SAUDAGAR: A Path-Based Multi-Tenant Point-of-Sale Platform with Two-Layer Tenant Isolation for University-Fostered Micro, Small, and Medium Enterprises

H.A. Mooduto[a,*]

[a] *Department of Information Technology, Politeknik Negeri Padang, Jl. Kampus Limau Manis, Padang, 25164, Indonesia*
Corresponding author: *mooduto@pnp.ac.id

***Abstract*— Micro, small, and medium enterprises (MSMEs) in Indonesia still record many sales by hand, and commercial point-of-sale (POS) applications charge monthly fees that small vendors struggle to sustain. This study aims to design and evaluate SAUDAGAR (Sistem Aplikasi Usaha Dagang Rakyat), a free web-based POS platform operated by a polytechnic Techno Center for the MSMEs it fosters, in which one PHP application and one MariaDB database serve many enterprises at once. Following the design science research methodology, we derived 31 functional and eight non-functional requirements, a path-based tenancy scheme (/u/{slug}) that needs no wildcard subdomains, and a 23-table schema whose 18 POS tables carry a tenant identifier. Isolation is enforced in two layers: an application layer of tenant resolution, path-bound sessions, and a base model that injects the tenant filter, and a database layer of 21 composite foreign keys. The schema and core classes were evaluated in four laboratory experiments. The database rejected all 24 cross-tenant reference attempts, whereas a single-column-key baseline accepted all of them; all 20 application-layer test cases passed; per-tenant query cost stayed constant from 10 to 200 tenants holding 2.55 million orders, with tenant-prefixed indexes answering in 0.09 to 4.4 ms, 2.8 to 95 times faster than alternative indexing; and row-locked invoice numbering issued 2,000 concurrent numbers with no duplicates, against 1,287 duplicates without locking. The results indicate that a shared-schema design can give small enterprises strong isolation at low operating cost. Field deployment with fostered MSMEs and a usability study are the next steps.**

***Keywords*— Point of sale; multi-tenancy; tenant isolation; MSME digitalization; composite foreign key; design science research.**

I. INTRODUCTION

Micro, small, and medium enterprises (MSMEs) are the backbone of employment in Indonesia, yet their digital adoption remains shallow and uneven. A Bank Indonesia survey of 5,035 ultra-micro, micro, and small enterprises in 17 provinces placed the average digital adoption index at about 20 out of 100, with a clear gap between Java and the other islands and between micro and small firms [1]. Studies of the national MSME ecosystem point to the same pattern: digital tools spread quickly during the pandemic, but mostly as marketing and payment channels rather than as instruments for managing the business itself [2], [3]. The digital divide between firms translates into a measurable productivity gap [4], and the skills and tools available to owners shape whether digitalization leads to sustained performance [5], [6].

Sales recording is where this gap is most visible. Many food stalls and small shops still write orders in notebooks or keep paper receipts, so daily turnover, best-selling items, and cash discrepancies are known only approximately and only after the fact. Automated recording of transactions is a precondition for any later bookkeeping or analytics [7]. Point-of-sale (POS) applications address this directly, and their adoption by Indonesian MSMEs has been studied through technology acceptance and information-system success models [8], [9], [10]. These studies consistently report that cost, perceived complexity, and the need for assistance hold small vendors back, while the spread of the national QR payment standard (QRIS) has made non-cash payments common even in the smallest outlets [11]. Commercial cloud POS products typically charge a monthly subscription, which is a recurring burden for a business whose daily margin is small.

Cloud delivery would seem to solve the cost problem, but adoption studies of small firms show that price, trust, and security concerns weigh as heavily as functionality [12], [13], [14], [15]. Security and privacy are repeatedly named as decisive factors in whether a small enterprise hands its records to a hosted service [16]. A free platform offered by a public institution can remove the price barrier, but only if it can be operated cheaply and if each enterprise can trust that its sales data are invisible to the others.

Universities are well placed to provide such a platform. Politeknik Negeri Padang (PNP) fosters MSMEs through community engagement programs, and its Department of Information Technology runs a Techno Center whose lecturers and students can build and maintain software together. Project-based learning with real clients is known to strengthen both technical and professional skills of software engineering students [17], [18]. Earlier POS work aimed at MSMEs, however, has produced single-store applications, for example a POS calculator [19], a POS integrated with a payment gateway on a NoSQL store [20], or a web POS for one company [21]. Deploying one such application per enterprise multiplies installation, update, and backup work by the number of enterprises, which is unrealistic for a small university team, as is maintaining many separate systems for traditional-market vendors [22].

Multi-tenancy is the established answer: one instance of the software serves many customers (tenants) who share infrastructure but must not see each other's data [23]. Three data-placement models are common, namely a database per tenant, a schema per tenant, and a shared schema in which each row carries a tenant identifier; they trade isolation strength against operating cost [24], [25], [26]. The shared schema is the cheapest to run and the easiest to migrate in one step, but it also concentrates risk: a single query that forgets the tenant filter exposes every tenant [27], [28].

Broken access control, which includes exactly this kind of insecure direct object reference, is ranked first in the OWASP Top 10 web application risks [29], and multi-tenant SaaS needs controls that remain effective even when one component is compromised [30]. Most guidance for shared schemas relies on filtering in the application code; the database itself is rarely asked to refuse a cross-tenant relationship, and published MSME systems seldom report any empirical test of isolation at all.

This paper presents SAUDAGAR (Sistem Aplikasi Usaha Dagang Rakyat, "people's trade business application system"), a free web POS platform that PNP's Techno Center operates for its fostered MSMEs. The name means *merchant* and resonates with the trading tradition of the Minangkabau region where PNP is located. The contributions are threefold. First, we present a requirements set and a path-based multi-tenant architecture sized for a campus server that cannot issue wildcard subdomains. Second, we propose a two-layer isolation design in which application-level tenant filtering is backed by composite foreign keys, so that the database itself refuses cross-tenant references. Third, we evaluate the design in four reproducible experiments covering database isolation, application isolation, per-tenant query scalability, and concurrent invoice numbering. Section II describes the method and the design, Section III reports and discusses the results, and Section IV concludes.

II. Materials and Method

A. Research Design

The study follows the design science research methodology (DSRM), which structures the creation and evaluation of an IT artifact into six activities [31]. Design science requires that the artifact address a relevant problem and that its utility be demonstrated by rigorous evaluation [32]; when the problem is a social one such as small-business inclusion, the artifact should also respect the interests of the people it serves [33]. DSRM has been applied to comparable web information systems in Indonesia [34]. Fig. 1 shows how the six activities were instantiated. This article reports the first iteration, in which evaluation was carried out in the laboratory on the schema and core classes before field deployment.

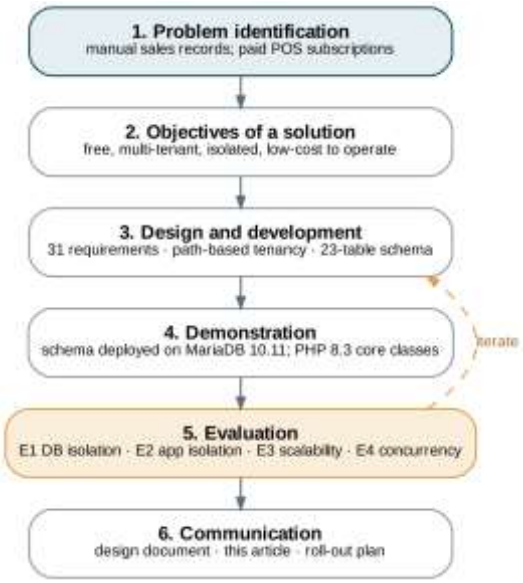


Fig. 1 Research flow based on DSRM; the dashed arrow marks the planned iteration after field evaluation

B. Problem Context and Requirements

Requirements were elicited from the Techno Center, from lecturers who mentor MSMEs in community engagement programs, and from the operating patterns of the culinary MSMEs that form most of PNP's fostered enterprises. Five stakeholder groups were identified: the Techno Center as platform operator, mentoring lecturers, information technology students as developers and field assistants, the MSMEs as daily users, and buyers who receive receipts without logging in. Two families of roles follow. At platform level, a super administrator, an operator, and a mentor work through the /admin area; at enterprise level, an owner, an administrator, a cashier, and a kitchen role work inside their own POS area. Mentors and the Techno Center may read aggregate sales of the enterprises they support but may never change a transaction, which keeps data ownership with the MSME.

Table I summarizes the 31 functional requirements. Platform modules (P-01 to P-11) cover the public site, registration and verification of fostered status, automatic provisioning, onboarding, quotas, and monitoring. POS modules (F-01 to F-20) cover the daily cycle of a food business: menu and variants, tables, orders for dine-in, take-away and online channels, the kitchen queue, split payments, 58 mm or 80 mm receipts, cashier shifts, simple expenses, and reports. Eight non-functional requirements complete the set; the most demanding is that enterprise A must not be able to read or change data of enterprise B even by editing an identifier or a slug in the URL.

TABLE I
SUMMARY OF REQUIREMENTS

Code	Requirement group
P-01–P-02	Public landing page; registration form with business, owner, program, mentor, and unique slug
P-03–P-05	Verification of fostered status with mentor confirmation; automatic provisioning; four-step onboarding
P-06–P-08	Tenant management and status changes; fair-use quotas; platform dashboard
P-09–P-11	Mentor monitoring; announcements and logged help access; data export and account closure
F-01–F-07	Login with lockout; business settings, tax and service rates; categories, menu, variants, tables, customers
F-08–F-12	Cashier screen; daily invoice and queue numbers; discounts; tax, service, rounding; kitchen queue
F-13–F-17	Split payment; receipts and digital receipt links; void with reason; cashier shifts; expenses
F-18–F-20	Sales, best-seller, payment-method, cashier, peak-hour, and gross-profit reports; dashboard; activity log
NFR-1–NFR-8	Data isolation; page load below 2 s on cellular networks; fair resource use; usability on tablets and phones; transactional integrity; daily backup; single code base; browser and thermal-printer compatibility

C. Architecture and Tenancy Model

Table II compares the three data-placement models against the operating conditions of a campus Techno Center. A database or an installation per enterprise would multiply migrations, backups, and updates by the number of tenants, while the team that maintains the platform is a handful of lecturers and rotating students. SAUDAGAR therefore uses one shared database in which every POS table carries `tenant_id`; the database-per-tenant model is kept as a fallback for later stages. This choice is consistent with work that treats the shared table as the most resource-efficient model, provided that isolation is engineered deliberately [25], [27].

TABLE II
MULTI-TENANCY MODELS CONSIDERED

Model	Strengths	Weaknesses	Decision
Shared database, tenant_id column	One migration for all; cheapest hosting; easy platform statistics	Every query must filter by tenant	Chosen
Database per tenant	Strong isolation; per-tenant backup	Tens to hundreds of databases; repeated migrations	Fallback
Installation per tenant	Free customization	Update and maintenance cost out of proportion for MSMEs	Rejected

The campus server does not support automatic wildcard subdomains, so tenants are addressed by path rather than by host name. Every enterprise receives an area under /u/{slug}, for example /u/warung-sate/pos for the cashier screen and /u/warung-sate/dapur for the kitchen display, while /admin serves the Techno Center and mentors and the root path serves the public site and registration. One domain and an ordinary SSL certificate are therefore sufficient. Fig. 2 shows the resulting architecture: three entry points lead into a single PHP 8 code base with a front controller, and the platform and POS tables live in one MariaDB database. The stack (native PHP with PDO, MariaDB 10.6 or later, Bootstrap 5, and Chart.js) was chosen because it runs on modest shared hosting and is already taught in the department, which lowers the barrier for students who will maintain the system.

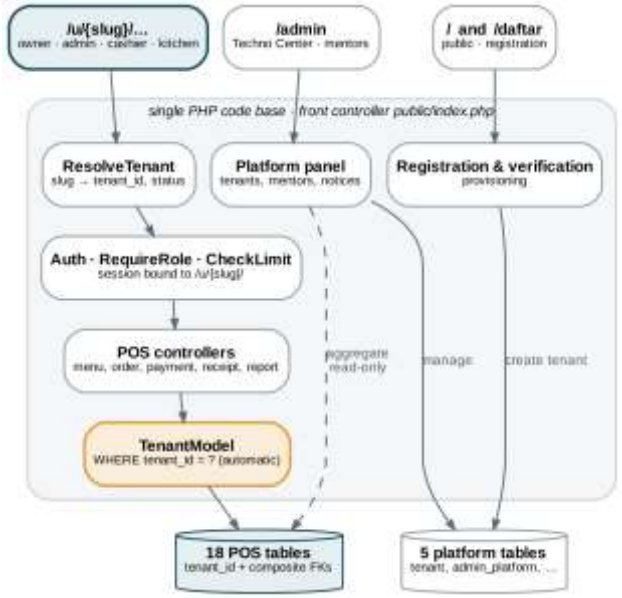


Fig. 2 Architecture of SAUDAGAR; three entry points share one PHP code base and one MariaDB database, and the platform panel reads POS data only in aggregate

D. Two-Layer Tenant Isolation

Isolation is designed as defense in depth: no single check is trusted to be sufficient. Fig. 3 traces a request through seven checks. The first four belong to the request pipeline. ResolveTenant extracts the slug with the pattern `^/u/([a-z0-9-]{3,40})`, rejects reserved words such as admin and api, loads the tenant row, and stores it in a request-scoped `TenantContext`. A status check returns 404 for tenants that are pending, rejected, or stopped and switches suspended tenants to read-only mode. The session cookie is scoped to the path /u/{slug}/ and stores the tenant identifier, so that a session presented under another slug is destroyed. Role and quota middleware then authorize the action.

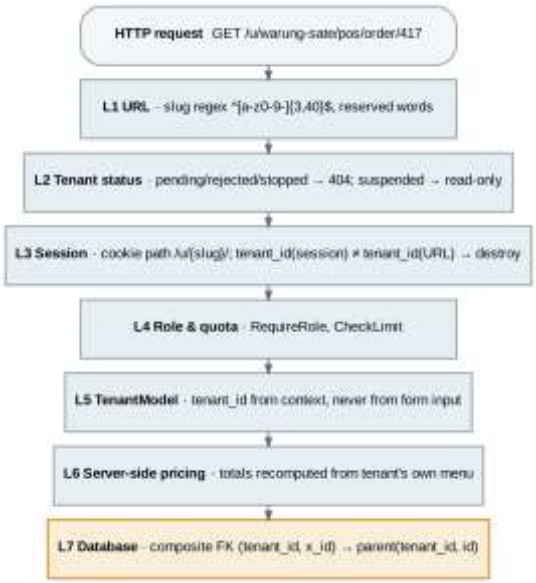


Fig. 3 Isolation checks applied to every POS request; L1–L6 run in the application and L7 in the database

The fifth and sixth checks live in the model layer. Every POS model extends an abstract `TenantModel` that prepends `WHERE tenant_id = ?` to reads, updates, and deletes, and that removes any `tenant_id` supplied in form data before inserting the context value. The developer of a new feature therefore cannot forget the filter, because the filter is not written in the feature at all. Totals are always recomputed on the server from the tenant's own menu and option rows; numbers sent by the browser are only for display, which also defeats price tampering. Database access uses prepared statements throughout, the standard defense against SQL injection [35], [36].

The seventh check is the database itself. Every parent table in the POS layer declares `UNIQUE (tenant_id, id)`, and every child table references its parent through a composite foreign key `(tenant_id, x_id) REFERENCES parent(tenant_id, id)`. For any row c in a child relation R that references a parent row ref(c), the database then enforces the invariant

$$tenant(c) = tenant(ref(c)) \quad \forall\, c \in R \tag{1}$$

so a detail line of enterprise A can never point to a menu item of enterprise B, whatever the application does. The schema contains 21 such composite keys across 11 child tables. Composite keys add no extra index, because the tenant-prefixed unique key on the parent and the tenant-prefixed foreign-key index on the child are needed for performance anyway.

E. Data Model

The database comprises 23 tables: five platform tables (platform accounts, platform settings, tenants, mentor assignments, and announcements) and 18 POS tables, all of which carry `tenant_id`. Fig. 4 shows the core of the POS layer, in which the order table connects menu items, tables, customers, payments, and cashier shifts. Four further rules complete the design. Names, prices, and costs are copied into order lines as a snapshot, so later price changes never rewrite past reports. Menu items, options, and users are deactivated rather than deleted, and transactions are cancelled or voided rather than removed, which keeps the audit trail intact. All secondary indexes begin with `tenant_id`, so a report for one enterprise scans only that enterprise's rows. Finally, daily invoice numbers of the form INV-YYYYMMDD-NNNN are drawn from a per-tenant counter row that is locked with `SELECT … FOR UPDATE` inside the order transaction, so each enterprise has its own gap-free sequence.

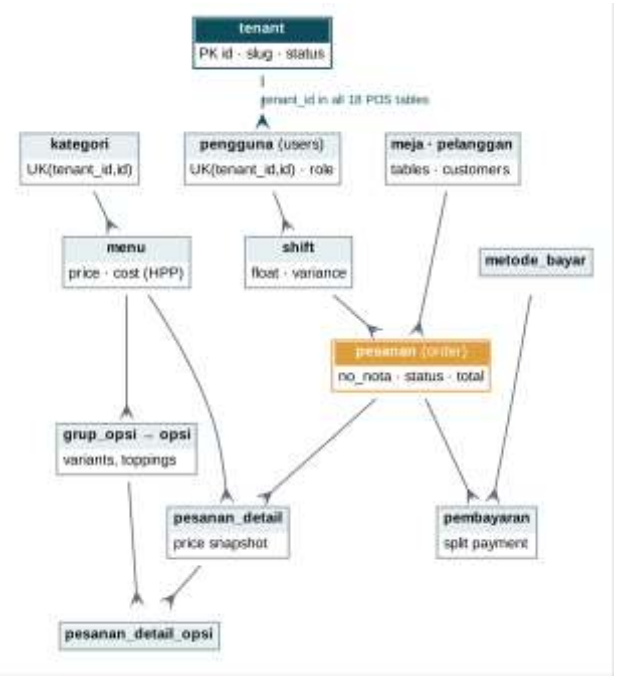


Fig. 4 Core entity-relationship diagram of the POS layer; crow's feet mark the "many" side, and each relationship is a composite (tenant_id, x_id) key

F. Tenant Lifecycle and Governance

Registration is open, but only enterprises verified as PNP-fostered are activated. Fig. 5 shows the five tenant states. An operator verifies the program and asks the named mentor to confirm; on approval, a provisioning service creates the owner account, default categories, payment methods (cash, QRIS, bank transfer), and receipt settings. A suspended tenant keeps read access to its reports but cannot record new sales. An enterprise that leaves the program can export its data, and its rows are purged after 90 days. When the Techno Center must enter a tenant's area to help, it states a reason, the access expires after 30 minutes, a banner is shown on the tenant's screen, and the event is written to the activity log. Using tenant data for research requires written consent, and only aggregates are used for program statistics.

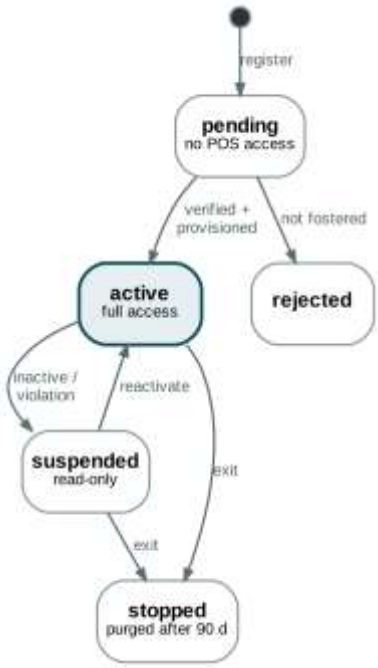


Fig. 5 Tenant lifecycle and the POS access granted in each state

G. Evaluation Design

The evaluation tests whether the design delivers the isolation, scalability, and integrity properties it claims. The complete DDL of the design document was executed unchanged on MariaDB 10.11.14, and the core classes (`TenantContext`, `TenantModel`, slug resolution, session binding, and server-side pricing) were implemented in PHP 8.3 exactly as specified. All experiments ran on a virtual machine with two vCPUs (Intel Xeon, 2.3 GHz) and 8 GB of RAM with the default 128 MB InnoDB buffer pool, a configuration comparable to an entry-level campus server. Four experiments were defined.

1) E1 Database isolation: Two tenants, A and B, were seeded with a complete set of rows. For each of the 21 composite foreign keys, a row of A was updated to point to the corresponding parent row of B; three further attacks inserted new rows of A that reference B. The same 24 attacks were repeated on a baseline schema generated from the same DDL in which every composite key was replaced by a conventional single-column key, which is the usual way such schemas are written.

2) E2 Application isolation: Twenty test cases exercised URL resolution, tenant status, session binding, the model layer, pricing, atomicity, and password handling. Each case has an expected outcome derived from the requirements.

3) E3 Per-tenant query scalability: Synthetic data for one year of trading were generated for 10, 25, 50, 100, and 200 tenants, with 40 menu items per tenant, a skewed best-seller distribution, and daily order counts drawn from log-normal distributions with medians between 8 and 70 orders. A fixed probe tenant with a median of 25 orders per day was measured at every step. Three index configurations of the order table received identical data: C1, the tenant-prefixed indexes of the design; C2, the same columns indexed without the tenant prefix plus a separate tenant index; and C3, a tenant index only. Four typical queries were timed: Q1, a 30-day daily sales report; Q2, the 30-day top-five menu items; Q3, the kitchen queue of open orders; and Q4, today's dashboard figures. Each query was run 25 times after a warm-up, and the median latency and the number of rows read (sum of InnoDB handler reads) were recorded.

4) E4 Concurrent invoice numbering: Eight workers, four per tenant, each requested 250 invoice numbers at the same time. The locked procedure of the design was compared with a naive read-then-write procedure without a row lock, and duplicate and missing numbers were counted.

III. Results and Discussion

A. Database-Layer Isolation

Table III summarizes E1. On the SAUDAGAR schema, every one of the 24 attempts was refused with MariaDB error 1452 (foreign key constraint fails), and no row changed. On the single-column baseline, all 24 attempts succeeded: enterprise A could attach enterprise B's menu item to its own order line, record a payment against B's order, open a shift for B's cashier, or attribute its expense to B's shift. Each of these is a silent corruption that would surface later as wrong totals in both enterprises' reports. The baseline is not a straw man; it is the schema most developers would write when adding a `tenant_id` column to an existing single-store design.

TABLE III
Cross-Tenant Reference Attempts by Child Table (E1)

Child table	Attempts	SAUDAGAR	Baseline
pesanan (order)	5	5 rejected	5 accepted
pembayaran (payment)	3 + 1 insert	4 rejected	4 accepted
pesanan_detail (line)	2 + 1 insert	3 rejected	3 accepted
pesanan_detail_opsi	2	2 rejected	2 accepted
pengeluaran (expense)	2	2 rejected	2 accepted
riwayat_harga (price log)	2	2 rejected	2 accepted
shift	1 + 1 insert	2 rejected	2 accepted
menu, grup_opsi, opsi, log_aktivitas	4	4 rejected	4 accepted
Total	**24**	**24/24 (100%)**	**0/24 (0%)**

The value of this layer lies in what it does when the application layer fails. A missed filter in a new controller, a hand-written report query, or a data-import script that bypasses the models can all produce cross-tenant references, and these are the code paths that student developers are most likely to write. Composite keys turn such defects from silent data corruption into immediate errors that are caught during testing. This complements guidance that places tenant isolation mainly in application or middleware code [26], [24], and it costs nothing at run time beyond the indexes that the design already needs.

B. Application-Layer Isolation

All 20 test cases of E2 passed (Table IV). URL resolution rejected malformed, reserved, traversal, and unknown slugs; pending tenants were hidden and suspended tenants were read-only; a session of A replayed under /u/umkm-b was destroyed. In the model layer, direct object references to B's menu or order returned nothing, an update aimed at B's menu changed zero rows, and a forged `tenant_id` in form

data was overwritten with A's identifier. A request made before the tenant was resolved raised an exception instead of returning unfiltered data. The client price of 1 for a menu item that costs 15,000 was ignored, and a cart line referencing B's menu was refused. Case T18 demonstrates the two layers working together: an order line inserted through the model with A's context but B's menu identifier passed the application check and was stopped by the database. Case T19 shows that a failed multi-step order left no partial rows because the order is written in one transaction.

TABLE IV
APPLICATION-LAYER TEST CASES (E2)

ID	Layer	Scenario	Result
T01–T05	URL	Valid slug accepted; malformed, traversal, reserved, and unknown slugs rejected	5/5 pass
T06–T07	Status	Pending tenant hidden (404); suspended tenant read-only	2/2 pass
T08–T09	Session	Session of A replayed under B destroyed; kept under A	2/2 pass
T10–T15	Model	IDOR on menu and order; listing scope; forged tenant_id; cross-tenant update; missing context	6/6 pass
T16–T17	Pricing	Client price ignored; foreign menu in cart refused	2/2 pass
T18	Both layers	Model insert referencing B's menu stopped by composite key	pass
T19	Atomicity	Failed order leaves no partial rows	pass
T20	Credentials	password_hash and password_verify round trip	pass
Total			**20/20**

C. Per-Tenant Query Scalability

The synthetic data set grew from 132,165 orders (10 tenants) to 2,548,442 orders and 5,521,343 order lines (200 tenants), while the probe tenant always held 9,841 orders. Table V lists the results at 200 tenants, and Fig. 6 shows how the work per query evolves. With the tenant-prefixed indexes of the design (C1), the rows read for each query were identical at every scale (853 for Q1, 5,535 for Q2, 5 for Q3, and 24 for Q4). Latency stayed between 0.09 and 4.4 ms, and Q1 rose only from 0.90 ms to 1.26 ms while the database grew nineteen-fold. In other words, the cost of serving one enterprise depends on that enterprise's own volume, not on how many enterprises share the server.

TABLE V
MEDIAN LATENCY AND ROWS READ FOR THE PROBE TENANT AT 200 TENANTS (E3)

Query	C1 (SAUDAGAR)	C2 (no prefix)	C3 (tenant only)
Q1 daily report	1.26 ms / 853	9.45 ms / 10,725	8.71 ms / 10,725
Q2 top-5 menu	4.39 ms / 5,535	13.12 ms / 14,517	12.32 ms / 14,517
Q3 kitchen queue	0.09 ms / 5	1.39 ms / 606	8.72 ms / 9,842
Q4 dashboard	0.09 ms / 24	1.08 ms / 9,866	8.13 ms / 9,842

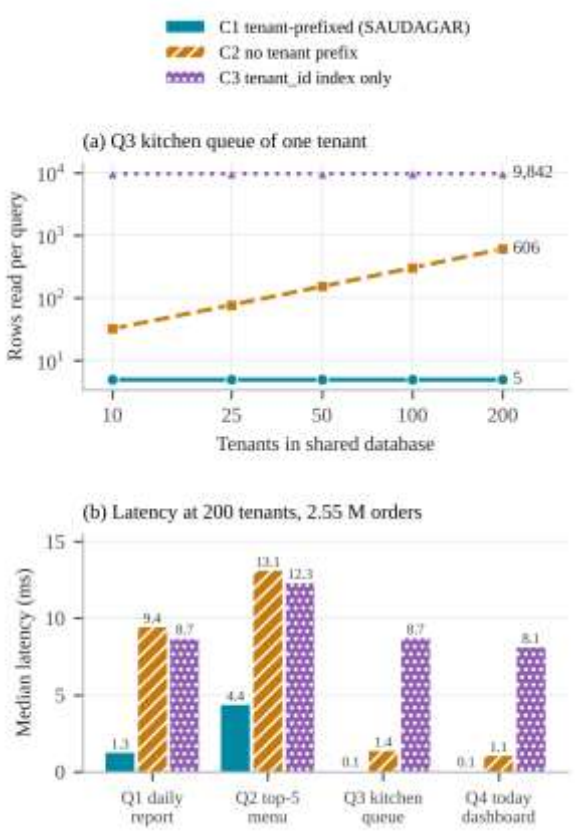


Fig. 6 (a) Rows read by the kitchen-queue query of one tenant as tenants are added (log scales); (b) median latency of the four queries at 200 tenants

The alternatives behave differently. C3 must read all 9,842 rows of the tenant for every query, so its cost grows with the tenant's history. C2 is subtler: its kitchen-queue index on status and time is shared by all tenants, so the open orders of every enterprise are scanned and then discarded, and the rows read grew linearly from 32 to 606 as tenants were added (Fig. 6a). This is the "noisy neighbour" effect that multi-tenant databases must manage [28], [24]. Defining the scan ratio as

$$\rho = \text{rows read (alternative)} / \text{rows read (C1)} \qquad (2)$$

gives ρ between 2.6 and 1,968 across the four queries at 200 tenants, and latency speed-ups of 3.0 to 15 times over C2 and 2.8 to 95 times over C3. The kitchen and dashboard screens refresh automatically every few seconds on many devices at once, so it is precisely these short, frequent queries whose cost must stay flat. With latencies of about 0.1 ms for them, the database is unlikely to be the bottleneck of the 2-second page-load target; connection handling and network latency will matter more, and connection reuse becomes relevant as the number of concurrent tenants grows [37].

D. Concurrent Invoice Numbering

Table VI reports E4. Without a row lock, workers read the same counter value before writing it back, and only 713 distinct numbers were issued for 2,000 requests, so 1,287 invoices (64.4%) would have duplicated another invoice number of the same day. With the locked procedure, each tenant received exactly the numbers 1 to 1,000 with no duplicates and no gaps. The lock is held per tenant and per

day, so cashiers of different enterprises never wait for each other, and throughput was not reduced (2,588 against 2,459 numbers per second). Sequential, unique invoice numbers matter to MSMEs because they are what owners, mentors, and tax officers use to reconcile receipts with cash.

TABLE VI
CONCURRENT INVOICE NUMBERING, 8 WORKERS × 250 REQUESTS (E4)

Measure	Naive read-then-write	Locked (SAUDAGAR)
Numbers issued	2,000	2,000
Distinct numbers	713	2,000
Duplicates	1,287 (64.4%)	0
Gaps per tenant	0	0
Throughput (numbers/s)	2,459	2,588

E. Discussion

Taken together, the experiments support the central design claim: a shared-schema POS platform can be cheap to operate and still give each enterprise strong, verifiable isolation. The cost argument matters because price is a recurring barrier in MSME technology adoption [8], [13], and the isolation argument matters because security and privacy concerns are among the strongest reasons small firms hesitate to put records in a hosted service [16], [14]. Compared with earlier MSME POS systems, which are single-store applications [19], [20], [21], SAUDAGAR treats multi-tenancy as a first-class requirement and tests it empirically. The governance rules (read-only mentoring, logged and time-limited help access, consent for research use, and export on exit) address the trust dimension that technical controls alone cannot.

The design also suits the institutional setting. One code base means that an update, a security patch, or a new report is written once and reaches every enterprise. The plain PHP and MariaDB stack and the explicit `TenantModel` give students a codebase they can understand within a semester, and the composite keys make it safe for them to contribute, because a mistake in a new query fails loudly rather than leaking data. In this way the platform can serve both community engagement and project-based learning [17], [18], and mentors gain structured sales data for monitoring their programs.

Several limitations apply. The evaluation was conducted in the laboratory with synthetic data, so the performance figures describe the database layer rather than end-to-end page loads on cellular networks. The application tests cover the core classes specified in the design document, not a complete user interface. No usability, acceptance, or business-impact data are yet available; these will be collected when the minimum viable product is piloted with three to five fostered MSMEs, for example with the DeLone and McLean success model used in recent POS studies [10]. The composite-key technique protects referential relationships but not every column, so the application layer remains necessary; it is a second layer, not a replacement.

IV. CONCLUSION

This study designed SAUDAGAR, a free multi-tenant web POS platform that a polytechnic Techno Center can operate for the MSMEs it fosters, and evaluated its isolation and performance properties. The platform uses path-based tenant addressing that works without wildcard subdomains, a 23-table shared schema, and a two-layer isolation design that combines automatic tenant filtering in the application with composite foreign keys in the database. In laboratory experiments the database layer rejected all 24 cross-tenant references that a conventional schema accepted, all 20 application-layer tests passed, per-tenant query cost remained constant from 10 to 200 tenants with millisecond latencies, and locked numbering produced gap-free invoice sequences under concurrency without loss of throughput. These findings indicate that a university can offer a shared POS service to many small enterprises with modest hardware and a small team, without asking those enterprises to trust a single line of filtering code. Future work will pilot the platform with fostered MSMEs, measure usability and acceptance, evaluate end-to-end performance over mobile networks, and add an offline-capable progressive web app, raw-material stock, and dynamic QRIS payments.

ACKNOWLEDGMENT

The author thanks the Techno Center of the Department of Information Technology, Politeknik Negeri Padang, and the lecturers and students involved in the SAUDAGAR design discussions for their input on the requirements and governance of the platform.

REFERENCES

[1] Y. Affandi, M. M. Ridhwan, I. Trinugroho, and D. Hermawan, "Digital adoption, business performance, and financial literacy in ultra-micro, micro, and small enterprises in Indonesia," Bank Indonesia, Jakarta, Indonesia, Working Paper RP/SSP/01/2023, 2023.

[2] E. Aminullah *et al.*, "Interactive components of digital MSMEs ecosystem for inclusive digital economy in Indonesia," *J. Knowl. Econ.*, vol. 15, no. 1, pp. 487–517, 2024, doi: 10.1007/s13132-022-01086-8.

[3] E. Kurniawati, I. Idris, P. Handayati, and S. Osman, "Digital transformation of MSMEs in Indonesia during the pandemic," *Entrep. Sustain. Issues*, vol. 9, no. 2, pp. 316–331, 2021, doi: 10.9770/jesi.2021.9.2(21).

[4] E. N. Lukas and A. Hasudungan, "The impact of the digital divide on MSMEs' productivity in Indonesia," *Int. Res. J. Bus. Stud.*, vol. 16, no. 3, pp. 241–252, 2024, doi: 10.21632/irjbs.16.3.241-252.

[5] M. Muafi, Z. M. Sanusi, and R. Roostika, "Digital skills, digital entrepreneurship, job satisfaction, and sustainable performance of MSMEs: A survey on MSMEs in Indonesia," *Int. J. Sustain. Dev. Plan.*, vol. 18, no. 2, pp. 465–473, 2023, doi: 10.18280/ijsdp.180215.

[6] H. Bahtiar, L. R. Rabbany, Y. F. Bele, M. Husna, and G. S. Matulessy, "Digital transformation towards sustainability: Challenges and opportunities for Indonesian MSMEs," *J. Ekon. Bisnis*, vol. 28, no. 1, pp. 131–150, 2025, doi: 10.24914/jeb.v28i1.13935.

[7] N. A. Azman, A. Mohamed, and A. M. Jamil, "Artificial intelligence in automated bookkeeping: A value-added function for small and medium enterprises," *JOIV: Int. J. Informatics Vis.*, vol. 5, no. 3, p. 224, 2021, doi: 10.30630/joiv.5.3.669.

[8] D. V. Eliezer and Sfenrianto, "Understanding POS ABC adoption in Indonesia: A quantitative analysis using modified UTAUT," *CommIT J.*, vol. 19, no. 2, pp. 167–181, 2025, doi: 10.21512/commit.v19i2.13258.

[9] N. A. Prasetyo, A. R. Bahtiar, A. Febriani, and W. A. Saputra, "Adoption of POS technology in MSMEs: A case study of Ecoprint Forum, Purbalingga," *J. Karya Abdi Masy.*, vol. 7, no. 1, pp. 19–28, 2023, doi: 10.22437/jkam.v7i1.21453.

[10] F. Nurdiyansyah, I. Akbar, and A. R. Muslikh, "Evaluation of the success of implementing point of sale applications for MSMEs using

the DeLone & McLean model," *JOINTECS*, vol. 10, no. 2, pp. 71–82, 2025, doi: 10.31328/jointecs.v10i2.7330.

[11] A. C. Sari and H. H. Adinugraha, "Implementation of QRIS-based payments towards the digitalization of Indonesian MSMEs," *Ekonomika Syariah: J. Econ. Stud.*, vol. 5, no. 2, 2022, doi: 10.30983/es.v5i2.5027.

[12] A. Gui, Y. Fernando, M. S. Shaharudin, M. Mokhtar, I. G. M. Karmawan, and Suryanto, "Cloud computing adoption using TOE framework for Indonesia's micro small medium enterprises," *JOIV: Int. J. Informatics Vis.*, vol. 4, no. 4, pp. 237–242, 2020, doi: 10.30630/joiv.4.4.458.

[13] H. Yaseen, A. S. Al-Adwan, M. Nofal, H. Hmoud, and R. S. Abujassar, "Factors influencing cloud computing adoption among SMEs: The Jordanian context," *Inf. Dev.*, vol. 39, no. 2, pp. 317–332, 2023, doi: 10.1177/02666669211047916.

[14] M. Skafi, M. M. Yunis, and A. Zekri, "Factors influencing SMEs' adoption of cloud computing services in Lebanon: An empirical analysis using TOE and contextual theory," *IEEE Access*, vol. 8, pp. 79169–79181, 2020, doi: 10.1109/ACCESS.2020.2987331.

[15] M. Chen, H. Wang, Y. Liang, and G. Zhang, "Net and configurational effects of determinants on cloud computing adoption by SMEs under cloud promotion policy using PLS-SEM and fsQCA," *J. Innov. Knowl.*, vol. 8, no. 3, Art. no. 100388, 2023, doi: 10.1016/j.jik.2023.100388.

[16] R. Nagahawatta, M. J. Warren, S. Salzman, and S. Lokuge, "Towards an understanding of cloud computing adoption in SMEs: The role of security and privacy factors," *Int. J. Cyber Warfare Terrorism*, vol. 14, no. 1, pp. 1–13, 2024, doi: 10.4018/IJCWT.343315.

[17] L. H. Silva, R. X. Castro, and M. C. Guimarães, "Supporting real demands in software engineering with a four steps project-based learning approach," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.: Softw. Eng. Educ. Training (ICSE-SEET)*, 2021, pp. 50–59, doi: 10.1109/ICSE-SEET52601.2021.00014.

[18] D. Stahl, K. Sandahl, and L. Buffoni, "An eco-system approach to project-based learning in software engineering education," *IEEE Trans. Educ.*, vol. 65, no. 4, pp. 514–523, 2022, doi: 10.1109/TE.2021.3137344.

[19] M. A. Rony and M. D. Anggraeni, "MSME digitalization using point of sale calculator to improve efficiency and productivity," *J. Informatics Inf. Syst. Softw. Eng. Appl. (INISTA)*, vol. 7, no. 1, pp. 118–126, 2024, doi: 10.20895/inista.v7i1.1635.

[20] I. Briliansyah and D. Avianto, "Point of sale application for MSMEs in Indonesia with payment gateway integration and NoSQL-based," *Int. J. Innov. Res. Eng. Manag.*, vol. 11, no. 6, pp. 138–150, 2024, doi: 10.55524/ijirem.2024.11.6.15.

[21] M. Melvin, J. Wiratama, R. Sutomo, and S. A. Sanjaya, "A web-based point of sales for automotive component industry using rapid application development model," *JOINS (J. Inf. Syst.)*, vol. 8, no. 2, pp. 167–176, 2023, doi: 10.33633/joins.v8i2.9383.

[22] B. H. Irawan, D. Prihadi, D. N. Rahmatika, C. Nugroho, J. Waskita, and O. M. Alsaifira, "Application development model e-commerce in traditional markets using the TOGAF framework," *JOIV: Int. J. Informatics Vis.*, vol. 8, no. 3, p. 1253, 2024, doi: 10.62527/joiv.8.3.2388.

[23] J. Kabbedijk, C.-P. Bezemer, S. Jansen, and A. Zaidman, "Defining multi-tenancy: A systematic mapping study on the academic and the industrial perspective," *J. Syst. Softw.*, vol. 100, pp. 139–148, 2015, doi: 10.1016/j.jss.2014.10.034.

[24] V. R. Narasayya and S. Chaudhuri, "Cloud data services: Workloads, architectures and multi-tenancy," *Found. Trends Databases*, vol. 10, no. 1, pp. 1–107, 2021, doi: 10.1561/9781680837759.

[25] S. K. Pippal, S. Kumar, and R. Rani, "Optimizing multi-tenant database architecture for efficient software as a service delivery," *TELKOMNIKA*, vol. 22, no. 5, p. 1128, 2024, doi: 10.12928/telkomnika.v22i5.26385.

[26] J. G. Kim, "Analysis of data isolation methods for secure web site development in a multi-tenancy environment," *J. Inf. Secur.*, vol. 24, no. 1, pp. 35–42, 2024, doi: 10.33778/kcsa.2024.24.1.035.

[27] G. Rosinosky, S. Youcef, F. Charoy, and É. Rivière, "A methodology for tenant migration in legacy shared-table multi-tenant applications," in *Distributed Applications and Interoperable Systems (DAIS 2021)*, LNCS, vol. 12718, Cham, Switzerland: Springer, 2021, pp. 3–20, doi: 10.1007/978-3-030-78198-9_1.

[28] G. Pallavi and P. Jayarekha, "An efficient resource sharing technique for multi-tenant databases," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 3, pp. 3216–3226, 2020, doi: 10.11591/ijece.v10i3.pp3216-3226.

[29] OWASP Foundation, "OWASP Top 10:2021," 2021. [Online]. Available: https://owasp.org/Top10/

[30] M. Yassin, H. Ould-Slimane, C. Talhi, and H. Boucheneb, "Multi-tenant intrusion detection framework as a service for SaaS," *IEEE Trans. Serv. Comput.*, vol. 15, no. 5, pp. 2925–2938, 2022, doi: 10.1109/TSC.2021.3077852.

[31] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2007, doi: 10.2753/MIS0742-1222240302.

[32] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.

[33] M. Monson, "Socially responsible design science in information systems for sustainable development: A critical research methodology," *Eur. J. Inf. Syst.*, vol. 32, no. 2, pp. 207–237, 2023, doi: 10.1080/0960085X.2021.1946442.

[34] R. G. Guntara, M. R. Nugraha, and M. D. A. Ridlo, "Web-based counseling skills evaluation information system using design science research methodology (DSRM) approach," *Int. J. Adv. Data Inf. Syst.*, vol. 4, no. 2, pp. 116–124, 2023, doi: 10.25008/ijadis.v4i2.1288.

[35] A. Paul, V. Sharma, and O. Olukoya, "SQL injection attack: Detection, prioritization & prevention," *J. Inf. Secur. Appl.*, vol. 85, Art. no. 103871, 2024, doi: 10.1016/j.jisa.2024.103871.

[36] M. Alenezi, M. Nadeem, and R. Asif, "SQL injection attacks countermeasures assessments," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 21, no. 2, pp. 1121–1131, 2021, doi: 10.11591/ijeecs.v21.i2.pp1121-1131.

[37] N. A. N. Sobri *et al.*, "A study of database connection pool in microservice architecture," *JOIV: Int. J. Informatics Vis.*, vol. 6, no. 2-2, p. 566, 2022, doi: 10.30630/joiv.6.2-2.1094.