DOI: https://doi.org/10.36719/2789-6919/56/173-177

Kamil Aliyev
Odlar Yurdu University
Master's student
https://orcid.org/0009-0004-9060-9086
kamil.aliyev.cs@gmail.com

Design and Experimental Evaluation of a Microservice-Based Architecture for University Workflow Automation

Abstract

In modern universities, there exist complex information systems that support academic, administrative, and financial operations. It has been realized that monolithic architecture faces difficulties in achieving scalability and flexibility in these information systems. This research aims at investigating and evaluating the microservices architecture in designing an information system that can support automation in universities. A prototype information system was developed using ASP.NET Core and an API Gateway, and it was divided into four microservices: Student, Course, Enrollment, and Notification. From the experiment conducted, comparing monolithic and microservices architecture, it was realized that although there is an increase in latency due to communication between distributed services, there is an improvement in modularity and error isolation.

***Keywords*:** *microservice architecture; university information systems; workflow automation; domain-driven design; API gateway; distributed systems*

Kamil Əliyev
Odlar Yurdu Universiteti
magistrant
https://orcid.org/0009-0004-9060-9086
kamil.aliyev.cs@gmail.com

Universitet iş axınının avtomatlaşdırılması üçün mikroservis əsaslı arxitekturanın dizaynı və eksperimental qiymətləndirilməsi

Xülasə

Müasir universitetlərdə akademik, inzibati və maliyyə əməliyyatlarını dəstəkləyən mürəkkəb informasiya sistemləri mövcuddur. Monolit arxitekturanın bu informasiya sistemlərində miqyaslılıq və çevikliyə nail olmaqda çətinliklərlə üzləşdiyi məlum olub. Bu tədqiqatın məqsədi universitetlərdə avtomatlaşdırmanı dəstəkləyə bilən informasiya sisteminin dizaynında mikroservis arxitekturasını araşdırmaq və qiymətləndirməkdir. ASP.NET Core və API Gateway istifadə edərək prototip informasiya sistemi hazırlanmışdır və o, dörd mikroservisə bölünmüşdür: Tələbə, Kurs, Qeydiyyat və Bildiriş. Monolit və mikroservis arxitekturasını müqayisə edərək aparılan təcrübədən məlum olub ki, paylanmış xidmətlər arasında rabitə səbəbindən gecikmədə artım olsa da, modulluq və səhv izolyasiyasında yaxşılaşma var.

Açar sözlər: *mikroservis arxitekturası, universitet informasiya sistemləri, iş axınının Avtomat-laşdırılması, domen yönümlü dizayn, API gateway, paylanmış sistemlər*

Introduction

The digital transformation has profoundly changed the operational architecture of modern universities. A university is a complex system with multiple processes, including student enrollment, managing courses, developing curricula, financial management, and administrative reporting. These processes are often automated by heterogeneous information systems, which were developed at different times and with different technologies. This has led to data inconsistencies and operational inefficiencies.

In the past, information systems in universities were often developed using a monolithic architecture, where all functional components operate in a single application and share a common database. This is a simple and effective way to develop information systems, but it is difficult to maintain, especially when the system becomes complex.

The microservice architecture has been identified as a new alternative for developing complex information systems. This study aims to transform a university workflow automation system from a monolithic architecture to a microservice architecture and experimentally evaluate the system.

Research

Research Contributions. The main results of the study are:

- Design of a domain-oriented microservice architecture for automating university workflows;
- Implementation of Monolithic and Microservice Prototypes Using Asp.Net Core;
- Comparative Performance Assessment of the Two Architectures;
- Experimental demonstration of fault isolation in a distributed workflow environment.

The results provide practical insights into the advantages and disadvantages of implementing microservices in complex information systems such as university management platforms.

Research method. The research methodology used in this study combines *system analysis, architectural modeling, prototype implementation, and experimental evaluation*. The goal is to investigate how microservices architecture can improve the modularity and reliability of university workflow systems, while analyzing the associated performance tradeoffs.

System Analysis. Universities are complex sociotechnical systems consisting of multiple interconnected functional domains. Key processes include student enrollment, course management, enrollment processing, financial transactions, academic assessment, and administrative documentation.

A functional analysis of typical university information systems has identified several key domains involved in automating academic workflows. These domains include:

- Student management;
- Course management;
- Enrollment processing;
- Notification and communication services.

The interactions between these domains form the basis for the workflow model analyzed in this study.

Monolithic Baseline Implementation. To establish a benchmark, we implemented a monolithic version of the university workflow system using ASP.NET Core Web API and a single relational database. This monolithic implementation integrates student registration, course management, course selection processing, and audit logging into a single deployable system.

All functional modules share the same database and runtime environment. Although this architecture simplifies initial development, it introduces strong coupling between modules and limits the ability to independently scale or deploy individual system components.

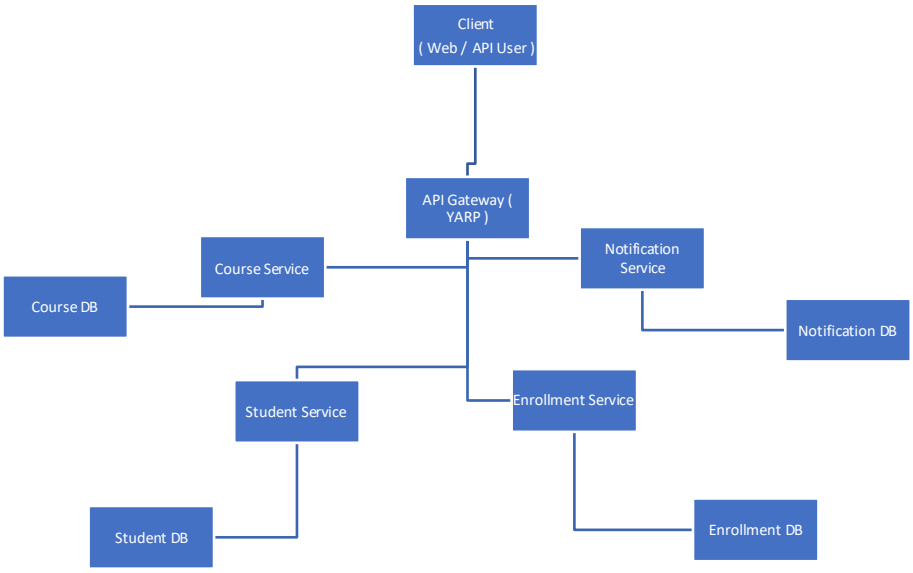
Microservice Architecture Design. Based on the principles of *Domain-Driven Design (DDD),* the system was decomposed into four independent microservices:

- Student Service – manages student records and identity data;
- Course Service – manages course information and academic offerings;
- Enrollment Service – orchestrates the enrollment workflow;

- Notification Service – records workflow notifications.

Each service maintains its own independent database following the *database-per-service* principle.

Figure 1. Overview of the proposed microservice-based architecture for university workflow automation.



An *API Gateway* was introduced to provide a unified entry point for client requests and to route requests to the appropriate services. The gateway was implemented using the *YARP reverse proxy framework*.

Communication between services is performed using REST APIs over HTTP. During enrollment creation, the Enrollment Service validates the existence of students and courses through service-to-service REST calls.

Experimental Setup. Two experiments were conducted to evaluate the architectural advantages and disadvantages between monolithic architecture and microservice applications.

Performance Experiment. Its record creation latency was measured in two architectures. The experiment is conducted by executing multiple HTTP operations and logging their responses.

Fault Isolation Experiment. To assess system resilience, the Notification Service was intentionally disabled while registration confirmation requests were being processed. The aim was to observe whether failures in secondary services would disrupt the primary workflow.

The performance evaluation compared the response latency of the Enrollment Creation operation in both architectures.

Monolithic implementation achieved an average steady-state response time of approximately 146 ms, while the microservice application achieved an average response time of approximately 357 ms. This difference reflects the overhead introduced by distributed communication, gateway routing, and inter-service REST interactions (Taibi et al., 2018; Richards, 2015; Lewis & Fowler, 2014).

Table 1.
Response Latency Comparison Between Monolithic and Microservice Architectures

Request No.	Monolithic (ms)	Microservices (ms)
1	772	2520
2	169	880
3	128	314
4	96	277
5	183	298
6	160	242
7	157	325
8	160	273
9	158	292
10	103	310

Table 2.
Average Response Time Across All Requests

Metric	Monolithic	Microservices
Average (all requests)	208.6 ms	573.1 ms
Average (excluding first request)	146.0 ms	356.8 ms

Significant differences between the initial request and subsequent requests indicate that the cold start effect has a measurable influence on the two architectures, especially on the distributed microservice-based approach.

The microservice architecture introduced approximately 211 ms additional latency compared to the monolithic baseline.

Fault isolation experiment. A fault isolation experiment was performed by disabling the Notification Service during the registration approval process. After implementing resilience mechanisms to the Registration Service, the approval process was successfully completed even with the Notification Service unavailable.

The system logged the notification error to the audit log while allowing the primary workflow to continue. This behavior demonstrates the ability of microservice architectures to isolate errors in secondary services without disrupting core system operations.

Analysis and discussion. Based on the experimental results, a clear trade-off can be observed between architectural modularity and runtime efficiency. The microservice-based implementation demonstrated higher average response latency than the monolithic implementation due to inter-service communication and API gateway routing. However, this additional communication overhead provides several architectural advantages, including independent service deployment, clearer service boundaries, improved maintainability, and better fault isolation.

The fault isolation experiment further highlights these advantages. In this experiment, the Notification Service was intentionally disabled in order to evaluate whether the core business process (Approval Workflow) could still be completed by the Enrollment Service. After introducing resilience handling, the Enrollment Service successfully completed the approval workflow (Aliyev, 2026; Thönes, 2015; Newman, 2021; Evans, 2003; Fowler, 2014). The communication failure with the Notification Service was recorded in the audit log without interrupting the primary business process. This behaviour demonstrates the capability of microservice architectures to support graceful degradation when secondary services fail.

Another observation from the experiment is that the first request exhibited significantly higher response time compared to subsequent requests. This behaviour can be explained by cold-start effects

such as framework initialization, JIT compilation, database connection establishment, and HTTP pipeline warm-up. Therefore, performance measurements were analysed both including and excluding the first request (Dragoni et al., 2017; Zhang et al., 2018; Zhang et al., 2019; Richardson, 2018; Aliyev, 2026).

Overall, the results suggest that monolithic architectures remain advantageous in scenarios where low latency and minimal operational complexity are the primary requirements. In contrast, microservice architectures are more suitable for complex environments where scalability, service autonomy, and system resilience are more important than minimizing request latency.

Conclusion

This study examined the design and practical evaluation of a microservice-based architecture for university business automation. The results demonstrate significant characteristics of microservice architectures in terms of modularity, service independence, and fault isolation.

The experimental evaluation reveals that microservices cause additional latency compared to monolithic architectures due to distributed communication overhead. However, the health and maintainability of the architectural systems show significant improvement.

The fault experiment shows that, with appropriate resilience, failures in comprehensive services can be handled without disrupting critical business isolations.

Overall, microservice architectures represent a viable and scalable approach and scalable approach for modern university information systems. Future reviews could explore advanced integration models such as event-driven communication, messaging tools, and distributed transaction management technologies to further improve system changeover and reliability.

References

1. Aliyev, K. (2026). *Monolith University Workflow*. GitHub repository. Available: https://github.com/kamilali01/Monolith-University-Workflow
2. Aliyev, K. (2026). *University Workflow Automation – Microservice Architecture.* GitHub repository. Available: https://github.com/kamilali01/University-Workflow-Automation---Microservice-Architecture
3. Dragoni, N. et al. (2017). *Microservices: Yesterday, Today, and Tomorrow*. In: Present and Ulterior Software Engineering. Springer.
4. Evans, E. (2003). *Domain-driven design: Tackling complexity in the heart of software.* Addison-Wesley.
5. Fowler, M. (2014). *Microservices: A definition of this new architectural term.* Available: https://martinfowler.com/articles/microservices.html
6. Lewis, J. & Fowler, M. (2014). *Microservices: A definition of this new architectural term.* Available: https://martinfowler.com/articles/microservices.html
7. Newman, S. (2021). *Building microservices: Designing fine-grained systems* (2nd ed.). O'Reilly Media.
8. Richards, M. (2015). *Microservices vs. service-oriented architecture.* O'Reilly Media.
9. Richardson, C. (2018). *Microservices patterns: With examples in Java.* Manning Publications.
10. Taibi, D., Lenarduzzi, V. & Pahl, C. (2018). *Architectural patterns for microservices: A systematic mapping study.* IEEE Cloud Computing.
11. Thönes, J. (2015). Microservices. *IEEE Software, 32*(1), 116–116.
12. Zhang, Q., Chen, M. & Li, L. (2019). Microservice architecture in cloud-based systems: A systematic literature review. *Journal of Systems and Software*.
13. Zhang, Y., Chen, H. & Li, H. (2018). Service decomposition in microservice architectures. *IEEE International Conference on Software Architecture*.

Received: 29.11.2025
Approved: 02.03.2026