Назарова Виктория Олеговна, студент
Санкт-Петербургский политехнический университет Петра Великого
Nazarova Viktoria Olegovna**,** student
Peter the Great St. Petersburg Polytechnic University

ВЫСОКОДОСТУПНЫЕ API-ПЛАТФОРМЫ КАК ОСНОВА УСТОЙЧИВОСТИ ФИНТЕХ-ПРИЛОЖЕНИЙ
HIGHLY AVAILABLE API PLATFORMS AS A BASIS FOR FINTECH APPLICATION RESILIENCE

Аннотация. Финтех-приложения используют программные интерфейсы для проведения платежей, доступа к счетам и обмена данными. В статье рассмотрены архитектурные условия сохранения работоспособности API-платформ при росте нагрузки, инфраструктурных сбоях и отказах зависимых сервисов. На основе актуальных исследований и открытой статистики проанализированы резервирование, изоляция отказов, ограниченные повторные запросы, идемпотентность и наблюдаемость. Предложена концептуальная архитектура и систематизированы показатели проверки доступности, времени восстановления, задержек и корректности транзакций. Установлено, что масштабирование вычислительных компонентов не обеспечивает устойчивость без контроля общего состояния и проверки завершения пользовательских операций.

Abstract. Financial applications depend on application programming interfaces (APIs) for payment initiation, account access, and data exchange. This review examines how API-platform architecture supports service continuity under load spikes, infrastructure faults, and failures of downstream components. The analysis draws on recent research and published open-banking performance indicators. It distinguishes redundancy at the request-processing layer from transaction correctness at the state layer and evaluates failure isolation, bounded retries, idempotency, and observability. A reference architecture is presented, together with criteria for testing availability, recovery, tail latency, and transaction integrity. The analysis indicates that replicas and autoscaling alone do not guarantee resilience: the outcome depends on the treatment of shared dependencies and the verification of complete user transactions.

Ключевые слова: API-платформы, финтех-приложения, высокая доступность, изоляция отказов, идемпотентность, распределённые системы, операционная устойчивость

Keywords: API platforms, fintech applications, high availability, fault isolation, idempotency, distributed systems, operational resilience.

Introduction

Application programming interfaces (APIs) increasingly connect customer channels, financial institutions, and external service providers. UK open-banking participants recorded 24.0 billion successful API calls in 2025, a 27% increase over 2024; weighted availability remained above 99.50%, while mean response time reached 324 ms [1]. These figures describe the UK ecosystem rather than an individual fintech platform, but they illustrate the scale at which reliability must be maintained. Availability is not equivalent to the presence of several service replicas. An API may remain reachable while a payment is processed twice, a downstream service times out, or a database cannot confirm a transaction. The EU Digital Operational Resilience Act (DORA), applicable since 17 January 2025, also places operational risk management, testing, and incident response within the governance of financial entities [2].

The aim of this review is to identify architectural mechanisms that protect fintech API operations from partial failures and to establish measurable criteria for evaluating their effectiveness.

Architectural boundaries of API availability

A resilient API platform separates external traffic admission from business processing and persistence. The architectural literature describes health-checked load balancing, API gateway

controls, and service replication across failure zones as complementary ways to preserve request processing under partial infrastructure failures [3]. A banking middleware study published in 2026 found improvements in latency, throughput, and recovery after a microservice redesign, although those results are specific to its tested environment [4]. Service decomposition permits different scaling policies for account queries and payment execution. Asynchronous operations can absorb short bursts, provided that queue growth, delivery semantics, and consumer capacity remain controlled [5].

A conceptual arrangement of these responsibilities is shown in Fig. 1.

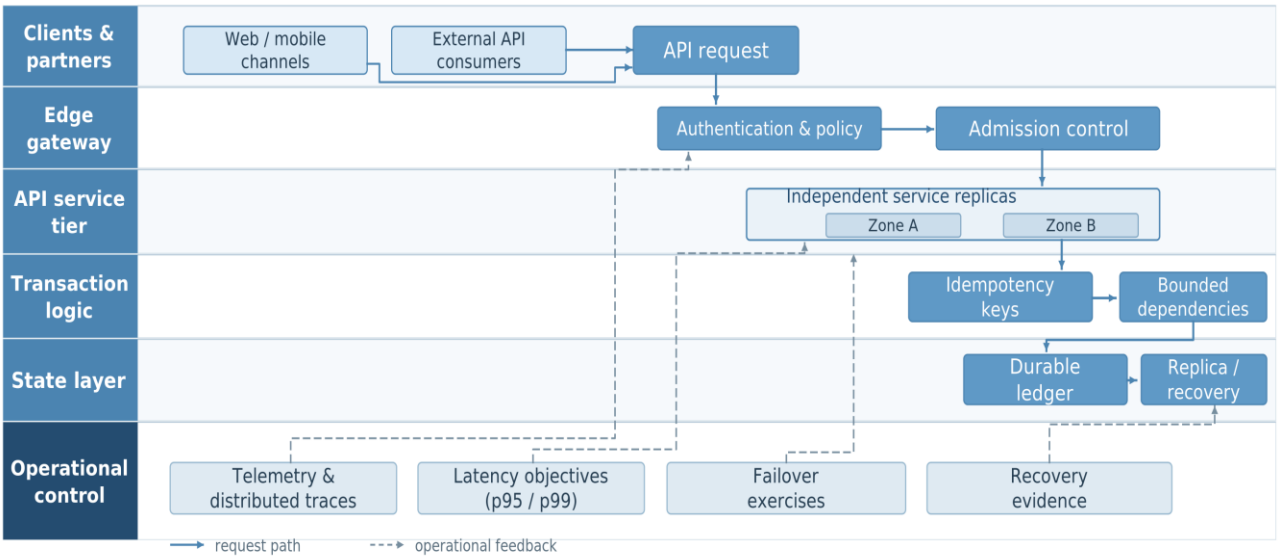

Fig. 1. Functional separation and operational controls in a highly available fintech API platform

The diagram separates request admission and stateless execution from the durable transaction state. Traffic can be redirected between processing zones without assuming that concurrent regions may modify the same ledger without coordination. This distinction is essential when a temporary increase in availability could otherwise introduce inconsistent financial records. The lower control layer denotes operating practices rather than another path for customer requests. Telemetry and failover exercises are shown across component boundaries because availability must be demonstrated for an end-to-end operation, including authorization, durable recording, and an unambiguous response to the client.

Failure containment and transaction correctness

Replication addresses server loss but does not prevent retry storms or the propagation of dependency failures. Fintech backends can employ circuit breakers, backpressure, bounded timeouts, and deduplication so that the failure of a nonessential integration does not exhaust the payment-processing tier [6]. Retries of money-moving requests require stable operation identifiers and durable idempotency records; a successful transport response is not sufficient evidence of exactly-once business execution. Consistency requirements introduce a separate limitation. When replicated state requires agreement, quorum membership and network latency constrain commit throughput; the choice of consensus protocol must reflect the fault model and acceptable finality guarantees [7]. Strong agreement should be reserved for state transitions requiring it rather than inserted indiscriminately into every read-only API call. Host-level optimization remains relevant at the edge of this architecture. CPU isolation, interrupt placement, and memory locality can reduce latency variability in particularly demanding Linux workloads [8]. Techniques described for high-frequency trading should not be assumed to yield the same improvement in ordinary payment services without workload-specific measurements. Similarly, protocol selection among REST, GraphQL, and gRPC changes communication overhead but cannot compensate for an overloaded shared data store [9].

Verification of end-to-end resilience

Operational resilience must be assessed against the behaviour seen by payment clients, not simply instance uptime. User-interface responsiveness can deteriorate even when back-end requests remain within their latency target, especially where blocking scripts and inefficient rendering delay the completion of a customer journey [10]. The complementary validation criteria are summarized in Table 1.

Table 1

Indicators and fault-injection scenarios for fintech API resilience

Criterion	**Observable indicator**	**Validation scenario**
Request continuity	Successful critical-journey responses	Withdraw a gateway or API-service zone while maintaining offered traffic
Transaction integrity	Duplicate commits and unresolved payment states	Replay requests with the same idempotency key during a timeout
Tail latency	End-to-end p95/p99 response time	Inject dependency delay and a short traffic burst
Recovery	Restoration time and reconciliation backlog	Interrupt a stateful dependency, restore it, and verify records
Client completion	Payment-flow completion rate	Introduce slow networks and frontend resource contention

Table 1 distinguishes continuity of API responses from correctness of financial outcomes. A service can pass a regional failover test while still violating transaction integrity if a repeated payment is committed without deduplication. Joint observation of the two indicators avoids interpreting infrastructure redundancy as proof of payment safety. Latency percentiles and recovery metrics capture different failure modes. The use of p95/p99 values exposes periods of degraded service concealed by average response times, while reconciliation checks identify cases in which an incident has ended technically but left the customer transaction unresolved. These indicators should be evaluated under declared loads and fault assumptions rather than treated as universal thresholds. Failure testing also needs to be reflected in release procedures. Gradual deployment, rollback readiness, tracing, and periodic failover tests turn architectural provisions into evidence of operational behaviour. Such evidence is pertinent to a fintech platform whose software changes, third-party integrations, and transaction mix evolve throughout its service life.

Conclusion

Highly available API platforms provide a foundation for fintech resilience only when request continuity is accompanied by transaction correctness and demonstrable recovery. Separating traffic admission, replicated processing, and durable financial state limits fault propagation, while bounded dependencies and idempotency protect business operations during retries and partial failures. The proposed evaluation criteria connect architectural decisions with observable behaviour under load and deliberate fault injection. The review is limited to published studies and a conceptual reference architecture; comparative deployment experiments remain necessary to establish the performance gains and operating costs of particular configurations

Список литературы:

1. Open Banking Limited. Open Banking in 2025: Now Part of the UK’s Everyday Financial Life. 29 January 2026. URL: https://www.openbanking.org.uk/insights/open-banking-in-2025-now-part-of-the-uks-everyday-financial-life/

2. European Insurance and Occupational Pensions Authority. Digital Operational Resilience Act (DORA). URL: https://www.eiopa.europa.eu/digital-operational-resilience-act-dora_en

3. Kovalevskyi K. Architectural approaches to building highly available API platforms for fintech applications // Cold Science. 2026. №25. P. 13-24.

4. Fauziah R., Surantha N. Evaluating middleware performance in the transition from monolithic to microservices architecture for banking applications // Electronics. 2026. Vol. 15. № 1. P. 221. DOI: 10.3390/electronics15010221.

5. Rakhmatullayev O. Modern software development methods for high-load systems // Professional Bulletin: Information Technology and Security. 2024. №1. P. 20-23.

6. Kovalenko A. Architectural and algorithmic methods for enhancing the resilience of high-load backend services in the financial sector // Norwegian Journal of development of the International Science. 2025. №158. P. 87-91.

7. Kovalevskyi K. Study of the relationship between consensus algorithms and the scalability of distributed computing environments // International independent scientific journal. 2026. №84. P. 26-31.

8. Otkidach I. Architectural and system-level aspects of linux environment optimization for high-frequency trading systems // Professional Bulletin: Information Technology and Security. 2026. №1. P. 36-42.

9. Niswar M., Arisandy Safruddin R., Bustamin A., Aswad I. Performance evaluation of microservices communication with REST, GraphQL, and gRPC // International Journal of Electronics and Telecommunications. 2024. Vol. 70. № 2. P. 429-436.

10. Meshcheryakov V. Optimization of user interface performance as a factor in digital product resilience // Professional Bulletin: Information Technology and Security. 2026. №2. P. 10-15