bilimger.kz
Республикалық білім және ғылым порталы
https://bilimger.kz
ҚР Мәдениет және Ақпарат министрлігі тіркелген — Куәлік нөмірі: ***KZ45VPY00102718***
2026

БӨЛІМ: FARABI UNIVERSITY / UNIVER / СТУДЕНТ

Architecture of a scalable spaced repetition system based on an event-driven model

ЖАРИЯЛАНДЫ	СІЛТЕМЕ
17.04.2026	https://bilimger.kz/188203/

Тыныштық Азамат Қуанышбекұлы
Әл-Фараби атындағы Қазақ ұлттық университеті
Ақпараттық технологиялар факультеті
Ақпараттық жүйелер кафедрасы 4 курс студенті

Жетекшісі: ***Көпбосын Лейла Серикбековна***

Abstract

The rapid expansion of digital learning platforms, together with growing demands for personalization, low-latency interactions, and continuous synchronization of educational data, creates substantial architectural pressure on backend systems. A modern spaced repetition platform must coordinate user identities, learning content, review sessions, reminders, subscription logic, collaboration workflows, analytics, and search. In tightly coupled systems, such responsibilities often accumulate in long chains of synchronous service calls, which reduces resilience, complicates scaling, and slows down product evolution. This paper presents the architecture of a scalable spaced repetition system based on an event-driven model, using the Learnly platform as a practical architectural reference. The proposed approach combines microservices, asynchronous domain events, schema-governed messaging, polyglot persistence, and projection-oriented read models. Kafka acts as the event backbone, while Avro schemas and Schema Registry enforce contract discipline between producers and consumers. PostgreSQL, MongoDB, Redis, and Elasticsearch are used according to workload characteristics. Special attention is paid to deck synchronization, subscription propagation, reminder delivery, search indexing, and idempotent event consumption. The study shows that the event-driven model enables the platform to separate user-facing command paths from downstream reactions, reduce service coupling, support replay-safe processing, and scale critical workloads independently.

Keywords: event-driven architecture, spaced repetition, microservices, scalable learning

platform, Kafka, domain events, distributed systems, asynchronous processing, projections

1. Introduction

Educational software has evolved from static content repositories into rich, continuously reacting digital ecosystems. A contemporary spaced repetition platform no longer stores only flashcards and user accounts. It coordinates adaptive review schedules, queue generation, user progress, reminders, social interactions, premium access policies, content sharing, analytics, and search. Each user action may therefore trigger a set of dependent processes. Updating a deck can affect review-ready projections, completing a study session can update progress statistics and streaks, and a subscription change can alter access to premium features in several subsystems at once.

If these processes are implemented primarily through synchronous point-to-point integrations, the system gradually becomes brittle. Every additional dependency increases the chance that one slow or unavailable service will degrade the user experience in unrelated flows. Under growth, the architecture begins to suffer from high latency, deployment coupling, cascading failures, and operational complexity. These issues are especially visible in learning platforms, where interactions are frequent, state transitions are numerous, and product teams continuously add recommendation logic, analytics, gamification, or proactive notifications.

An event-driven architecture offers a more sustainable model. Instead of coordinating all reactions within the critical request path, a service completes its local domain change and emits a domain event. Other services subscribe to this event and independently update their own state, projections, or delivery pipelines. This pattern preserves domain autonomy while enabling scalable fan-out. The objective of this work is to present a practical architectural model for a scalable spaced repetition system built on this principle and to show how the Learnly platform can be framed as a resilient educational event mesh rather than a conventional CRUD-oriented backend.

2. Architectural Context of Learnly

Learnly is structured as a distributed microservice platform centered on learning content delivery, user progress management, subscription-enabled features, and asynchronous educational interactions. The platform includes an API gateway, user service, content service, learning service, payment service, notification service, analytics service, collaboration service, storage service, URL shortener service, an indexing service, and a search layer. Each service focuses on a distinct domain, yet the complete product experience depends on coordinated reactions among them.

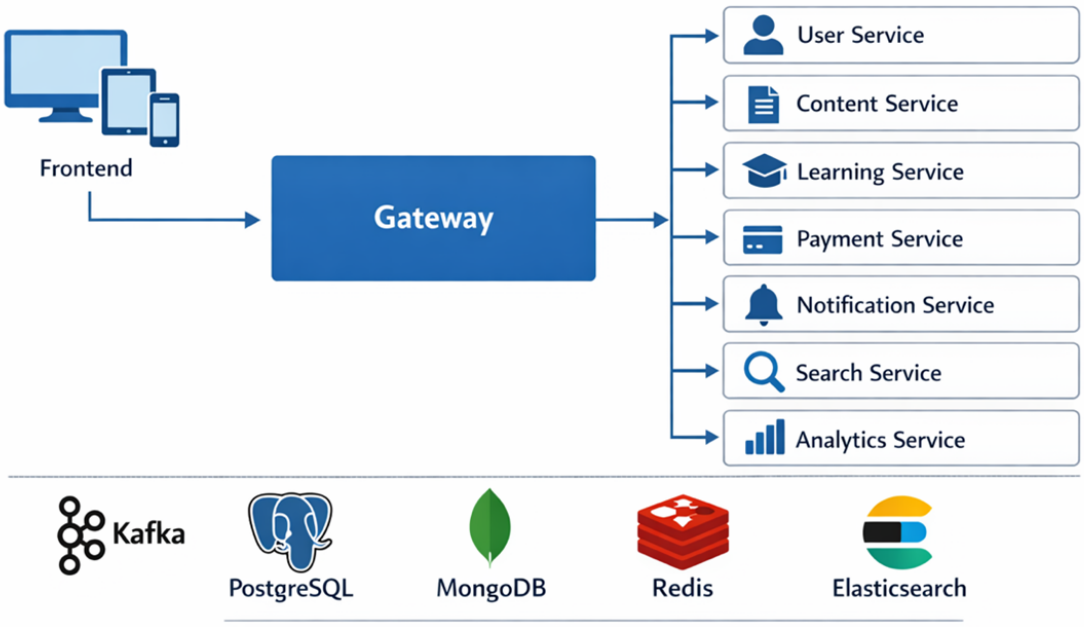

Figure 1. High-Level Architecture of the Learnly Platform

The technological profile of the platform reflects workload specialization. Transaction-sensitive domains, such as users and payments, rely on PostgreSQL. Content and learning domains use MongoDB for flexible document-oriented models and evolving aggregates. Redis is applied to rate limiting, duplicate-suppression guards, and short-lived operational counters. Search-facing read models are handled through Elasticsearch. Kafka acts as the inter-service event transport layer, while Avro schemas and Schema Registry provide contract discipline for event evolution.

Service	Primary Responsibility	Persistence	Event Role
Gateway	Public ingress, routing, security boundary	Stateless edge layer	Routes command traffic
User Service	Identity, profiles, social graph, account-facing state	PostgreSQL plus domain snapshots	Publishes user communication events and consumes subscription updates
Content Service	Decks, cards, folders, publication workflow, AI-assisted generation	MongoDB	Publishes deck synchronization events and consumes entitlement updates
Learning Service	Review sessions, card states, streaks, reminders, user statistics	MongoDB	Consumes deck updates and publishes reminder events
Payment Service	Subscriptions, transactions, external billing webhooks	PostgreSQL	Publishes subscription state events

Notification Service	Email confirmation, password reset, reminder delivery	Operational state with Redis safeguards	Consumes user and reminder events
Indexer and Search	Searchable read models and discoverability	Elasticsearch	Consumes search-oriented update streams

3. Core Components of the Event-Driven Architecture

The API gateway serves as the ingress layer and intentionally keeps orchestration responsibilities small. Its role is limited to routing, security integration, and request coordination at the boundary of the platform. Core domain services own their business rules and data. In the proposed model, the command side is responsible for validating requests, persisting domain changes, and publishing resulting events. It does not synchronously orchestrate every downstream consequence. A deck update should not block on reminder logic, analytics enrichment, or search indexing. A subscription state change should not require a chain of direct calls to every service that depends on entitlements.

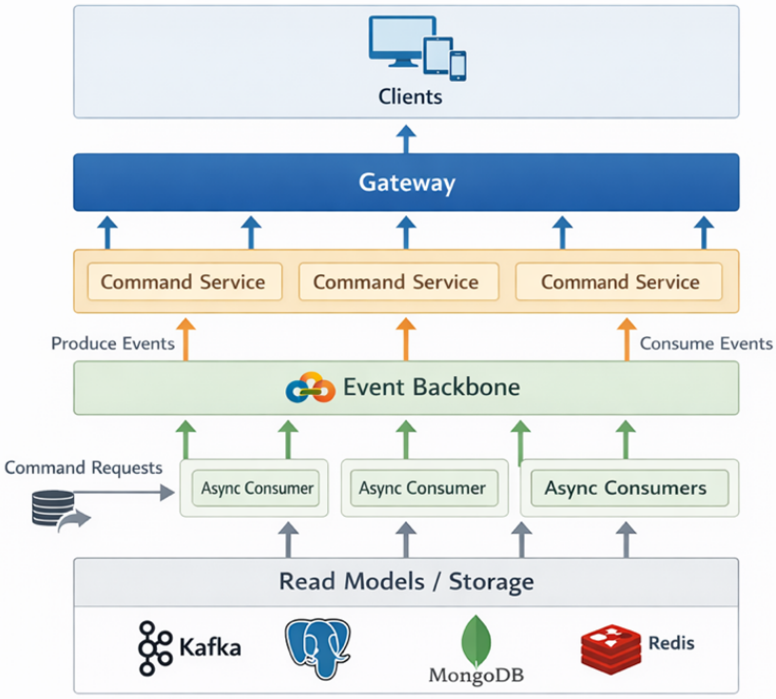

Figure 2. Layered Event-Driven Architecture of the Spaced Repetition Platform

Kafka functions as the backbone for decoupled asynchronous communication. Event streams are partitioned by aggregate keys such as deckId or userId, preserving local ordering where it matters while enabling horizontal scaling across consumer groups. Avro schemas provide compact serialization and explicit structure for domain events, while Schema Registry allows

producers and consumers to evolve at different speeds without silently breaking compatibility. In a platform expected to expand feature by feature, schema evolution, replay-safety, and compatibility management are architectural necessities rather than optional refinements.

A further strength of the architecture lies in assigning persistence technologies according to access patterns. PostgreSQL supports strongly consistent transactional domains. MongoDB stores flexible educational and content documents that evolve quickly as the product grows. Redis supports operational guards and short-latency checks. Elasticsearch powers read-optimized search experiences. In the event-driven model, many services consume events not to execute core business logic directly, but to build projections: local read models derived from another service’s events and optimized for fast queries or domain-local decisions.

4. **Representative Event Flows**

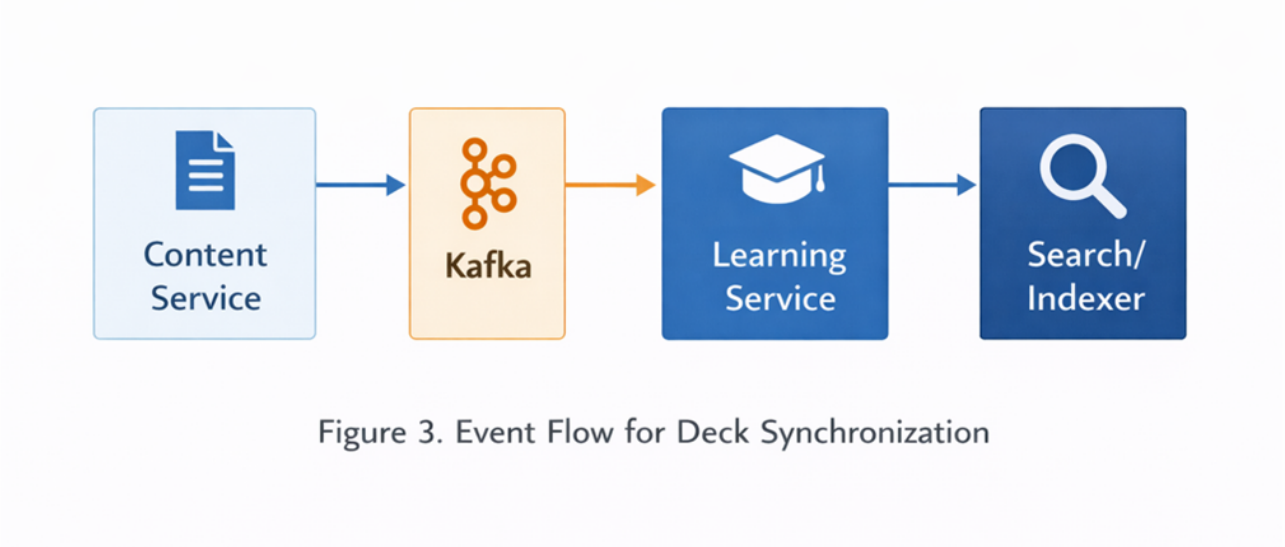

Figure 3. Event Flow for Deck Synchronization

The deck synchronization flow is one of the clearest illustrations of the event-driven model. When a user creates, updates, or deletes a deck, the content service persists the local change and publishes a synchronization event. The event may contain either a full snapshot of the deck state or a delta of changed cards. The learning service consumes this stream and updates its local card catalog, user deck representations, and review-related structures. This pattern is particularly effective for a spaced repetition platform because the content service remains the source of truth for authoring, while the learning service is free to maintain a model optimized for review operations.

Subscription handling illustrates another important principle: services should not depend on the payment service in real time for every authorization decision. When the payment domain changes subscription state, it emits a subscription event that is consumed by other services. The user service updates its local view of the effective tier, while the content service maintains an entitlement snapshot used for premium feature gating. This architecture significantly improves resilience because user and content operations can continue to make decisions

based on local projections even if the payment domain is temporarily unavailable.

Reminder delivery is central to retention in spaced repetition systems. Learnly’s architectural model supports a dedicated reminder scheduler within the learning domain. The scheduler evaluates user preferences, time zones, due-card counts, current streak status, and notification windows. When a reminder condition is met, the learning service emits a reminder event to Kafka. The notification service consumes the event and handles delivery through email infrastructure. The learning domain decides when a reminder should exist; the notification domain decides how it should be delivered. This separation allows both parts to evolve independently while remaining safe under retries and replays.

Event Stream	Key	Producer	Consumer	Architectural Purpose
Deck synchronization	deckId	Content Service	Learning Service	Builds review-ready projections and keeps learning-side catalogs up to date
Subscription state	userId	Payment Service	User Service, Content Service	Propagates entitlement state without synchronous payment lookups
Reminder events	userId	Learning Service	Notification Service	Separates reminder eligibility from delivery execution
User communication events	userId	User Service	Notification Service	Handles email confirmation and password reset workflows

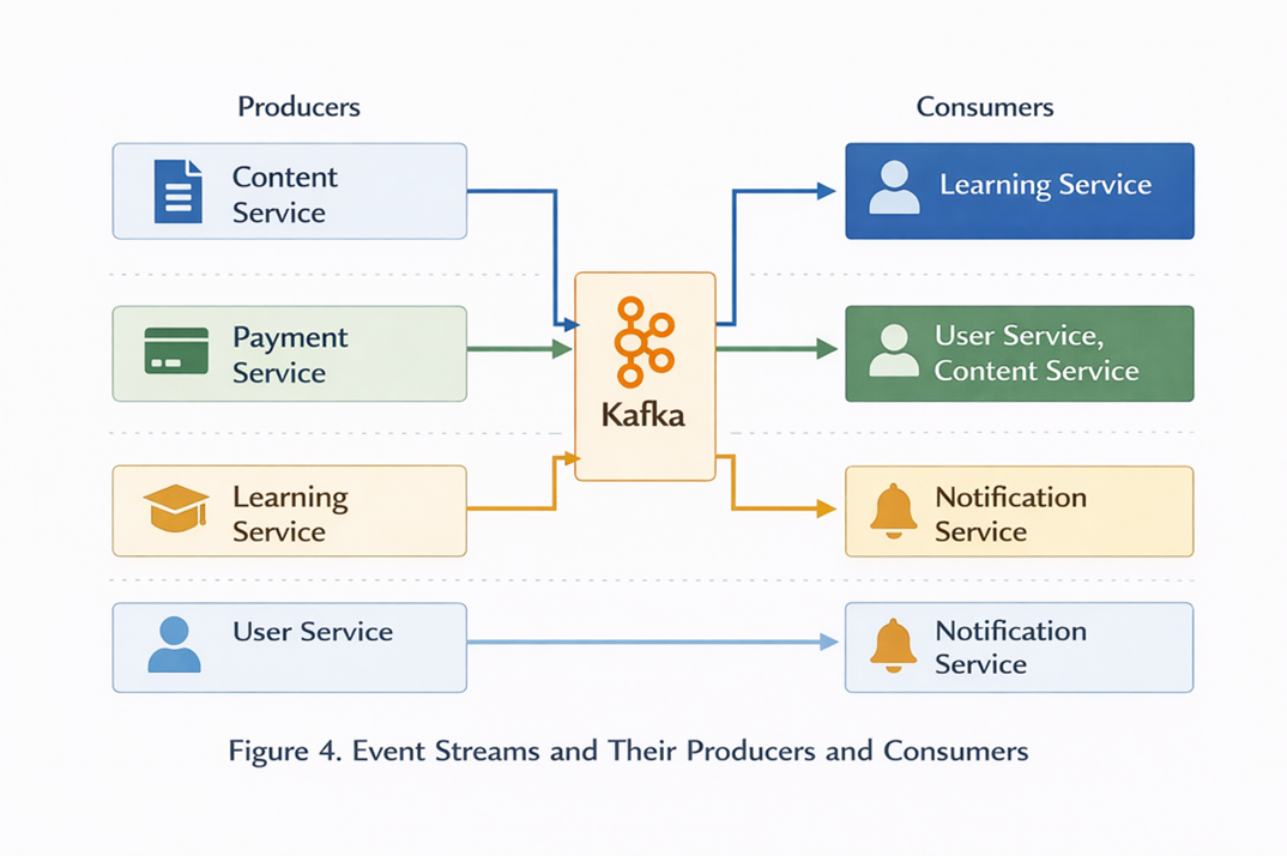

Figure 4. Event Streams and Their Producers and Consumers

5. **Scalability and Resilience Mechanisms**

The proposed architecture improves scalability by introducing a strict separation between the command path and the reaction path. User-facing requests remain lean because they do not directly coordinate every downstream effect. Instead, the system turns domain changes into durable event streams that can be processed by multiple independent consumer groups. As a result, learning projections, search indexing, analytics enrichment, and notification delivery can scale according to their own workload profiles rather than being constrained by the throughput of the original write service.

The architecture also improves resilience. If a downstream consumer is temporarily unavailable, the originating service can still complete the local transaction and publish an event. The delayed consumer can later catch up by replaying the stream. This is significantly safer than requiring the original request to remain open while every dependent service performs its work. Event retention and replay make recovery from temporary failures both faster and more predictable.

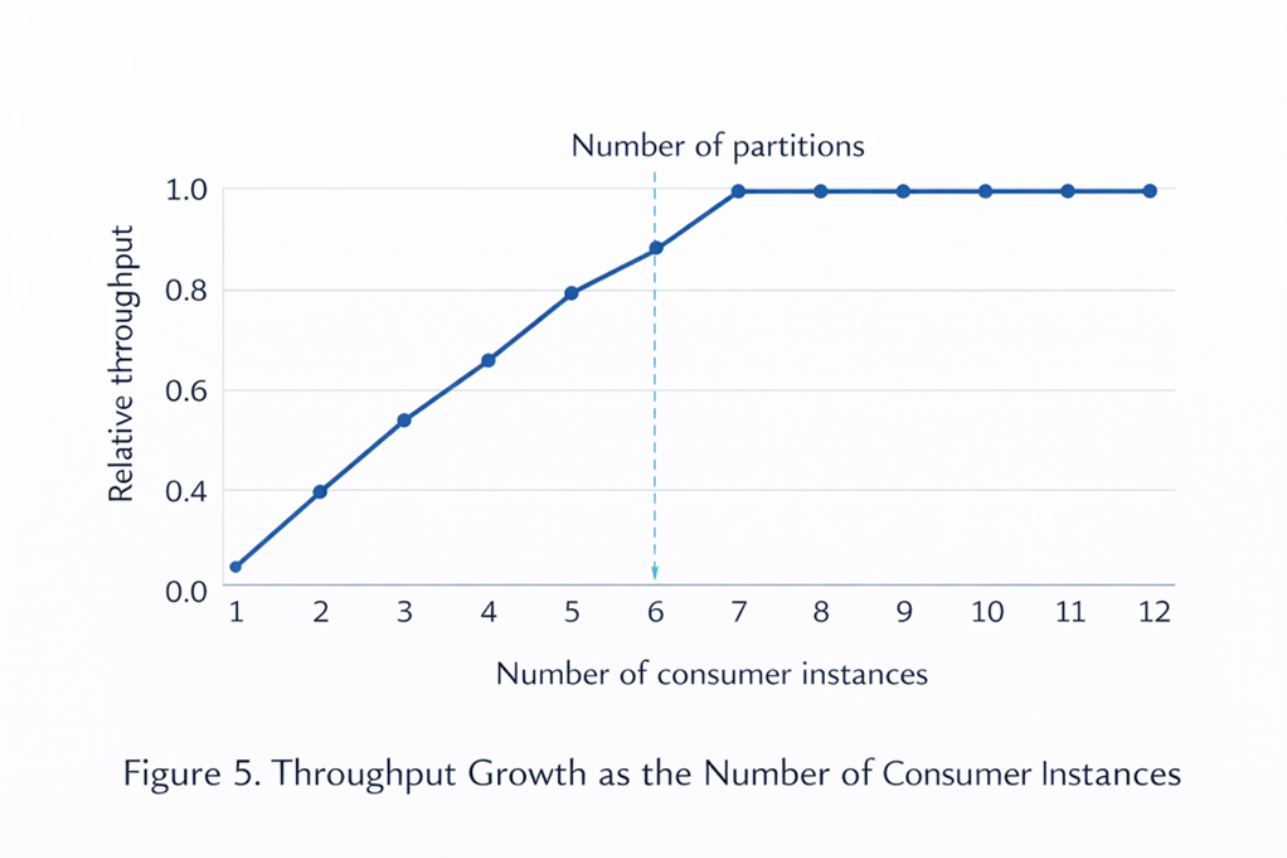

Figure 5. Throughput Growth as the Number of Consumer Instances

Idempotency is a critical property in this model. Consumers must assume that a message may be redelivered. Reminder workflows in Learnly already reflect this principle through unique idempotency keys and duplicate-suppression mechanisms. Similar protection should be applied consistently across other critical streams. Contract versioning, retry strategies, dead-letter handling, and consumer lag monitoring are equally important for sustained operation in production environments.

6. Architectural Evolution Path

Although Learnly already demonstrates strong event-oriented architectural characteristics, further evolution can strengthen the platform considerably. A particularly valuable next step is the adoption of a more explicit outbox or change-data-capture pattern for domains in which a database state transition and an emitted event must be treated as one reliable business fact. This reduces the risk of divergence between local persistence and event publication during partial failures.

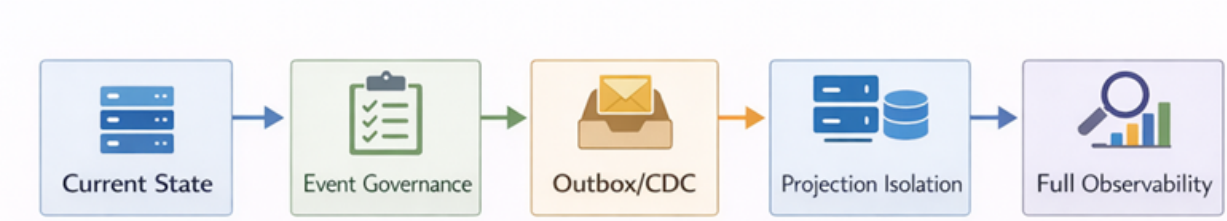

Figure 6. Evolution Roadmap of the Learnly Event-Driven Architecture

Another important direction is the formalization of event governance. Naming conventions, schema compatibility policies, keying rules, and replay expectations should be documented and enforced uniformly across domains. A mature operational model should also include targeted observability: topic lag dashboards, retry and dead-letter volume tracking, producer error monitoring, contract validation in continuous integration, and business-level propagation metrics. For a spaced repetition platform, propagation delay is itself a product metric, because stale deck state or late reminders directly affect learning quality.

7. Discussion

The architectural value of the event-driven model is especially visible in educational systems because the platform must continuously reconcile authoring, learning, communication, entitlement, and discovery concerns. Learnly can be understood not merely as a flashcard application, but as a projection-rich educational event mesh in which domain events function as the connective tissue of the product. This is what allows the platform to remain responsive at the edge, expressive in the core domain, and extensible in its downstream intelligence.

In practical terms, the architecture elevates Learnly from a set of backend services to a coordinated learning infrastructure. Content becomes not only editable but distributable across bounded contexts. Learning state becomes not only persistent but analytically exploitable. Subscription state becomes not only billable but locally actionable in multiple domains. Notifications become not only deliverable but replay-safe and policy-driven. These characteristics make the platform well suited for growth into a larger ecosystem of adaptive learning services.

8. Conclusion

This paper presented the architecture of a scalable spaced repetition system based on an event-driven model, using Learnly as a concrete architectural reference. The study demonstrated that a combination of microservices, asynchronous domain events, schema-governed messaging, projection-oriented read models, and specialized persistence technologies forms a strong foundation for a modern educational platform.

The proposed model supports low-latency user interactions while enabling independent scaling of secondary workloads such as reminder delivery, search indexing, entitlement propagation, and analytics. It reduces service coupling, improves resilience to partial failures, and provides a natural framework for replay-safe distributed processing. For spaced repetition systems in particular, these properties are essential because learning quality depends on the timely coordination of many small but interconnected domain transitions.

Learnly illustrates how an educational platform can evolve toward a resilient event-driven architecture without losing domain clarity. With further investment in outbox reliability, event governance, projection isolation, and operational observability, such a platform can scale not only technically but also as a foundation for richer adaptive learning experiences.

ҚМ АА Куәлік нөмірі: **KZ45VPY00102718** — ҚР Мәдениет және Ақпарат министрлігі