as-built business process flow.

C. Architecture

Senling is a modular monolith: a single deployable application whose domain boundaries are kept explicit in the schema and code rather than enforced by network boundaries. Fig. 2 shows the deployment topology: a reverse proxy (Nginx) in front of the Laravel application, backed by PostgreSQL, all on one server. A monolith was preferred over microservices

because the system was built and is operated by a small development team rather than a dedicated IT department; one artifact, one database, and no inter-service network hops keep both operations and data-integrity reasoning simple.

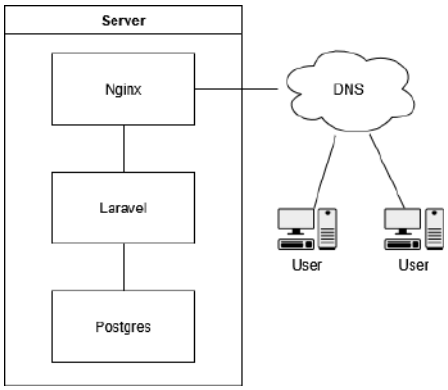

Fig. 2. Deployment topology on the single on-premise server.

D. Data Design

The database schema is organized into five domain clusters — Customer, Product Knowledge, Sales, Operations, and Finance — so a future split into services remains possible without being paid for up front [12]. Customer holds client and contact records; Product Knowledge holds the catalog of test parameters, packages, and reference methods; Sales covers quotations and sales orders; Operations covers sample collection, scheduling, worksheets, and supervisory review; Finance, as described below, is reduced to a single integration point.

Table II lists representative entities per cluster, illustrating the granularity of the domain boundaries — e.g., separating the test-parameter catalog (Product Knowledge) from the quotations that reference it (Sales) keeps catalog changes from touching order history.

TABLE II
DOMAIN CLUSTERS AND REPRESENTATIVE ENTITIES

Cluster	Representative Entities
Customer	Customers, contacts, addresses
Product Knowledge	Test parameters, packages, regulations, reference methods, reagents, containers
Sales	Quotations, quotation details, quotation-parameter links
Operations	Sales orders, order details, sample collection, samples, sample-parameter results
Finance	Sales-order integration point only (Section III-D)

Statuses of core entities (sales order, sample, test, certificate) are modeled as an explicit lifecycle rather than as free-form flags: each entity moves through a defined chain of states (e.g., open, in preparation, collected, stored, under test, under review, completed), with rejection and cancellation as explicit side branches. This lifecycle both drives the user interface — which action a user can take depends on the current state — and provides the traceability trail that accreditation audits require, since every state transition is timestamped and attributable to a user. The design phase specified 31 use cases along the operational chain.

One deliberate boundary shapes the finance domain. Because SSLab shares its finance function with a sibling calibration unit under the same holding, management requested a single shared finance application serving both units. Senling therefore implements finance only as an integration point — an API-key-protected sales-order endpoint — and the shared finance application itself is outside the scope of this work.

E. Technology Selection

The stack is Laravel (PHP) with the Filament admin-panel framework on PostgreSQL, served by Nginx, all packaged with Docker Compose. Filament was chosen because the system is form-and-table intensive; building the equivalent admin UI from scratch would consume effort disproportionate to its value.

Hardware was selected against four explicit criteria: SBC form factor for a small on-premise footprint, a total procurement budget of roughly IDR 2 million, support for common Linux distributions (Ubuntu, Debian, Armbian) able to run the Senling stack, and processor headroom for long-term growth. Table III compares the two SBC candidates considered viable against these criteria. An Intel N100 mini PC was also considered but eliminated on form factor (not an SBC) and budget, not on performance. At near-identical price, the Orange Pi 5 Pro's eight-core big.LITTLE configuration was judged to offer more long-term capacity headroom than the Raspberry Pi 5's four-core configuration. This was a design consideration based on specifications, not a measured performance comparison.

TABLE III
SBC CANDIDATES COMPARED AGAINST SELECTION CRITERIA

Spec.	Raspberry Pi 5	Orange Pi 5 Pro
Processor	Cortex-A76, 4-core @ 2.4 GHz	RK3588S, 8-core big.LITTLE (4×A76@2.4GHz + 4×A55@1.8GHz)
Max. RAM	16 GB LPDDR4X	16 GB LPDDR4/5
Storage	microSD, PCIe M.2	microSD, M.2 NVMe
Total cost[a] (8 GB)	±IDR 2M	±IDR 2M

[a]Includes casing, cooling, and power adapter, at Indonesian market prices at time of procurement.

IV. Implementation and Evaluation Method

A. Development Process

Development proceeded iteratively, one business flow at a time, with the Orange Pi 5 Pro as the development and sandbox server throughout. Work began with a lightweight, file-based ticket system, one ticket per feature or fix. As flows grew more complex, particularly the order-to-acquisition chain, the team added a further discipline: a written plan was required before implementation began, reducing rework and keeping design decisions attached to the code they affected — important for a two-person team without a dedicated project manager.

Exposure to real operational use during development caused the use-case inventory to grow from the 31 designed to an as-built inventory of 39, reflecting a balancing of the ideal design, management requests, and field use. Some of the 39 remain designed-but-not-yet-built (e.g., digital certificate signing), so the paper distinguishes explicitly between the designed count (31), the as-built inventory (39), the tested subset (76 automated test cases across 21 test units), and the operationally used subset (part of the sales flow).

The deployed stack comprises four containers: the Laravel application, PostgreSQL, Nginx as reverse proxy, and a fourth, backup-labeled container discussed further below. Role-based access control, enforced at the panel level, distinguishes nine roles mapped onto the laboratory's business actors (Sales, Planning, Petugas Sampling, Teknisi, Penyelia, and MT/management).

Functionality is organized into modules (Table IV) that mirror the process groups, plus an unnumbered follow-up-and-resample module handling cases where a supervisor's rejection or a failed review sends a sample back for re-collection or re-testing. Each module is a set of Filament resources (list, form, and detail views generated from the underlying Eloquent models), consistent with the Filament rationale in Section III-E.

TABLE IV
IMPLEMENTATION MODULES, PRIMARY ACTORS, AND SCOPE

Module	Actors	Scope
Offer and Sales Order	Sales	Quotation drafting, revision, and order creation (UC-01–06)
Collection and Reception	Planning, Petugas Sampling	Pickup scheduling, chain-of-custody, sample intake (UC-07–22)
Scheduling and Testing	Petugas Sampling, Teknisi, Planning	Analyst/equipment scheduling, execution with per-parameter results and worksheet links (UC-23–29)
Supervision and Certification	Penyelia, MT, Sales	Result review, sign-off, certificate issuance (UC-30–39)

Inventory (reagents and equipment, FR-5) is managed as master data with its own system-level access role, but — per the case's own process documentation — has no dedicated business actor or process-flow module of its own; it is consumed indirectly wherever the Testing module records reagent and equipment usage.

Hosting the full stack on one SBC concentrates risk on a single device: there is no redundant node to fail over to. A fourth, backup-labeled container was present and running alongside the application, database, and reverse proxy, but its schedule, snapshot behavior, and restore procedure were not tested in this study and are not claimed here as a validated mitigation.

B. Evaluation Method

Evaluation addressed two questions: whether the system functions correctly over its tested scope, and whether the SBC sustains it under the load observed so far. Five instruments were used: automated functional testing, user testing, a perception questionnaire, a deployment demonstration, and resource monitoring, all on the Orange Pi 5 Pro (8 GB RAM) in its role as development/sandbox server; no production-scale load test was performed, and resource logs were captured at whole-OS granularity rather than per container.

The 39 as-built use cases were grouped into 21 test units by screen, so that steps belonging to one operation on one screen count as a single unit even when several use cases contribute to it. Each test unit was evaluated by two methods sharing the same input and expected output — automated PHPUnit tests for business logic, data integrity, and API endpoints, and manual user testing for interface-dependent actions (physical sample handling, barcode scanning, certificate signing) that automated tests cannot exercise. Table V summarizes coverage per functional requirement.

The six non-functional requirements were not evaluated uniformly, since each calls for a different method. NFR-1 (data integrity) and NFR-2 (access control) were verified through tests and an access-control matrix; NFR-3 (usability), NFR-4 (responsiveness), and NFR-6 (resource frugality) were measured, the first two via questionnaire and the last via resource logging; NFR-5 (deployment portability) was demonstrated through the repository-clone and Docker procedure described in Section V-C.

TABLE V
TRACEABILITY FROM FUNCTIONAL REQUIREMENTS TO USE CASES AND TEST UNITS

FR	Use Cases	Test Units
FR-1	UC-01–06	Test-1–6
FR-2	UC-07–22	Test-7–12
FR-3	UC-23–29	Test-13–15
FR-4	UC-30–35, 38–39	Test-16–20
FR-5	— (indirect)[a]	Integrity tests
FR-6	UC-36–37[b]	Test-21

[a]FR-5 is a master-data module without a dedicated operational use case, verified indirectly via entity-integrity tests. [b]FR-6 is realized only as an integration point (Section III-D); full finance functionality is out of scope.

V. RESULTS

A. Functional and Operational Evidence

All 76 automated test cases passed and none failed. This total comprises the automatable portion of the 21 test units (business logic and API endpoints) together with a separate suite of data-integrity checks: 106 foreign-key relations verified, and 9 negative-test cases across three integrity-rule categories (rejecting orphan foreign-key inserts, rejecting duplicate unique values, and blocking or cascading deletes of referenced parents). The role-permission matrix (9 roles $\times$ 14 function groups) also matched its specification. One test unit could not be exercised by either automated or manual testing at the time of writing — uploading a digitally signed certificate depends on a third-party signing integration that was still in progress — and is reported as an open item rather than a pass.

Fig. 3 shows the testing work-list screen, one of the Filament resources exercised by both automated tests and, on this

flow, real laboratory use: analysts record per-parameter results (Hasil) against each sample's threshold (Ambang/Satuan) and attach an external worksheet reference before submitting.

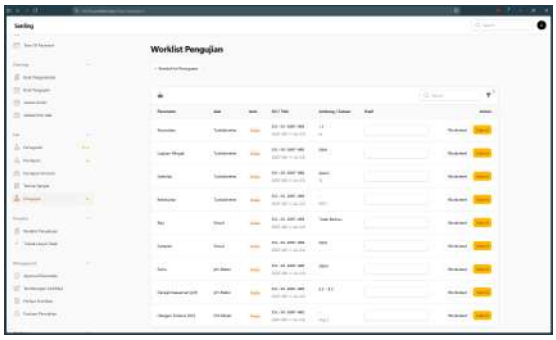

Fig. 3. Testing work-list screen (Worklist Pengujian), showing per-parameter result entry against thresholds, with worksheet attachment and submission controls.

In parallel, part of the sales flow — from quotation up to sample preparation — entered real operational use at SSLab during the study period, providing usage evidence beyond scripted scenarios. User testing was performed by a technical manager under supervisor oversight, covering the chain from quotation through supervisory review of test results, but not the later certificate-issuance stage; broader coverage across all nine roles is planned but not yet complete.

B. User Perception

A questionnaire on usability and perceived responsiveness (four-point Likert scale) was answered by five internal users spanning director, technical manager, quality assurance, analyst, and sales administration roles. Usability averaged 3.98 and perceived responsiveness 4.00. With five respondents, these scores are indicative only and cannot be generalized; they are reported as one strand of evidence among several. As an informal early indication of efficiency, staff directly involved estimated that preparing a quotation manually took roughly an hour, versus roughly 5–10 minutes in Senling; this is an uncontrolled verbal estimate, not a measured comparison.

C. Deployment and Resource Behavior

Deployment reproducibility was demonstrated as a four-step clean build (clone, build, seed, dashboard access) on WSL x86-64. On the Orange Pi ARM64 device itself, verification was limited to confirming that the containers of the same Docker Compose package were running; a from-scratch rebuild on the target device was not performed.

Resource behavior was logged over an observation window of about 39 minutes while the system was under light real use — specifically, live quotation preparation on the same device rather than a controlled load test. Table VI summarizes the results at whole-OS granularity: no resource showed signs of saturation. These figures describe a development/sandbox environment under one light-load window and do not establish production capacity.

TABLE VI
RESOURCE UTILIZATION OF THE ORANGE PI 5 PRO RUNNING THE FULL STACK (39-MINUTE WINDOW, LIGHT REAL LOAD)

Metric	Mean	Peak
CPU utilization	7.0%	13.9%
Memory utilization	34% (2.7 GB)	41%
SoC temperature	41.4°C	54.5°C
Load average (1 min, 8 cores)	0.64	—

VI. DISCUSSION

A. Synthesis

Taken together, the evidence supports an initial feasibility claim along two complementary, descriptive facets: the ERP functions over its tested scope — all automated tests pass, and part of the flow is in real operational use — and the SBC runs the full stack without observable resource saturation during the single light-load window observed. The two strands are not fully independent (part of the resource log was recorded during the same light real use), but assess different aspects: whether the software does its job, and what that job costs the device. Neither strand establishes production capacity, comparative hardware performance, or usability validation. It is this pairing that supports the feasibility claim: a system that only passed tests would leave open whether it fit the organization's workflow, and a system in real use without passing tests would leave open whether that use was masking latent defects.

The growth of the use-case inventory from 31 designed to 39 as built is itself a finding worth stating plainly: building against a live organization, rather than a fixed specification, surfaced needs (extended supervisory review, resample handling, additional certificate steps) the initial design did not anticipate. The chosen architecture — explicit domain boundaries within one deployable artifact — accommodated this mid-project growth without a full redesign, though no comparison against an alternative architecture was made to establish this as a general property. The hardware and architecture choices reported here are transferable considerations for a similarly scoped organization; the specific process diagrams, entities, and use cases are not, and would need to be re-derived for a different laboratory.

B. Scalability Outlook

Should load grow, two application-level paths are open: horizontal replication of the monolith behind a load balancer — for which SBC-cluster load balancing offers a limited precedent under static HTTP traffic [9] — or splitting a hot domain into a separate service along the already-explicit domain clusters [12], untested and requiring refactoring. The latter is only worthwhile once load across domains becomes markedly uneven; absent that evidence, keeping the monolith is the simpler default. On hardware, a multi-node SBC cluster or a move to a conventional x86 server (e.g., the Intel N100 eliminated in Section III-E) remain open if load outgrows the SBC, though no comparative benchmark was run to prefer either.

Deciding between these paths presupposes visibility this study lacks: the whole-OS resource log (Section V-C) cannot attribute load to the application, database, or reverse-proxy container individually. Per-container monitoring is therefore a precondition for choosing a scaling path with any confidence.

VII. Limitations and Threats to Validity

A. Construct and Internal Validity

"Initial feasibility" is operationalized narrowly as two observed conditions — that the ERP functions correctly over its tested scope, and that the SBC shows no resource saturation under light real load — not as economic feasibility, long-term maintainability, or user-acceptance validation. The usability/responsiveness questionnaire measures perception, not task-completion time or error rate. Part of the resource log and part of the user-perception evidence were collected during the same period of light real use, so the two are not fully independent. The manual-versus-Senling quotation-time comparison is an uncontrolled verbal estimate from two staff members, not a timed experiment, and could reflect recall bias.

B. External Validity

As a single-organization case study, the findings characterize this specific deployment — one laboratory, one device, one 39-minute window — and do not generalize to other laboratories, SBCs, or production load without further evidence. The five-user questionnaire sample is too small to generalize beyond a broadly favorable initial impression. The requirements are grounded in Indonesia's ISO/IEC 17025 regime as administered by KAN; laboratories under other accreditation bodies may face differently shaped traceability requirements, affecting how directly Section III-A transfers, even where the architectural and hardware considerations do not.

C. Other Limitations and Reproducibility

Resource measurements used whole-OS granularity, without per-container breakdown or a controlled workload script. The clean-build demonstration ran on x86-64, not the ARM target; on-device verification only confirmed containers were running, not a from-scratch rebuild. No production-scale load test or hardware comparison was performed. Senling is proprietary; its source and operational data are not public, so results rest on this paper's descriptions rather than a reproducible artifact. What is reproducible in principle is the method: the requirements analysis, hardware-selection procedure, and evaluation instruments do not depend on Senling's codebase and could be applied to a different laboratory, though the resulting entities and measurements would differ.

VIII. Conclusion

This paper presented an empirical design-and-deployment case study of Senling, a custom modular-monolith ERP for a small environmental testing laboratory, and an initial feasibility evaluation of hosting it on an Orange Pi 5 Pro. Within the observed conditions — a development/sandbox environment under one light-load window — the system passed all 76 automated test cases, entered partial real operational use, and showed no resource saturation on the device. The findings support initial feasibility for this case; they do not establish production capacity, comparative hardware performance, or generalizability to other laboratories. Read together with the requirement analysis, architecture, and hardware-selection method in Sections III and IV, the case suggests that a similarly scoped organization — a small team, a specific domain process, and a modest hardware budget — can reach a working, in-use ERP without an enterprise-scale platform or team, provided these caveats around production readiness and generalizability are carried forward.

Future work includes production load testing, per-container resource measurement, a controlled manual-process comparison, completing the remaining use cases, finance-application integration, and broadening the questionnaire to all nine business actors.

Acknowledgment

The authors thank Sentral Sistem Laboratory for access to its processes and staff during this study.

References

[1] E. F. Monk and B. J. Wagner, Concepts in Enterprise Resource Planning, 2nd ed. Boston, MA: Course Technology, 2008.

[2] ISO/IEC 17025, General requirements for the competence of testing and calibration laboratories, International Organization for Standardization, Geneva, Switzerland, 2005.

[3] D. O. Skobelev, T. M. Zaytseva, A. D. Kozlov, V. L. Perepelitsa, and A. S. Makarova, "Laboratory information management systems in the work of the analytic laboratory," Measurement Techniques, vol. 54, 2011.

[4] Odoo S.A., "Odoo documentation," 2025. [Online]. Available: https://www.odoo.com/documentation

[5] L. Arief, F. Akbar, N. Novani, and I. Saputra, "Pengujian kinerja server portable berbasis single board computer (SBC) dalam mendukung kegiatan pembelajaran," Jurnal Teknologi dan Sistem Informasi, vol. 4, pp. 98–106, Sep. 2018.

[6] J. Álvarez Ariza and H. Baez, "Understanding the role of single-board computers in engineering and computer science education: A systematic literature review," Computer Applications in Engineering Education, vol. 30, pp. 304–329, Jan. 2022.

[7] S. Cottafavi, "Odoo - free ERP on the Raspberry Pi," 2023. [Online]. Available: https://www.stefanocottafavi.com/embedded/odoo-raspberry/

[8] Frappe Community Forum, "Setting up ERPNext on a Raspberry Pi 5 running Ubuntu 23.10," 2023. [Online]. Available: https://discuss.frappe.io/t/setting-up-erpnext-on-a-raspberry-pi-5-running-ubuntu-23-10/111399

[9] R. A. Putra and A. P. Sujana, "Implementasi cluster server pada Raspberry Pi dengan menggunakan metode load balancing," Komputika Jurnal Sistem Komputer, vol. 8, no. 1, pp. 37–43, 2019.

[10] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in Proc. Int. Symp. High-Performance Computer Architecture (HPCA), Feb. 2013, pp. 1–12.

[11] R. Aroca and L. Gonçalves, "Towards green data centers: A comparison of x86 and ARM architectures power efficiency," Journal of Parallel and Distributed Computing, vol. 72, pp. 1770–1780, Dec. 2012.

[12] M. Fowler and J. Lewis, "Microservices: a definition of this new architectural term," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[13] Shenzhen Xunlong Software Co., Limited, "Orange Pi 5 Pro," 2024. [Online]. Available: http://www.orangepi.org/

[14] Raspberry Pi Ltd., "Raspberry Pi 5," 2024. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5