International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Evolution of Cloud LMS Architecture: From Local Spreadsheet Workflows to a Distributed White-Label System

Burmistrov Aleksandr*

Chief Technology Officer at Piogroup LTD, Poland, Krakow
Email: burmistrov.alexander@gmail.com

Abstract

The article examines the architectural evolution of a commercial cloud-based learning management system that originated as a set of Google Sheets used to coordinate operations in a Ukrainian IT school and later evolved into the white-label platform DrivEd, adopted by networks of schools in the USA, Australia, and Europe. The study focuses on the transition from fragile spreadsheet-centred workflows to a dedicated LMS, and then to a multi-tenant, cloud-native solution that supports branded deployments for hundreds of tenant schools without code forks. The objective is to reconstruct the main architectural decisions that enabled this trajectory and correlate them with current research on cloud LMS and multi-tenant SaaS. The analysis relies on published work on LMS design, SaaS multi-tenancy, and serverless platforms, and employs an analytical case-study method. The conclusions highlight architectural patterns that are reproducible in other education-technology products facing international scaling demands.

Keywords: cloud learning management system; SaaS; multi-tenant architecture; white-label platform; Google Sheets migration; EdTech scalability; microservices; asynchronous data processing; international deployment.

Received: *12/10/2025*
Accepted: *2/10/2026*
Published: *2/20/2026*

---

* *Corresponding author.*

1.Introduction

Private training centres and boutique schools have traditionally organised their educational operations using office tools such as spreadsheets, shared documents, and messaging channels. In the case of later becoming DrivEd, all processes at an IT school in Kharkiv from 2014 onward were coordinated through Google Sheets, including enrolments, payments, group schedules, instructors' workloads, and reporting. Growth in student numbers and expansion into new cities increased the volume of interlinked spreadsheets and formulas, while almost every staff member had editing access. Formula chains frequently broke after accidental edits, errors were propagated across dependent tables, and locating the root cause required manual reconstruction of changes.These limitations produced two types of risk. First, operational reliability suffered: updates to payments and group assignments often lagged behind real transactions because staff had to trace and repair broken formula chains before confirming records; as a consequence, management periodically worked with inconsistent headcount data, and corrections left only partial traces in change history, which reduced auditability. Second, expansion into a network of branches across multiple cities required more granular access control than Google Sheets provided out of the box. The internal LMS, which a single engineer in the school had already initiated, became the foundation for a rapid expansion of the school's architecture. Synchronous online classes, remote attendance tracking, materials distribution, and basic gamification for engagement were added directly to the existing codebase without interrupting ongoing teaching.

Subsequent adoption of this system by external schools, first in Ukraine and later in the USA, Australia and several European countries, transformed the internal product into a white-label LMS. School networks in the USA and Australia with a combined footprint of more than 250 institutions utilize the platform as a shared technological foundation while retaining their own brands and pedagogical models. In this setting, the central question is not only functional coverage but architectural structure: how a system that started as a spreadsheet replacement can support hundreds of independent tenants, handle cross-border scale and preserve a single codebase across markets.

The purpose of the article is to analyze the evolution of the platform's architecture along three interrelated objectives:

1. To describe the transition from spreadsheet-centred workflow automation to a dedicated LMS with basic online functionality;
2. to examine how the architecture was reorganized to support multi-tenant, white-label operation across independent schools and networks;
3. to identify design decisions that enabled international scaling to new countries without per-tenant forks in code.

The novelty lies in combining a detailed case study of a commercial LMS that emerged from a single organisation with current research on cloud LMS, SaaS multi-tenancy, and serverless infrastructure. The focus is placed on architectural mechanisms rather than feature lists: data segregation models, tenant configuration, deployment patterns and operational practices that sustained the transition from local spreadsheets to a distributed white-label product.

2.Materials and Methods

The analysis draws on a set of recent works on cloud-based LMS, SaaS architectures and multi-tenant design, supplemented by practitioner case material on educational platforms. J. Azmitia [1] describes the evolution of a multi-tenant educational SaaS platform and discusses tenant isolation, caching and performance bottlenecks under school-centred workloads. S. Azouzi and S. Ayachi Ghannouchi [2] examine the design of multi-tenant e-learning systems in the cloud and emphasise process-oriented decomposition for higher-education environments. E. Bessonova and M. K. I. Gamage [3] formulate design principles for LMS in higher education, with attention to usability, complexity reduction and alignment of LMS structure with institutional processes. S. S. Bilur and M. S. Salunkhe [4] investigate a cloud-native, multi-tenant SaaS application with tenant-specific UI generation, highlighting how extensive customisation influences the architecture. M. Ghorbian and M. Ghobaei-Arani [5] provide a structured overview of serverless computing, discussing automatic scaling and cost models relevant for LMS back ends. R. Kumar [6] analyses design principles and security considerations for multi-tenant SaaS, focusing on database partitioning, access-control models and compliance. A. A. Marar, Y. Niharika, T. Akhila, V. Vaishnavi and B. B. M. [7] present a cloud-based LMS built on Google Firestore and authentication services, illustrating typical module decomposition and role-based access. H. Woods, T. Gilbert, A. Tate, and M. Song [8] discuss the design of multi-tenant architecture in cloud-native applications, utilising Kubernetes namespaces, service meshes, and tenant-aware governance. Z. Tao and H. Wang [9] propose an online education SaaS system based on Spring Cloud Alibaba microservices, demonstrating how monolithic LMS implementations transition to microservice-based, multi-module architectures. S. Werner [10] formulates a reference architecture for serverless big data processing, which informs discussions of event-driven components, analytics, and reporting in LMS backends.

The work uses an analytical case-study method. Materials on the DrivEd platform are presented as a longitudinal narrative of product evolution, which is compared against architectural patterns derived from sources [1–10]. The study uses comparative analysis of deployment and data-segregation models, functional–architectural mapping, and synthesis of recurring design motifs (multi-tenant isolation, configuration-driven branding, and progressive decoupling of services). No experimental measurements are performed; instead, the article reconstructs the architectural trajectory and evaluates it against current recommendations for scalable cloud LMS and multi-tenant SaaS engineering.

3.Results

The starting point of the evolution is a spreadsheet-centred infrastructure that encoded most of the school's operational logic in Google Sheets. Similar to the educational platform described by Azmitia [1], spreadsheets served as ad-hoc databases, state machines, and reporting tools simultaneously. Every new branch added its own copies and derivative sheets, while business logic was embedded in formulas rather than explicit application code. As in other documented cases, this design constrained scalability in three ways: the coupling between logic and data made changes risky; concurrent editing by staff compromised data quality; and the lack of programmatic interfaces hindered integration with payment systems, communication channels and future LMS components Reference [1,3].

The first architectural transition replaced this implicit logic with a dedicated LMS whose back end stored data in a central database and encapsulated business rules in application code. From the perspective of LMS design research, such a move corresponds to a shift from “tool-oriented” infrastructure to a coherent information system that explicitly models entities such as students, groups, schedules, payments and attendance [3]. The early versions of the platform replicated the workflows that previously existed in spreadsheets: enrollment, grouping, instructor assignment, and basic financial tracking. By concentrating these operations in a single system, the architecture eliminated formula-level fragility. It enabled consistent validation, transaction management and audit logging, in line with recommendations for online education SaaS systems [9].

When the original school opened branches in other cities, the architecture had to confront requirements typical of multi-site enterprise systems: a uniform curriculum across locations, local control by branch managers, central oversight, and consolidated reporting. In Google Sheets, this had been addressed by duplicating templates per city, resulting in a proliferation of linked files and brittle access controls. Under the LMS, these concerns were handled through explicit concepts of organisation, branch and role. Case materials emphasise the introduction of structured rights management that separated the visibility of groups, financial records and reports between headquarters staff and local administrators, while still using a single application for all branches. Research on multi-tenant architecture shows that such consolidation reduces operational complexity but requires stronger guarantees of logical isolation at the data and access-control levels [6, 8]. The platform’s transition from “one school, one codebase” to “one organisation with multiple branches on one codebase” corresponds to the move from deployment-level to application-level multi-tenancy, as described by Kumar [6], in which several organisational units share infrastructure but are separated in the domain model and authorisation layers.

The pandemic forced a second, more abrupt architectural phase. On-site teaching stopped almost overnight, while the LMS had been designed primarily as a back-office tool. To sustain operations, the platform integrated video-conferencing providers, implemented online group sessions, added content-sharing modules and introduced gamification elements targeted at adolescent learners. Research on cloud-based LMS emphasises that such evolution from administrative tooling to delivery platform increases the number and diversity of transactions by an order of magnitude [2, 7]. Lessons, homework submissions, chat messages, and achievements are all recorded, and each extends the system's functional and performance envelope. In architectural terms, the platform moved closer to the end-user-facing LMS described by Marar and his colleagues [7], in which students, teachers, administrators, and parents interact through differentiated roles and notification channels.

The white-label phase commenced when external schools requested access to the system under their own brand names. Instead of duplicating the codebase, the platform adopted a tenant-oriented model similar to those analysed by Song and his colleagues [8] and Bilur and Salunkhe [4]. Each tenant school corresponds to a separate logical space with its own user base, course catalogue, pricing rules and branding configuration. At the same time, the binary code and core service set remain shared. In multi-tenant terms, the evolution progressed from single-tenant deployment to organisational multi-tenancy, and then to cross-organisation multi-tenancy, where tenants are independent businesses. Literature on cloud-native multi-tenancy recommends three broad data-isolation patterns: a separate database per tenant, a shared database with tenant-specific schemas, and a fully shared schema with tenant identifiers [6, 8]. The DrivEd case materials emphasise the need to onboard new schools without code forks

and to maintain consistent upgrades across markets. Such requirements are typically satisfied by either a schema-per-tenant or tenant-id-per-row model with strict filtering in the data-access layer [4, 6, 8].

Figure 1 summarises this evolution against a generic cloud-based LMS architecture. The lower layer corresponds to the infrastructure services described in [2, 7], including cloud databases, object storage for learning materials, authentication, and messaging services. Above it, application services implement enrolment, scheduling, billing, content delivery and analytics; they expose APIs to web and mobile clients used by students, teachers, and administrators. The multi-tenant control plane mediates between tenant-agnostic services and tenant-specific configuration, including branding, feature flags, pricing models, and custom reports. DrivEd's white-label capabilities align with this separation: each school obtains its own URL, corporate identity and course structure, while relying on the shared implementation of scheduling, attendance and reporting.

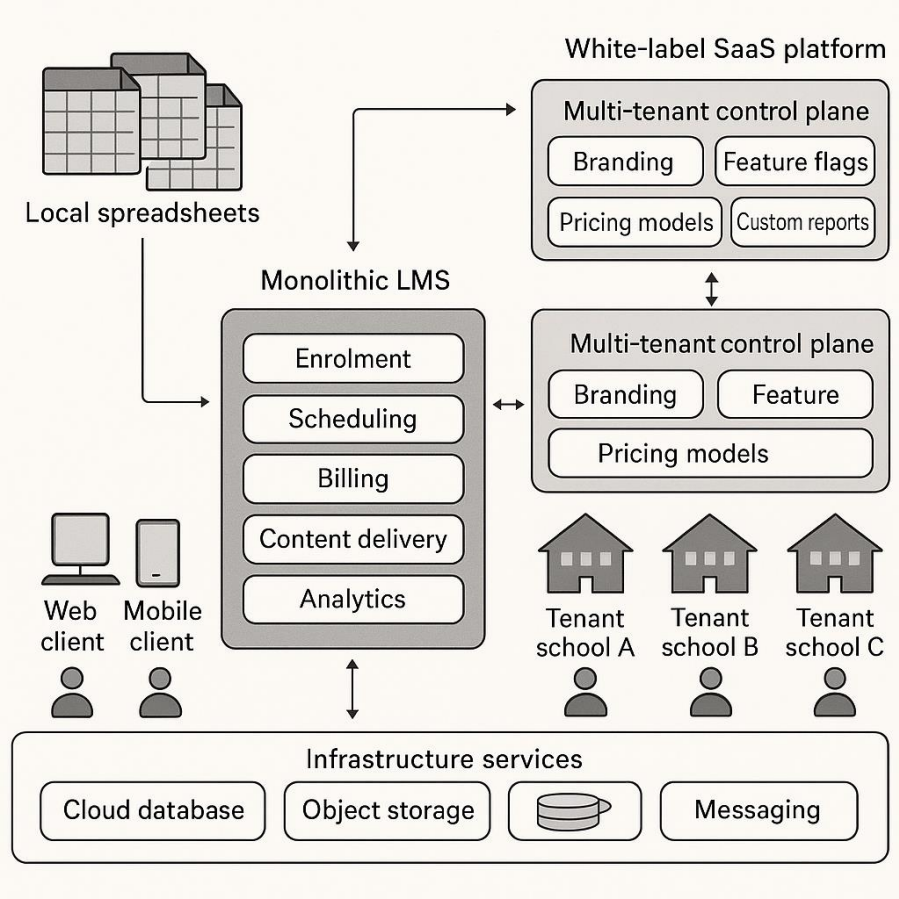

Figure 1: Generic cloud-based multi-tenant LMS architecture and the trajectory from local spreadsheets to a white-label SaaS platform (compiled by the author based on his own research)

International scaling added another architectural dimension. Schools in Australia and the USA work in different time zones, currencies and regulatory environments compared to those in Ukraine or the EU. Multi-tenant SaaS literature emphasises that such settings require careful separation between tenant-specific configuration and global platform behaviour [1,6,8]. Time-zone handling, localisation of interfaces, and adjustment of business hours are part of configuration; so are tax rules and integration endpoints for local payment providers. In contrast, the core

semantics of enrolment, attendance and lesson states must remain invariant to keep the codebase manageable. Kumar [6] emphasises the importance of uniform enforcement of security controls and logging across tenants. Werner [10] and Ghorbian and Ghobaei-Arani [5] describe serverless-based and big-data architectures in which analytics and reporting workloads move into event-driven background pipelines, reducing the impact of tenant-specific peaks on the transactional back end. In the DrivEd deployment, the same separation is achieved with asynchronous background services running on the platform's own servers: events from teaching workflows are written to queues and processed by worker services that aggregate data across tenants for reporting and decision-making without blocking transactional operations.

Finally, the platform's ability to scale to hundreds of schools without per-tenant forks depends not only on code and data structure but on deployment and operations. Cloud-native multi-tenant architectures, as described by Song and his colleagues [8] and Werner [10], rely on containerization, orchestration, and automated scaling rules. Namespace-based isolation, resource quotas, and tenant-aware routing through API gateways or service meshes enable a single cluster to serve multiple tenants while protecting them from noisy neighbour effects. In an LMS oriented toward synchronous teaching, peak loads occur around local lesson times; when tenants are distributed across time zones, these peaks partially offset each other, stabilising utilisation. This pattern aligns with reports from both the literature and practitioner case studies on educational SaaS platforms.

3.Discussion

The results allow the platform trajectory to be interpreted as a sequence of architectural "pressure points" at which operational failure modes forced explicit design choices. In the spreadsheet stage, operational state was encoded in formulas and cross-sheet references; therefore, correctness depended on user discipline rather than on enforceable invariants. The observed failure patterns—broken chains, silent propagation of wrong values, delayed reconciliation of payments, and weak traceability—map directly to the absence of transaction boundaries, schema constraints, and controlled write paths. In practical terms, this stage exhibits the same anti-properties that SaaS security and multi-tenant engineering literature treats as preconditions for incident emergence: ungoverned writes, weak audit trails, and implicit authorisation through document sharing [6].

After migration to an LMS back end, those failure modes became addressable through explicit domain entities and constrained state transitions. The empirical narrative in the Results section indicates that the main "gain" was not feature accumulation but the replacement of fragile spreadsheet semantics with an enforceable model: validated writes, consistent identifiers, and role-scoped views. This interpretation aligns with LMS research that frames platform maturation as a move from tool aggregation to institution-like information systems with explicit entities, lifecycle states, and traceable actions [3, 7].

Table 1 can be read as an architectural clarification of why the migration reduced operational uncertainty. "Implicit in formulas and sheet structure" indicates that the spreadsheet stage lacks a stable contract for data meaning; any structural edit effectively changes the semantics of stored records. In contrast, "Explicit schema with entities and relations" denotes a contract enforced by the database and application layer. The same logic applies to auditability: spreadsheet change history records cell edits, not business events. Centralised logging in an LMS can be designed

to record domain events (e.g., enrollment created, payment posted, attendance confirmed) together with actor identity and timestamps, which supports reconciliation and internal investigations. This distinction explains why the Results section reports fewer "invisible corrections" after migration: the data plane is no longer edited indirectly via formula chains, but rather through controlled operations [2, 3, 7].

Table 1: Operational characteristics before and after migration from Google Sheets to a cloud LMS [1–3, 7]

Dimension	**Spreadsheet-centred operations**	**Cloud LMS operations**
Data model	Implicit in formulas and sheet structure	Explicit schema with entities and relations
Consistency and integrity	High risk of broken formulas and silent errors	Database constraints and validated application logic
Access control	Shared editing links, coarse-grained permissions	Role-based access governed by LMS authorisation
Auditability	Limited change history, difficult to reconstruct modifications	Centralised logging of user actions and state transitions
Reporting	Manual aggregation across multiple sheets	Built-in reports across branches and programmes
Automation	Scripted macros tied to specific sheets	Reusable services and scheduled jobs in the back end

Operational characteristics before and after migration from Google Sheets to a cloud LMS [1–3, 7]. The evolution from single-organisation use to a white-label product extends this comparison into the domain of multi-tenant architecture. In an internal LMS, tenants correspond to branches of a single organisation; in a white-label product, tenants are independent businesses. This transformation demands a shift from ad-hoc configuration and manual onboarding to explicit tenant lifecycle management. Works by Kumar [6], Bilur and Salunkhe [4] and Song and his colleagues [8] outline recurrent patterns for such systems:

- tenant-aware data models (tenant identifiers, schema-per-tenant or database-per-tenant strategies);
- configuration layers that express branding, feature sets and local policies without branching the core code;
- deployment models in which tenants share infrastructure but receive strong isolation guarantees at the network, compute and data levels.
- Table 2 synthesises these patterns in relation to the DrivEd trajectory. According to the taxonomy in [4, 6, 8], such a design corresponds to a shared-application, logically isolated-data model with configuration-driven customisation, a combination that balances operational efficiency with tenant independence.

Table 2: Architectural patterns enabling white-label deployment across countries [1, 4, 6, 8–10]

Architectural concern	**Pattern**	**Implications for white-label LMS**
Tenant isolation in data	Shared application with logical tenant identifiers or schemas	Unified codebase, strong tenant-level separation of records
Tenant-specific configuration	Declarative configuration for branding, feature flags and pricing	Branded portals per school without code forks
Deployment and scaling	Container orchestration with namespace-based isolation and autoscaling	Elastic handling of regional peaks in lesson traffic
Compliance and observability	Centralised logging, audit trails and tenant-tagged metrics	Cross-tenant monitoring while preserving privacy
Analytics and reporting	Event-driven background pipeline on dedicated services fed from transactional events	Scalable cross-tenant analytics with low impact on core LMS

Sources on serverless and data-processing architectures [5, 10] highlight the advantage of decoupling real-time teaching workflows from heavy analytics jobs. In a multi-tenant LMS, each lesson generates various events, including attendance, submissions, grading actions, and content downloads. If these events are processed synchronously in the transactional database, tenants with intense usage in one region can degrade performance for others. In practice, event streams benefit from being routed to dedicated background pipelines—implemented either on serverless platforms or long-running worker services—so that computation-intensive operations, such as progress analytics or revenue reports, run independently of classroom activity. In DrivEd, this pattern is realised through asynchronous workers and queues deployed on the company's own servers rather than on third-party serverless infrastructure, which shifts code execution into the background while preserving complete control over the runtime environment. Ghorbian and Ghobaei-Arani [5] document the performance gains of such decoupling across several domains, including education, while Werner [10] formalises reference patterns for these data flows.

A closer comparison with prior work clarifies the architectural meaning of the "white-label phase" reported in the Results section. Bilur and Salunkhe [4] describe tenant-specific UI generation as a customisation driver that can easily spill into code divergence unless customisation is expressed declaratively. In the examined case, branded portals are defined as configuration outcomes (URL, identity, course structure), while scheduling and attendance remain shared services. This matches the configuration-centric position in [4] and provides a concrete explanation of how "no code forks" is sustained: customisation is moved to data and configuration, while the execution path remains uniform.

The Results narrative also describes a shift in analytics and reporting to asynchronous processing. Serverless literature and reference architectures [5,10] motivate this by isolating compute-heavy aggregation from interactive workloads. The reported implementation differs by using queues and long-running workers rather than third-party serverless functions. The architectural interpretation remains the same: teaching workflows emit events; aggregation consumes events; transactional paths avoid blocking on reporting. This alignment with [5,10] clarifies why international scaling did not require redesign of core lesson flows: latency-sensitive paths were protected by decoupling.

Finally, multi-tenant governance works in cloud-native settings [8], emphasising tenant-aware routing, quota enforcement, and mechanisms to mitigate "noisy neighbour" behaviour. The Results section notes that peak loads arise around lesson times and that time-zone distribution smooths aggregate utilisation. The literature adds a necessary clarification: smoothing reduces average contention but does not remove the need for isolation controls. In practice, tenant-tagged metrics and quota boundaries are the mechanisms by which the shared infrastructure remains stable under heterogeneous tenant behaviour [8].

The DrivEd case diverges from much of the LMS literature primarily in its origin conditions and workload profile. A substantial part of research prototypes assume an LMS is designed as a platform from the start, often for higher education or generic e-learning scenarios [2,3,7]. In the examined trajectory, architecture emerges from operational improvisation (spreadsheets) and only later becomes product-grade. This origin explains why backward compatibility with legacy workflows imposes a structural constraint: data semantics and user routines are inherited rather than specified from scratch.

The second divergence concerns synchronous group teaching in small and medium private schools. In such settings, the dominant peak driver is timetable concurrency rather than content consumption at scale. Many MOOC-oriented or higher-education systems foreground content delivery and assessment pipelines, while group lesson operations foreground schedule integrity, real-time attendance, and low-latency notifications. This difference helps interpret the architectural emphasis reported in the Results section: queue-based background aggregation is prioritised to protect interactive lesson flows, whereas customisation is confined to configuration to keep operational overhead low [4–6, 8–10].

In terms of SaaS engineering, the platform's journey confirms several expectations from the literature. Kumar [6] and Song and his colleagues [8] both emphasise that the evolution of multi-tenant SaaS benefits from early decisions that separate tenant-agnostic code from tenant-specific configurations. The DrivEd story suggests that, even when the initial system is not designed as a white-label product, paying attention to the separation of concerns and explicitly modelling organisational units lays the foundation for later multi-tenant generalisation. Bilur and Salunkhe [4] demonstrate how declarative configuration enables tenant-specific UI while preserving shared services; a similar principle underpins the branded deployments of DrivEd in different countries.

Prior studies provide strong taxonomies of multi-tenancy and customisation, yet they often under-specify the migration path from informal office-tool workflows to tenant-governed SaaS. In practice, the migration step is a significant risk surface: spreadsheets are not only a storage but a user interface, a computation engine, and an ad-

hoc integration layer. The examined case suggests that the decisive architectural act is to "decompose the spreadsheet" into a domain model with stable identifiers, controlled write APIs, and an event stream for derived reporting. This migration logic is consistent with the principles in [6] and the decoupling rationale in [5,10], while providing a more concrete pathway than is typically shown in cloud LMS prototypes [7] or process-oriented designs [2].

The results support several design implications that are directly testable in similar products. First, when a platform originates from spreadsheets, migration success depends on modelling the spreadsheet's "implicit workflow" as explicit state machines (enrollment → active learner → completion/withdrawal; payment → reconciled/failed; lesson → scheduled → delivered → recorded). Without this, the LMS merely reproduces spreadsheets through screens. Second, a white-label LMS remains operationally manageable when tenant variability is constrained to configuration surfaces such as branding, feature flags, pricing rules, localisation parameters, and integration endpoints. Third, cross-border expansion becomes an engineering problem of controlled variability: time zones, currencies, and local tax logic are part of tenant configuration, while lesson semantics and authorisation invariants remain uniform. These implications explain the trajectory described in the Results section and connect it to established SaaS design principles [6, 8].

The discussion of serverless computing by Ghorbian and Ghobaei-Arani [5] and the reference architecture by Werner [10] suggest future directions for LMS, such as DrivEd, including further decomposition of background tasks into event-driven functions, more granular autoscaling, and flexible data-processing pipelines for learning analytics. Given the platform's international footprint, serverless mechanisms can help minimise latency for users in distant regions and implement localised data-processing policies that comply with regulations requiring data residency or tenant-specific encryption.

This study is based on an analytical case-study reconstruction rather than on instrumented experimentation. The Results section describes architectural phases and inferred design consequences. Still, it does not report quantitative performance metrics (latency distributions, peak lesson concurrency throughput, queue backlogs, or cost profiles per tenant). Therefore, claims about scalability are supported by architecture–pattern alignment with the literature [4–6, 8–10] and operational narratives, rather than controlled benchmarking.

The case material is centred on one product lineage; generalising to other LMS segments (e.g., higher education with complex assessment workflows or MOOC-scale content delivery) requires caution, as workload structures and compliance constraints differ. The white-label interpretation is grounded in multi-tenant SaaS taxonomies. Yet, the study does not provide formal verification of tenant isolation (e.g., penetration testing results, cross-tenant data-leakage tests, or audited compliance reports).

Finally, parts of the platform description remain necessarily high-level to avoid disclosure of proprietary implementation details. This reduces the reproducibility of specific low-level mechanisms (such as the exact database partitioning strategy, concrete gateway routing rules, or internal SLO thresholds). Future work can address these limitations by adding anonymised telemetry aggregates, workload traces, and a structured evaluation protocol for tenant isolation and noisy-neighbour resistance.

4.Conclusion

The architectural evolution of the examined LMS illustrates how an internal automation tool built around Google Sheets can be transformed into a distributed white-label SaaS platform. The first research objective, the reconstruction of the transition from spreadsheets to a dedicated LMS, shows that replacing implicit spreadsheet logic with an explicit domain model and a transactional back end removes structural fragility in formulas, improves consistency, and supports branch expansion. The second objective, analysis of multi-tenant reorganisation, reveals that explicit modelling of organisations, branches, and roles, combined with tenant-aware data segregation and configuration layers, enables independent schools to share a single codebase while maintaining branded portals and isolated data. The third objective, identification of design decisions that support international scaling without per-tenant code forks, highlights several key patterns: configuration-driven localisation, separation of transactional and analytical workloads, and deployment models based on container orchestration with tenant-aware routing.

The reported findings are interpretive and architecture-centred; a metric-based validation using anonymised telemetry and tenant-isolation test suites is a logical next step for strengthening external validity.

The DrivEd case demonstrates that multi-tenant, cloud-native architectural principles from general SaaS literature apply to education products that originate from modest, spreadsheet-based workflows. At the same time, the case underscores that early attention to data modelling, access control and configuration structures strongly influences whether such products can later evolve into robust white-label systems for international markets.

References

[1]. Azmitia, J. (2024, December 19). *Building multi-tenant SaaS: Lessons from an educational platform.* Josias Azmitia's technical blog. https://www.josiasemanuel.dev/blog/multi-tenant-saas-architecture

[2]. Azouzi, S., & Ghannouchi, S. (2025). Designing multi-tenant e-learning systems in the cloud: A process-oriented approach for higher education. *International Journal of Computer Applications, 187*(16), 16–24. https://doi.org/10.5120/ijca2025924998

[3]. Bessonova, E., & Ihala Gamage, M. K. (2024). *Design principles for learning management systems in higher education* (Master's thesis, Lund University). Lund University Publications. http://lup.lub.lu.se/student-papers/record/9164253

[4]. Bilur, S., & Salunkhe, M. (2024). Multi-tenant yet customizable cloud native SaaS web application leveraging AI/ML: Architecture and strategies. *International Journal of Advances in Engineering and Management, 6*(11), 118–127. https://doi.org/10.35629/5252-0611118127

[5]. Ghorbian, M., & Ghobaei-Arani, M. (2025). *Serverless computing: Architecture, concepts, and applications.* arXiv. https://doi.org/10.48550/arXiv.2501.09831

[6]. Kumar, R. (2020). Multi-tenant SaaS architectures: Design principles and security considerations. *International Journal of Research and Analytical Reviews, 5*(2), 49–62. https://doi.org/10.35629/8193-05024962

[7]. Marar, A., Niharika, Y., Akhila, T., Vaishnavi, V., & B. M. B. (2025). Cloud-based learning management

system. In *Proceedings of the 3rd International Conference on Futuristic Technology (Vol. 3: INCOFT, pp. 152–159)*. SciTePress. https://doi.org/10.5220/0013610400004664

[8]. Song, M., Woods, H., Gilbert, T., & Tate, A. (2024). *Multi-tenant architecture design in cloud-native applications.* ResearchGate. https://www.researchgate.net/publication/392163585_Multi-Tenant_Architecture_Design_in_Cloud-Native_Applications

[9]. Tao, Z., & Wang, H. (2023). Design and implementation of an online education SaaS system based on microservice architecture. In D. Kumar et al. (Eds.), *Proceedings of IEIT 2023* (pp. 481–488). Atlantis Press. https://doi.org/10.2991/978-94-6463-230-9_57

[10]. Werner, S., & Tai, S. (2024). A reference architecture for serverless big data processing. *Future Generation Computer Systems, 155,* 179–192. https://doi.org/10.1016/j.future.2024.01.029