УДК 004.4

Y. A. Palamarchuk

METHODS OF BUILDING MICROSERVICE ARCHITECTURE OF E-LEARNING SYSTEMS

Vinnytsia National Technical University, Vinnytsia

Abstract. The basic principles of organization of microservice architectures (MSA), their parameters, functioning and application in electronic learning systems (ELS) are considered. A comparative analysis of MSA construction options, their components and methods was done. A complex method of building e-learning systems based on MSA is proposed. The microservice architecture of the electronic educational system of Vinnytsia National Technical University JetIQ has been developed and implemented. The expediency of using a domain organization for individual subsystems of ELS is substantiated.

Keywords: e-learning system, distance learning, microservice architecture, domain architecture, exchange protocol.

Анотація. Розглянуто основні принципи організації мікросервісних архітектур (МСА), їх параметрів, функціонування та можливості застосування у електронних навчальних системах (ЕНС). Проведений порівняльний аналіз варіантів побудови МСА, їх компонентів та методів. Запропоновано комплексний метод побудови електронних навчальних систем на основі МСА. Розроблено і реалізовано мікросервісну архітектуру електронної навчальної системи Вінницького національного технічного університету JetIQ. Обгрунтовано доцільність використання доменної організації для окремих підсистем ЕНС.

Ключові слова: електронна навчальна система, дистанційне навчання, мікросервісна архітектура, доменна архітектура, протокол обміну.

DOI: https://doi.org/10.31649/1999-9941-2022-53-1-43-54.

Introduction

The use of e-learning systems (ELS) allows to solve a number of important problems. The main one is the possibility of organizing distance learning (DL). DL provides ample opportunities to organize this process without geographical and temporal reference of participants (students and teachers), to organize asynchronous learning.

Depending on the set educational tasks, its architecture and software decisions are formed. These tasks also determine the technical characteristics and requirements for their structure, data types, logic of operation, computer and network requirements, etc. [1,2].

Most of the existing ENS have a monolithic architecture [3-14]. This limits their ability to make software changes or enhance functionality without rebuilding the ENS kernel. The consequence of this is the problem of scaling up and efficient use of cloud technologies and, in particular, the creation of geographically distributed architectures. However, such systems can be scaled only horizontally. It involves the use of more powerful machines, or the use of additional servers to maintain the same core system.

Emergencies with a monolithic core do not allow to increase their noise immunity by separating individual high-load subsystems.

The aim of the study is to develop a comprehensive methodology for building scalable and noise-tolerant ELS based on microservice architectures.

Features of application of monolithic and MSA architectures in ELS

ELS parameters are determined not only by the functionality and software tools embedded in them, but also by the architecture. Most ELS s with a relatively developed core use monolithic structures. This gives them a number of advantages and good dynamics of development and support.

An alternative to monolithic systems is the decomposition of the entire ELS into separate microservices and thus the creation of a service-oriented architecture (SOA) with distributed resources, where the exchange of information between its components is carried out according to certain network protocols.

MSA provides a number of advantages, namely:

- Ease of implementation of microservices. Their programs have relatively small program code and they are easy to maintain and manage.
- Scalability and flexibility. Microservices can be implemented in different programming languages, which makes it possible to optimize their implementation.
- Independence of operation provides an opportunity to increase the reliability of the MSA as a whole.
- High quality of individual MSA components. They are easy to develop and test, reminiscent of the UNIX philosophy [15].
- Focus on business functionality. This means that microservices can be used in several contexts, in more than one business process [16].
- Flexibility in system load balancing. MSA provides the ability to manage the load of its microservices and their individual scaling.

Restrictions on the use of microservice architecture

1. Organization of interaction of microservices in the whole system. It correlates with the number of processes involved. MSA has many more interoperable parts than monolithic architectures. This requires additional effort in planning, automation, control, monitoring, testing and deployment.

2. Costs. Microservices require the use of a larger total amount of memory than in a monolithic architecture, due to the need to organize isolated environments, as well as the cost of organizing communications between them (interfaces, protocols, communication channels, etc.) .

3. Security issues. Microservices require security solutions due to the availability of inter-service communication over computer networks.

4. Decrease in productivity in comparison with monolithic systems. The reason is the occurrence of additional delays in the exchange of data between microservices. Therefore, it needs to solve problems with the correct design of queues, asynchronous processing and balancing.

5. Problems of inconsistency. Unlike monolithic systems, in MSA errors when updating individual microservices can lead to a violation of desynchronization.

Microservice Architecture

Consider the construction of MSA. In essence, a microservice is a process, function, or set of functions that accepts incoming requests and performs certain actions. Interaction of microservices with the environment takes place through API (Application Programming Interface) and computer networks. Microservices are structured in tiers. The number of their purposes is determined by the architecture of the entire system. The most common are:

- Front-End - Client Application, Static Content
- Back-End - API Gateway, Experience Microservices, Domain Microservices
- Data & Integration - Data Stores, Integration Interfaces / Connectors

Logical tiers

Client program

In the case of web-based ELS, the client is a web browser or mobile application. The client (web browser) interacts with Backend to render the UI and call the API (fig. 1,2).

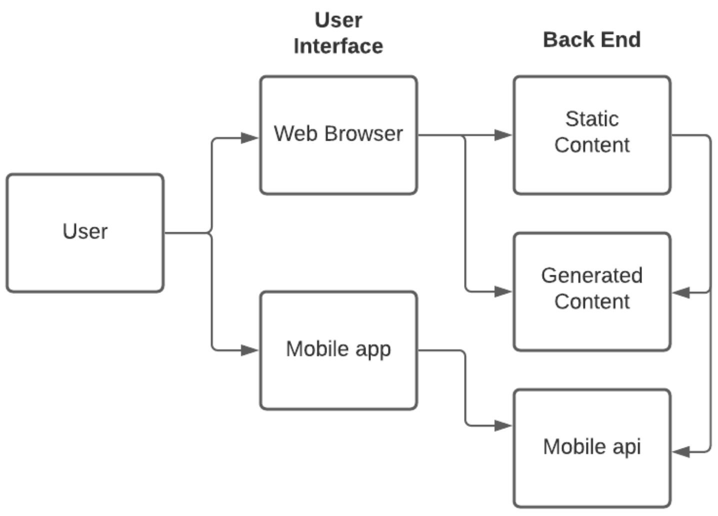

Figure 1 – General scheme of client-server architecture

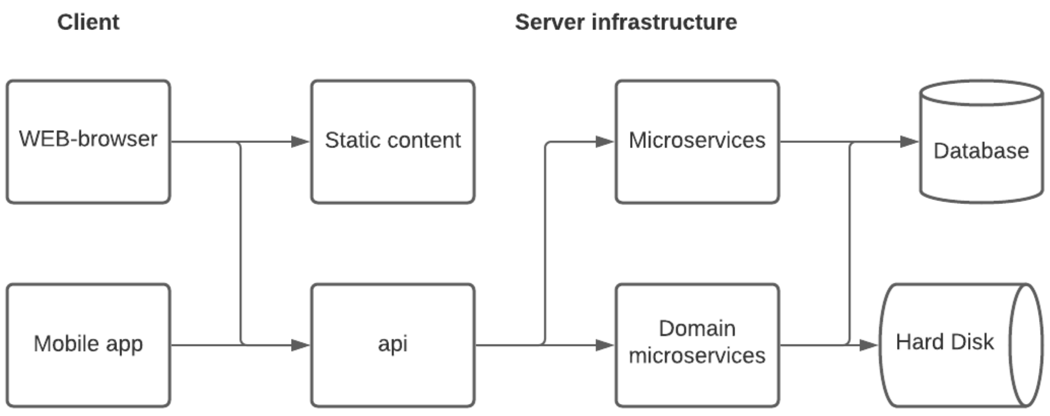

Figure 2 – Client-server architecture with static and dynamic content

For efficient operation, the responsibility for static components (UI) and logical components is usually divided. The latter also have problems with access control, caching and server-side rendering.

Gateway API

In ELS API Gateway represents a single point of access to Experience APIs (client modules). API Gateway is an important component of system management (API Management). This service sends requests backend service and is responsible for the following functionality:

- Access control
- Security Policies
- Request handling
- Caching
- Upstream / Downstream data transfer
- Rate limiting
- Throttling
- Analytics

Experience microservices

In ELS with a developed ecosystem, the number of microservices can be quite large, and it is important to maintain guaranteed control over them. In this case, tiered structuring is used. For ELS, the top tier is the Experience tier, which houses the micro-services of the web API, mobile-API and API of other external systems.

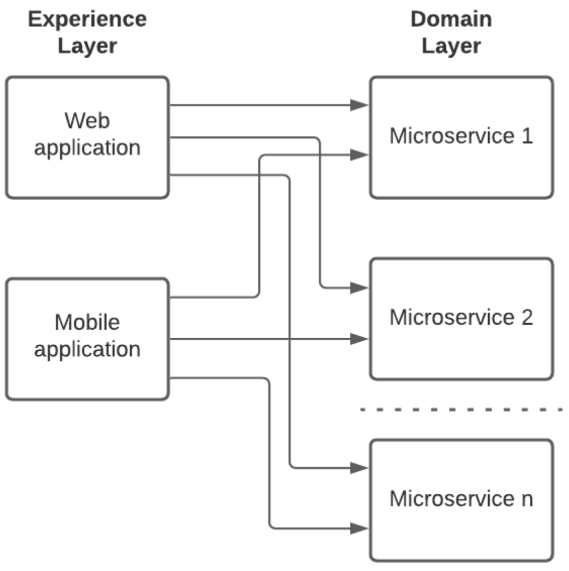

Figure 3 − Experience Layer

They focus on client interfaces and data using the low-level granular APIs available to them on the system. Thus, different products for web and mobile clients (Experience Layer) can use different sets of Experience-API, represented by independent microservices on the API gateway.

Domain microservices

Domain microservices in ELS are used to implement the appropriate functionality, such as authentication subsystems, business logic. storage and access to training materials, testing, etc. Individual subsystems can be represented by several microservices. Domain microservices organize granular APIs next to the experience tier. If experience services are focused on users and products, then domain microservices are responsible for the data and resources under their control. They provide the ability to create different experience services that can reuse the domain logic available to them.

Databases

Databases in ELS are a well-developed structure. But access to them is usually provided only to one microservice. All other microservices have access to the database only through the database owner microservice API. This approach allows you to maintain the consistency and structure of the database. In addition, it allows the use of the most appropriate types of databases for specific ELS purposes. For example, one microservice may use a normalized SQL database and another a NoSQL database. It is assumed that one microservice can operate multiple databases simultaneously (Fig. 4).

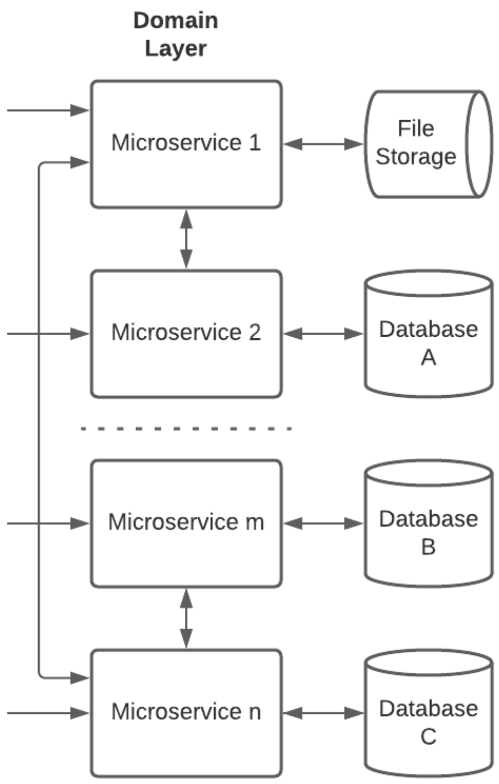

Figure 4 − Database Domain Layer

Integration interfaces and connectors

The MSA ELS can use an integration tier with microservice connectors to access other subsystems and third-party APIs.

Interservice communication

Microservices must process requests and provide the possibility of interaction according to the interprocess communication protocol. The interaction of microservices during the execution of requests should not affect their availability during the execution of requests. The solution is asynchronous messaging in MSA. The main types of inter-service communication of MSA components can be represented by two types: synchronous and asynchronous. Depending on requirements, the following variants are actual:

- synchronous blocking
- synchronous non blocking

- asynchronous blocking
- asynchronous non blocking

Their implementation is presented as follows:

- request / asynchronous response - exchange queries with pending response
- publish / asynchronous response - publishing a request to recipients waiting for answers
- request / response - exchange queries with pending immediate response
- notifications - send a message without waiting for a response.
- publish / subscribe - publishing a message to recipients

Interaction of components in MSA

The implementation of inter-service communication is also based on the use of certain architectural solutions and network protocols, both high and low level.

The basic protocols for the implementation of high-level API of inter-service communications are the architecture REST [17-19] with HTTP and GraphQL.

REST-approach involves the management of a particular system service resources of other components of the system. When performing requests, the microservice operates with different instances of this component, for example, changing its properties and parameters or relaying them for further processing to other microservices.

The HTTP specification [20--22] contains a number of features that cover most of the needs for the implementation of cross-server communication based on REST. Thus, the HTTP protocol has methods GET, POST and PUT, which already allow to fully implement the interaction in the system. Typically, MSA data formats for inter-component exchange depend on the protocols and specific requirements. It can be used in both simple plain text forms and various representation forms, such as JSON, XML, etc. GraphQL provides an API implementation approach where microservices can define the structure of the required data in their queries and thus prevent the transmission of redundant information. GraphQL also provides the ability to build multifunctional APIs that optimize the MSA infrastructure.

Fault tolerance

Fault tolerance in MSA is determined by parameters such as Latency. It means delays in data exchange and Exceptions (exception handling):

- Timeout - to to guarantee response time
- Retrytime - to automate retries in case of network degradation
- Circuit Breaker - to control in case of cascading errors
- Deadline- to control long requests that depend on many services
- Rate Limiter- to control load according to SLA (Service Layer) Agreement)

Data consistency

For data consistency MSA may be a serious problem because its database can have a structure distributed between microservices and can have multiple data sources. The latter involves the use of distributed transaction management mechanisms. Implementing, maintaining, and maintaining distributed transaction processing systems can be quite complex. Also, the use of existing solutions limits such projects to supported technologies.

Therefore, for microservices it is necessary to use transactions in the event-driven approach (Event-Driven Architecture) with two logical entities: messages and events [23-25].

Messages are part of systems based on message brokers. In such systems, messages are sent and queued. At a certain point in time, the consumer receives the message and executes the content without feedback from the source of the message on the principle of "fire and forget". The message format can be arbitrary (text, numbers, JSON, XML, etc.), which simplifies coordination between individual agents.

Events, in turn, have logically complete contextual information and a clearly defined structure of the resulting data. Event bus is used for their management (processing, storage, streaming) [23-25]. Events can be stored up to a specific time for event replay or audit.

Implementation of event-oriented MSA models

The SAGA pattern [25-27] can be used to ensure global data consistency using a sequence of local transactions. Its essence is based on the principle that the microservice publishes an event for each transaction, and the next transaction is initiated based on the results of the event. This can be done in two different ways, depending on the success or failure of the transaction. This approach involves maintaining data consistency between microservices, managing errors, establishing consistency, and coordinating transactions between them.

Choreography

It is assumed that each microservice performs a transaction and generates a message that similarly triggers transactions in subsequent microservices. In the event of an error in any link in such a chain, all previous transactions are canceled (Fig. 5, Fig. 6). Thus the consistency is maintained but the productivity of the system becomes low [27].

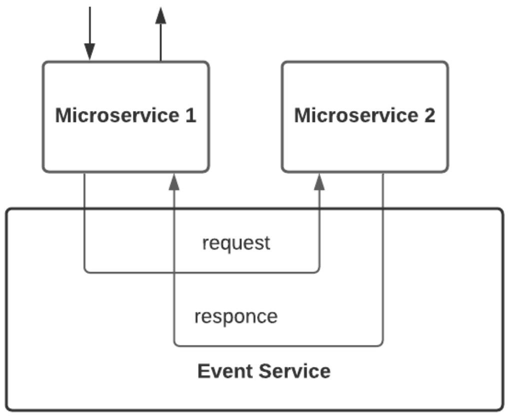

Figure 5 − Transaction scheme

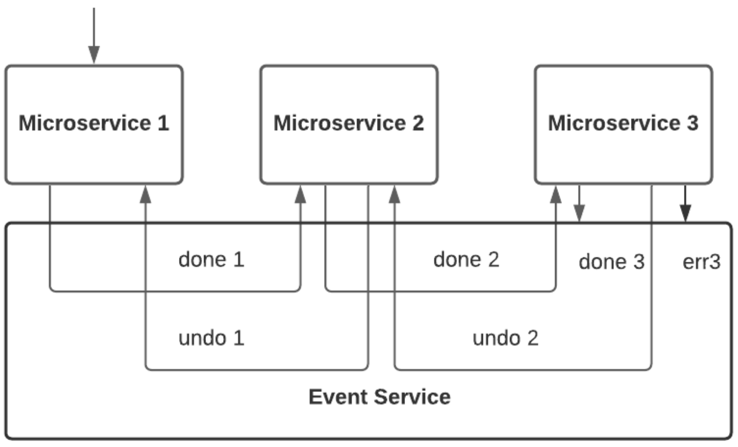

Figure 6 − Multiservice transaction scheme

Orchestration

In this case, an agent is used that sequentially initiates transactions with each microservice [27]. As in the case of choreography, the occurrence of an error with any microservice cancels all previous transactions. Orchestration can be used to search for events in their list in the event source. Each time the state of the system changes, a new one is added to the list of events, which is essentially atomic. Due to this, the orchestrator reproduces the current state of the essence, reproducing events (Fig. 7). In MSAs, which have a large number of events, to optimize the load, the system must periodically store the current state of the entities (Event Sourcing), which allows it to quickly restore their desired state.

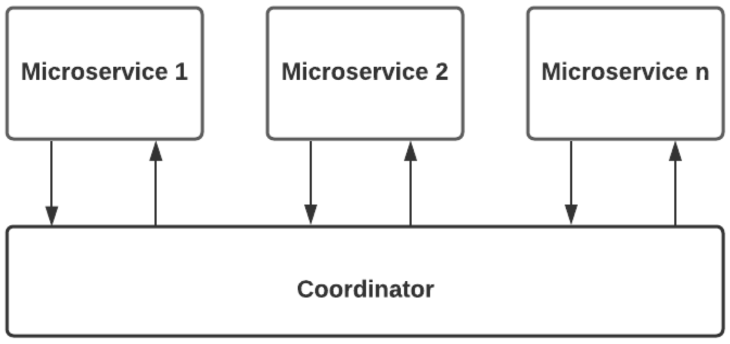

Figure 7 − Orchestration scheme

The advantage of the SAGA pattern is that the application is allowed to maintain data consistency between microservices without the use of distributed transactions, although the construction model becomes much more complex than for monolithic systems.

CQRS pattern

Command Query Responsibility Segregation (CQRS) allows MSA to take advantage of read and write pattern sharing. In the MSA, this may be the implementation of requests that receive data from multiple microservices. The purpose of using CQRS is also to maintain consistency of data and information about the state of the system. CQRS provides the ability to skip all claims checks in the current system, which improves its performance. CQRS simplifies the organization of the system by addressing its state and changing its state separately. The system must update the event replica of the domain to which the data belongs.

Advantages of CQRS:

- Ability to support many denormalized scalable datasets
- Separation of problems of obtaining status and implementation of teams
- The need to organize architecture for all sources of events

Disadvantages:

- Difficulty of implementation
- Delays replication

Patern ESP

For ELS, Event Streaming Processing (ESP) technology is effective for operational analysis of participants' activities, the results of current data, such as knowledge testing, etc.

ESP - applies an event streaming architecture to ensure that software components run in real time without communication and scaling. It includes the processing and visualization of events, their storage, processing of event streams or data in order to identify the necessary information. Search and processing use methods that are characteristic of streaming systems such as event correlation, event hierarchy, causality, analysis of component events and time series.

For ELS, ESP technology is effective for operational analysis of participants' activities, the results of current data, such as knowledge testing, etc.

ELS infrastructure

ELS are in the stage of permanent development in the course of their life cycle. Their complex MSA requires the use of a scalable and high-availability microservice solution. Maintenance and development of such systems with a large number of microservices is a very complicated and complex task that requires not only the implementation of the necessary functionality, speed and stability, but also scalability. From this point of view, ELS must have an appropriate optimized architecture (Fig. 8).

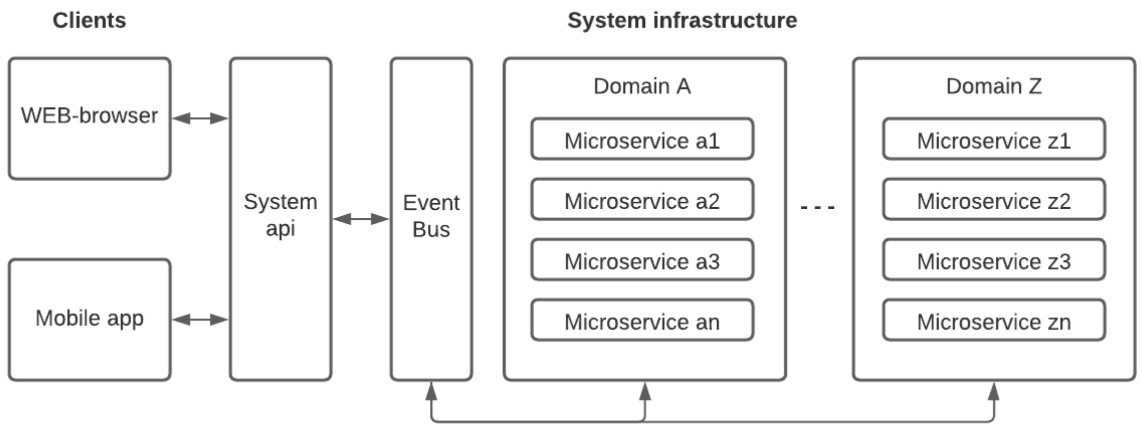

Figure 8 − Event-driven domain-based decoupling

One way to optimize is to use a domain-oriented architecture based on grouping microservices by functional domains (context). In such systems, domains interact through the event bus. Within each domain (context), services can interact with each other either through an API or through their own event bus. The subsystems built in this way are smaller and get better parameters of controllability and autonomy.

Organization of microservice architecture in ELS

The ELS microservice architecture depends on the organization of business processes, functionality, load balancing on system components, building databases, security requirements, reliability requirements and other parameters.

Given that ELS are multi-user systems, the requirements for their reliability are one of the most important tasks. From this point of view, the defragmentation of the monolithic ELS core into individual microservices that are able to perform their tasks autonomously is a rational solution. In addition, this approach can significantly increase the level of safety by isolating processes, as well as increase the load capacity through balancing and scaling.

When organizing the general architecture, separate groups of microservices should be combined into domains according to the functional contours of ELS, such as the student's domain, the teacher's domain, the dean's office domain, etc. Individual microservices can perform common functions between individual domains, such as the authentication module and the learning resources module between teachers and students.

In Fig.9 presents one of the options for building a student contour domain based on WEB-technologies. WEB-browsers and mobile applications are communication agents here. The HTTP protocol can be used for their communications in combination with data packing methods such as JSON, XML [28], etc.

The System API module provides authentication of client parts and the necessary interface for their interaction with the server part of the system, as well as data scheduling via Bus Event between domains and microservices of the system.

The Training materials domain provides teachers and students with access to resources of the streaming type (audio / video) and file type (text, hypertext, images, files…).

The Communications domain operates text and file exchange subsystems such as chats, forums, and social networks. The peculiarity of building its internal API is that domain resources can be represented by internal and external resources.

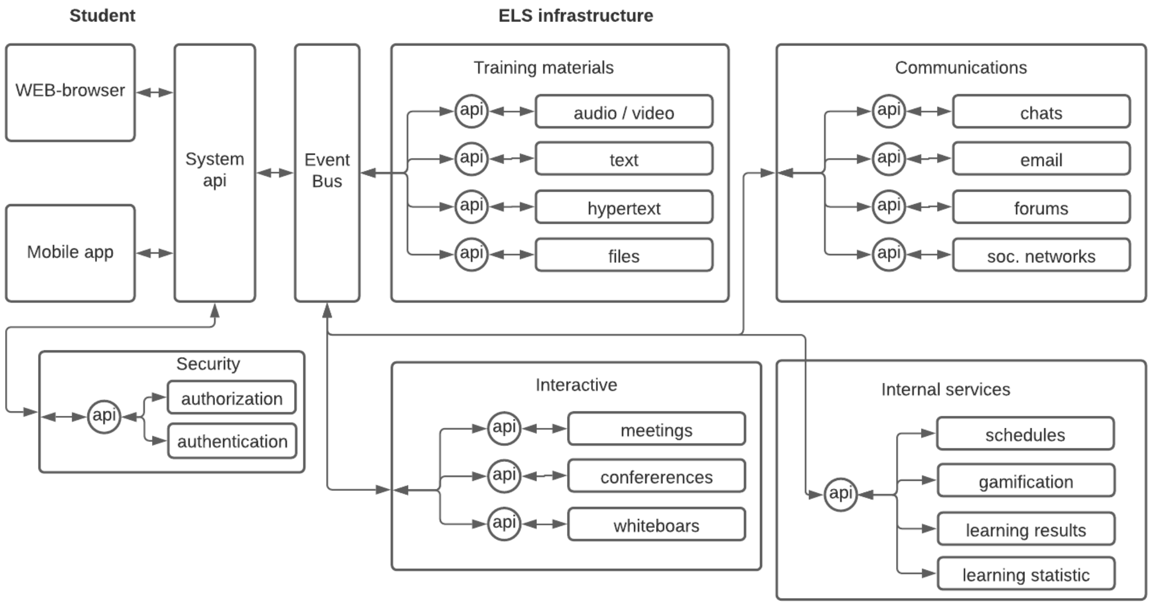

Figure 9 − MSA domain of ELS student`s contour

Therefore, in addition to data exchange support functions, such API can also provide system authentication in external services. The interactive communications domain supports streaming services such as audio and video chats, video conferencing, broadcasts, whiteboards, and more. The Internal services domain operates data processing products mainly from the entire ELS database. Figure 10 shows the MSA domain contour of the teaching part of ELS. It has components in common with the student part (WEB-interface, authentication and authorization subsystems), but differs in the presence of specific services for this task, namely:

- Personal repository of the teacher, which is designed to collect and use data on educational, scientific and other materials of the teacher.
- An institutional repository that presents official school files and has an API for interacting with teachers' personal repositories.
- Educational materials management subsystem.

- Student performance journals
- Document management subsystem (ELS), which is used to support the learning process of students, analysis of the effectiveness of their use by students.
- CV, teacher's portfolio, content for personal pages and teachers' sites.
- Archives of documents and educational materials

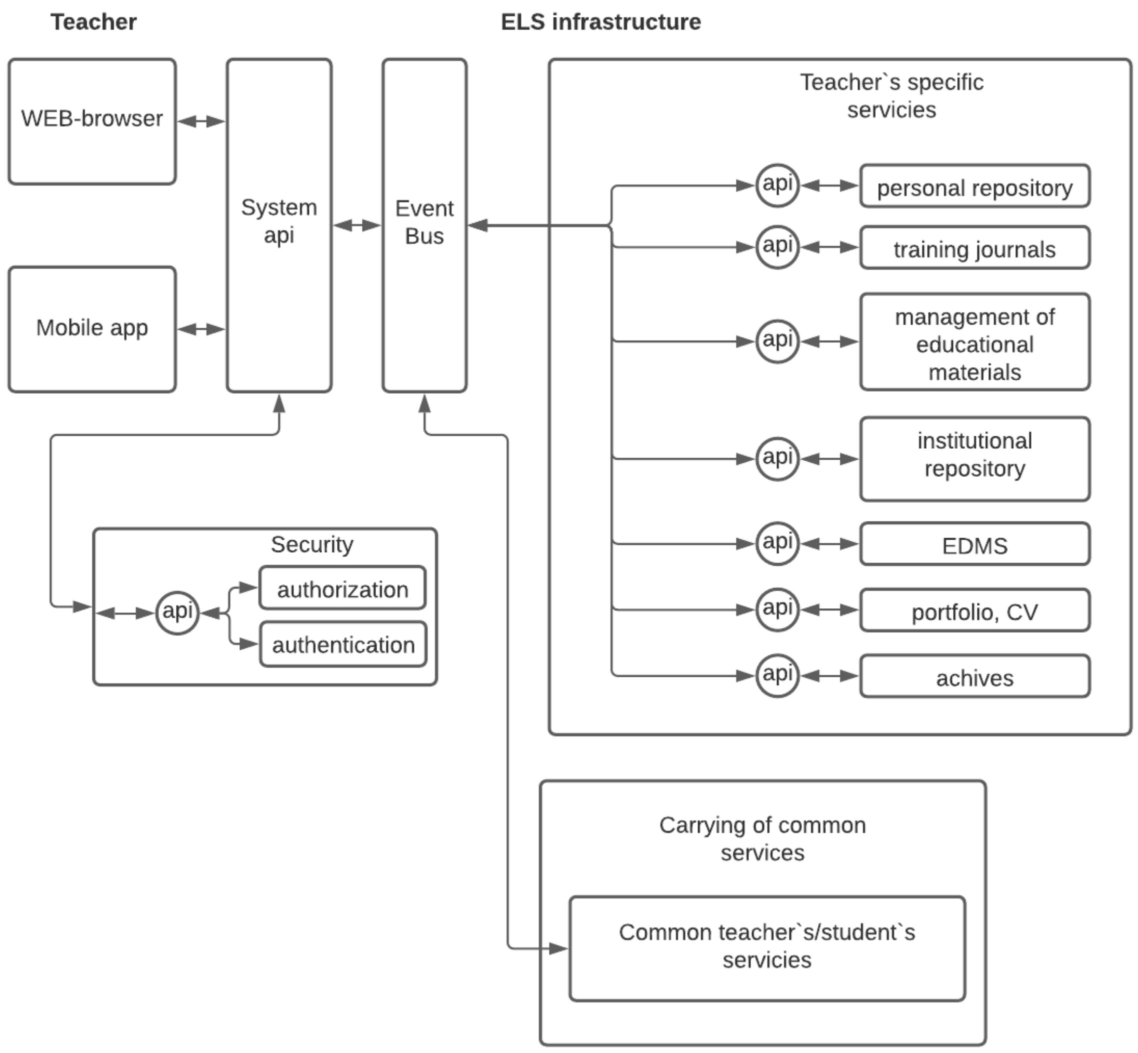

Figure 10 − MSA domain of ELS teacher`s contour

Figure 11 presents the MSA domain of the ELS deanery contour.

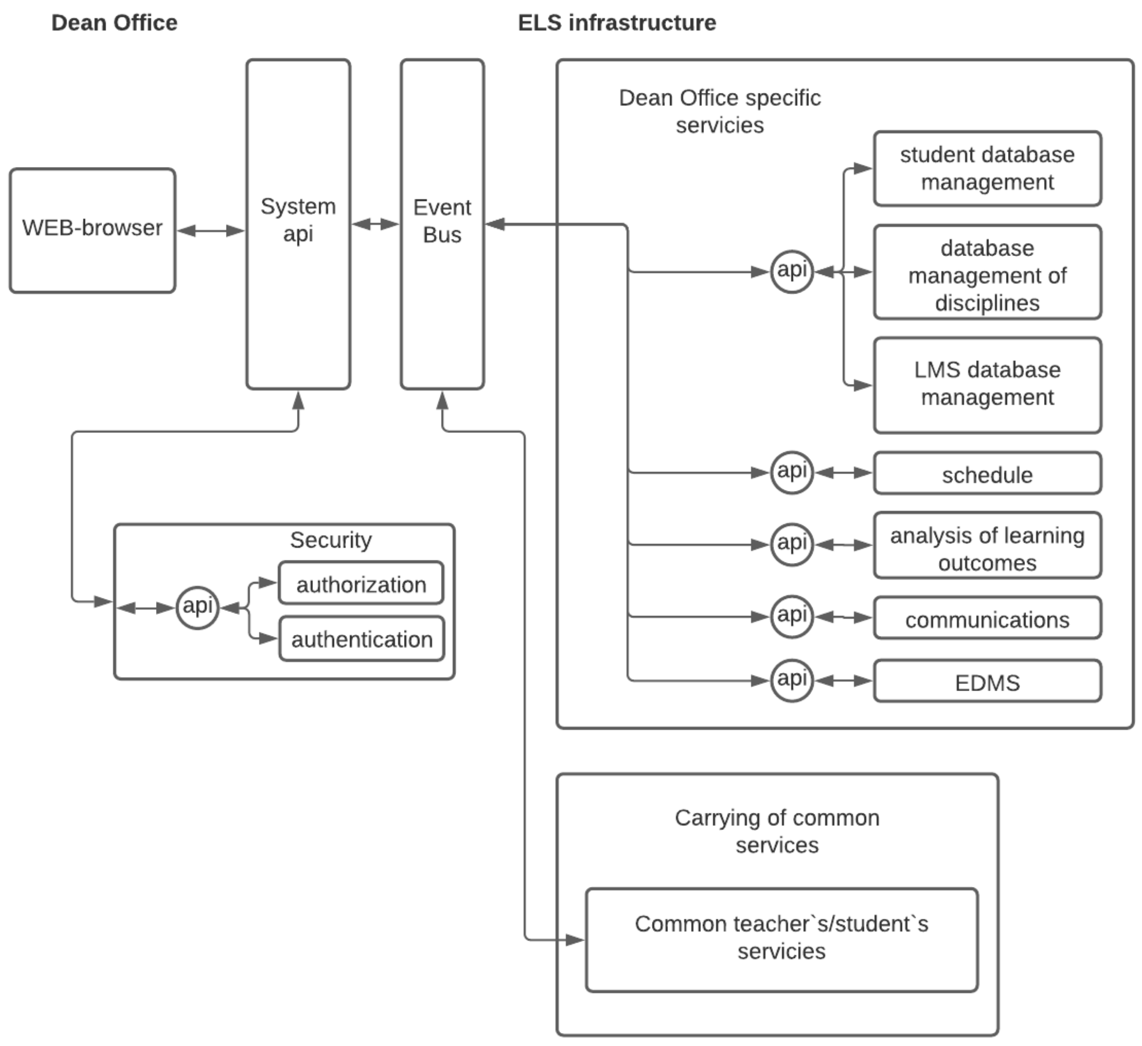
Dean Office
ELS infrastructure
WEB-browser
System api
Event Bus
Dean Office specific servicies
student database management
database management of disciplines
LMS database management
schedule
analysis of learning outcomes
communications
EDMS
api
Security
authorization
authentication
Carrying of common services
Common teacher`s/student`s services

Figure 11 − MSA domain of ELS deanery contour

Results

The proposed ELS architecture and principles of construction are implemented in the support system of the educational process JetIQ at Vinnytsia National Technical University [29-34]. The main part of JetIQ subsystems, clusters and microservices is located on virtual machines and containers on university servers. The project's microservices are implemented on LAMP-technologies using MySQL relational databases. Some microservices are located in the Google Cloud Platform cloud services.

Data exchange between microservices is based on RESTful Web API [19]. Data exchange formats - JSON [28], XML, HTML [20,22]. The orchestration of the system is based on LXC technology [35,36].

By 2021 the system has the following parameters (Table 1).

Table 1 – Results of MSA JetIQ implementation

Category	**Parameter**	**Value**
Users	Students	11,348
	Student groups	2,483
	Staff	1,708
Training and scientific resources	Course Materials	18,765
	Publications	32,194
	Navigators of educational resources	4,972

	Electronic tests	4,499
	Obtained answers to the question of electronic tests	4,463,799
E-dean	Lists of exams	40,809
	Scores	378,728
Electronic timetable	Lessons	49,830
Repositories (publications)	Science	31,682
	Documentary	2,498
	Personal docs.	71,441
Mobile applications (users)	JetIQ Student	3,706
	JetIQ Teacher	164

During operation from 2016 to 2021, a comparison of the monolithic prototype JetIQ and its microservice version was conducted. There has been a significant increase in the stability and reactivity of the MSA version due to load balancing and balancing between its subsystems. The microservice architecture has also increased the level of security by isolating microservices and using a multi-level authentication system.

Conclusions

The paper analyzes the aspects and technologies of using microservice architecture in e-learning systems. It is shown that MSA architecture allows to increase the reliability and stability of functioning due to relative autonomy of its separate microservice components. ELS microservice architecture has high scalability, including through the use of cloud technologies. The complexity of such systems is maintaining the state, synchronization and consistency of data.

References

[1] Goce Armenski, Marjan Gusev, "ARCHITECTURE of Modern e-learning Systems", *The 6th International Conference for Informatics and Information Technology*, pp. 38-42, 2008. [Online]. Available: http://ciit.finki.ukim.mk/data/papers/6CiiT/6CiiT-09.pdf.

[2] Blinco K., Mason J., McLean N., Wilson S. (2004), Trends and issues in e-learning infrastructure development, A White Paper for alt-i-lab 2004 Prepared on behalf of DEST (Australia) and JISC-CETIS (UK).

[3] Moodle. [Online]. Available: https://moodle.org.

[4] Elektronna systema upravlinnia VNZ "Sokrat". [Online]. Available: https://socrates.vsau.org.

[5] Elektronna systema upravlinnia VNZ "JetIQ". [Online]. Available: https://jetiq.vntu.edu.ua.

[6] OPENedX. [Online]. Available: https://open.edx.org.

[7] edX. [Online]. Available: https://edx.org.

[8] Canvas. [Online]. Available: https://www.instructure.com.

[9] Prometheus [Online]. Available: https://prometheus.org.ua.

[10] Liang, P.-H., Yang, J.-M.: Virtual Personalized Learning Environment (VPLE) on the Cloud. In: Gong, Z., Luo, X., Chen, J., Lei, J., Wang, F.L. (eds.) WISM 2011, Part II. LNCS, vol. 6988, pp. 403–411. Springer, Heidelberg (2011).

[11] Coursera. [Online]. Available: https://coursera.org.

[12] Udemy. [Online]. Available: https://udemy.com.

[13] Cisco Networking Academy. [Online]. Available: https://www.cisco.com/c/m/en_sg/sec-offerings/index.html.

[14] Adobe Captivate Prime LMS. [Online]. Available: https://www.adobe.com/ua/products/captivateprime/prime-rfi.html.

[15] Unix philosophy. [Online]. Available: https://en.wikipedia.org/wiki/Unix_philosophy.

[16] Patterns of Enterprise Application Architecture. With David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford. Addison-Wesley. ISBN 0-321-12742-0.

[17] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine. [Online]. Available: https://www.webcitation.org/67gOwyTek?url=http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[18] Erl, Thomas; Carlyle, Benjamin; Pautasso, Cesare; Balasubramanian, Raj (2012). "5.1". SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST. Upper Saddle River, New Jersey: Prentice Hall. ISBN 978-0-13-701251-0.

[19] Richardson, Leonard; Amundsen, Mike (2013), RESTful Web APIs, O'Reilly Media, ISBN 978-1-449-35806-8.

[20] Hypertext Transfer Protocol − HTTP/1.1. [Online]. Available: https://tools.ietf.org/html/rfc2616.

[21] Berners-Lee, Hendler, and Lassila, 2001; Markoff, 2006; Jensen, 2007.
[22] Fielding, Roy (June 2014). "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Section 4". IETF. Internet Engineering Task Force (IETF). RFC 7231. Retrieved 2018-02-14.
[23] Apache Kafka. [Online]. Available: https://kafka.apache.org.
[24] RabbitMQ. [Online]. Available: https://www.rabbitmq.com.
[25] Redux-Saga. [Online]. Available: https://redux-saga.js.org.
[26] Pattern SAGA. [Online]. Available: https://microservices.io/patterns/data/saga.html.
[27] SAGA Pattern. [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/saga-pattern.html.
[28] Java XML and JSON: Document Processing for Java SE. Jeff Friesen. Apress; 2nd ed. edition (January 11, 2019), ISBN 1484243293, 546 pages.
[29] Y. Palamarchuk., O. Kovalenko, "Optimization of electronic test parameters in learning management systems", CEUR Workshop Proceedings, v. 2762, pp. 98 – 109. 2020 2nd International Workshop on Information-Communication Technologies and Embedded Systems, ICTES 2020, Virtual, Mykolaiv, 12 November 2020, null, 165503.
[30] O. V. Bisikalo, Y. A. Palamarchuk, O. O. Kovalenko, "Results of implementation of the pilot project of management system for learning and concomitance of the educational, methodological and scientific activities "JetIQ", *Materialy 9-yi naukovo-praktychnoi konferentsii, m. Lviv, 21-23 lystopada 2017 r.* – Lviv : Vydavnytstvo Naukovoho tovarystva im. Shevchenka, 2017, s. 73-77.
[31] Oleg Bisikalo, Olena Kovalenko, Yevgen Palamarchuk, "Models of Behavior of Agents in the Learning Management System", *Materialy XIV-oi Mizhnarodnoi naukovo-tekhnichnoi konferentsii "Komp`iuterni nauky ta informatsiini tekhnolohii (CSIT -2019)"*. Lviv, Ukraine, 2019, tom 3, s. 222-227.
[32] Kovalenko Olena, Palamarchuk Yevhen, "Kontury systemy upravlinnia navchanniam: tradytsiine, zmishane ta dystantsiine navchannia", *«INTERNET-OSVITA-NAUKA-2020», XII Mizhnarodna naukovo-praktychna konferentsiia ION-2020, 26-29 travnia*. Vinnytsia, Ukraine: VNTU 2020, s. 230-231.
[33] Yevhen Palamarchuk, Olena Kovalenko, "Algorithms of blended learning in IT education", *XIII International Scientific and Technical Conference on Computer Sciences and Informational Tecnologies (CSIT), 11-14 September*. Lviv, Ukraine, 2018, pp. 382-386.
[34] Senthil Kumaran S., *Practical LXC and LXD: Linux Containers for Virtualization and Orchestration*. Apress, 2017, 159 c.
[35] Konstantin Ivanov, *Containerization with LXC*. Packt Publishing, 2017, 352 c.

Стаття надійшла: 25.12.2021.

Відомості про автора

Паламарчук Євген Анатолійович – кандидат технічних наук, доцент кафедри автоматики та інтелектуальних інформаційних технологій.

Є. А. Паламарчук

МЕТОДИ ПОБУДОВИ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ СИСТЕМ ЕЛЕКТРОННОГО НАВЧАННЯ

Вінницький національний технічний університет, Вінниця