The processes of designing and supporting e-Banking systems built using a microservice architecture are considered in this paper, taking into account the need to ensure an appropriate level of software quality attributes. The study resolves the task of integrating knowledge-oriented models and intelligent methods into model-technological tools required for developing a microservice architecture for systems of this class.

The key approaches and principles of microservice architecture design, as well as the limitations of their application in the e-Banking domain, have been investigated. The architecture has been designed; a prototype of an e-Banking system fragment was developed using the defined model-driven technological framework, followed by experimental evaluation of its quality metrics. The results demonstrate that the combination of microservice architecture, architectural patterns, and intelligent methods eliminates the shortcomings inherent to monolithic e-Banking systems and provides the required software quality levels.

The key features of the proposed approach include the integration of knowledge-oriented models and intelligent approaches into the designed architecture, as well as the adaptation of microservice architecture design patterns to the requirements and operational specificity of e-Banking systems. That made it possible to improve the performance and reliability indicators compared to monolithic architectures, which remain widely used in the development of such systems.

In practice, the proposed microservice architecture and the researched implementation tools could be applied while designing new e-Banking systems or when upgrading existing ones

Keywords: internet-banking, domain modeling, microservice architecture, software, metrics, quality

UDC 004.051

DOI: 10.15587/1729-4061.2026.351604

DEVELOPMENT OF INTELLIGENT MODEL-TECHNOLOGICAL TOOLS FOR E-BANKING SYSTEMS BASED ON MICROSERVICES

Tymur Daas
Corresponding author
PhD Student*
E-mail: timur.daas@karazin.ua
ORCID: https://orcid.org/0000-0003-2796-8145
Mykola Tkachuk
Doctor of Technical Sciences*
ORCID: https://orcid.org/0000-0003-0852-1081
*Department of Intelligent Software Systems and Technologies
V. N. Karazin Kharkiv National University
Svobody sq., 4, Kharkiv, Ukraine, 61022

Received 10.11.2025
Received in revised form 16.01.2026
Accepted date 26.01.2026
Published date 27.02.2026

How to Cite: *Daas, T., Tkachuk, M. (2026). Development of intelligent model-technological tools for e-Banking systems based on microservices. Eastern-European Journal of Enterprise Technologies, 1 (2 (139)), 48–57. https://doi.org/10.15587/1729-4061.2026.351604*

1. Introduction

E-Banking systems occupy a leading position in the modern global financial infrastructure, as they provide customers with access to a wide range of banking services in real time. More and more banking services are moving to automated and remote interaction channels. The constant growth in the number of users, their payment transactions and functional capabilities leads to a complication of the architecture of these systems. E-Banking systems must provide customers with the opportunity to receive services 24/7, the prompt implementation of new functions and guaranteed availability of services. Even a short-term suspension of such services can lead to significant economic losses, a decrease in the level of customer trust, and the loss of the bank's competitive position [1].

Most existing e-Banking systems were designed during a period when the three-tier monolithic architecture (MA) was dominant, focused on centralized data processing logic and resource sharing. Conventional monolithic solutions are increasingly unable to meet such non-functional requirements as scalability, performance, and reliability that are imposed on systems of this class. Such solutions usually lack a clear model approach to describing the domain, which complicates their further development and maintenance. Monolithic systems have a number of significant limitations, which are manifested in reduced performance with increasing loads, poor scalability, the complexity of making changes to business logic, and limited integration capabilities with new services. In addition, knowledge-based models and intelligent methods capable of formalizing complex business logic and supporting decision-making at the design and maintenance stages have not traditionally been used in the process of designing and developing such systems. The lack of application of such approaches leads to excessive dependence on the expert experience of stakeholders, increases the costs of designing and maintaining the architecture, and complicates the adaptation of the system to new business requirements. Under conditions of increased requirements for performance, security, and continuity of service, the architectural limitations of monolithic systems are becoming increasingly critical. The need to ensure low latency in server responses when performing financial transactions, high availability of services, and the ability to update them independently are tasks that are quite difficult to implement within a single software package. This creates a need to find new approaches to architectural design [2].

One of the most qualitative modern reference system architectures (RSA) is microservice architecture (MSA). MSA provides the division of the system into independent components, each of which has its own life cycle and can develop, scale, and deploy independently [3]. Using MSA makes it possible to eliminate key limitations that are present in conventional monolithic systems. However, the process of transition to MSA for banking systems is complex and involves the need to ensure transaction coordination, fault tolerance, high throughput, and

continuous availability of services. These challenges necessitate the search for a new architectural approach that would combine the advantages of MSA with the capabilities of knowledge-oriented models and intelligent methods. Such an approach should provide not only the decomposition of the system into logically consistent microservices but also improve the processes of designing and maintaining the system architecture. Such processes include business requirements analysis, load forecasting, adaptive configuration of components, and system adaptation to new operating conditions. Also, existing practices for building microservices are mostly general in nature and do not take into account the critical features of banking systems.

In addition, as noted above, banking systems cannot afford to be out of service even for a relatively short time. Therefore, one of the key tasks for all stakeholders (architects, analysts, developers, customer) is to ensure a seamless transition from a monolithic architecture to a microservice architecture without interrupting the system. This task is non-trivial and relevant, especially given the complex logic of banking processes and high requirements for the quality of customer service. Thus, there are currently no comprehensive architectural approaches that would simultaneously combine the advantages of MSA and knowledge-based methods, taking into account the specificity of banking systems.

An urgent scientific problem that needs to be solved is to devise an approach to designing MSA for e-Banking systems that integrates knowledge-based models, intelligent methods, and modern architectural practices. This will ultimately ensure an increase in the quality and adaptability of the architecture of such systems as a whole.

2. Literature review and problem statement

One of the topics of scientific papers is the study of using architectural patterns to increase the flexibility and adaptability of architecture. In [4], two issues are considered in the context of the FinTech domain: the use of design patterns and MSA. The authors emphasize the importance of the "4 + 1 projection" model as a means of formalizing the description of the architecture from the point of view of its various components: functional, process, component, and physical (deployment). However, despite the universality and potential of the defined model, the study is mainly methodological in nature. It does not take into account the specific requirements of highly loaded banking systems: transactional consistency, resistance to peak loads, security requirements and complex integration mechanisms. The possibility of integrating knowledge-oriented models or intelligent methods also remains unexplored, which creates a gap between the formal description of the architecture and its practical application in e-Banking class systems. The issue of architectural patterns is also raised in study [5]. The authors note that most e-Banking systems are still built on the basis of MA and propose a pattern-oriented framework for building an adaptive architecture of Internet banking systems based on MSA. The authors prove that the consistent application of architectural patterns makes it possible to improve the reliability, performance, and scalability indicators and ensure the possibility of adaptive reconfiguration of the system depending on changes in the load. It is important to note that in the conclusions of their work, the researchers directly point to the prospects for using neural networks and analyzing user behavior patterns. They point to the need to integrate knowledge-oriented and intelligent methods in future stages of framework development, which, in their opinion, should reduce the load on the server during critical operations. However, the work does not identify specific mechanisms for formalizing knowledge, methods for making and supporting architectural decisions, or applying intelligent approaches, which emphasizes the prospects for further scientific research in this direction. This drawback can be corrected by conducting relevant studies.

In work [1], the use of MSA implemented using the popular Java frameworks Spring Boot and Spring Cloud for the task of scaling banking applications is considered. The author notes that these frameworks offer tools for high-quality provision of modularity, separate scalability capabilities, and ease of testing services. In his work, the author also focuses on the use of various architectural patterns for distributed applications. It is indicated that they ultimately help achieve the appropriate quality attributes of the software (SW) in banking applications. In addition, the work considers various auxiliary functional capabilities of the frameworks that do not directly relate to the MSA but simplify the implementation of specific functional requirements that are imposed on e-Banking class systems. However, the work focuses only on the technological component and does not consider the issue of using knowledge-oriented and intelligent methods in the design of the architecture or in the business processes of the system.

The practical features and challenges of building MSA are considered in study [6]. It demonstrates the effectiveness of using typical patterns that best meet the requirements and features of MSA using the example of a FinTech loan processing platform. The work proves that using these patterns in the task of transferring a system from MA to MSA provides an increase in such quality attributes as productivity, scalability, and reliability. Separately, the authors note that they managed to significantly reduce the server response time for the most critical operations and achieve an 85% reduction in system deployment time. However, the work focuses exclusively on engineering aspects. It does not cover issues of domain modeling, semantic consistency, or intelligent methods for supporting architectural solutions. The authors clearly indicate this in their conclusions. They especially note the prospect of using AI approaches to verify the results of the decomposition of the system into separate services and using the Saga pattern for their interaction with each other. Thus, the work demonstrates the advantages of MSA but does not solve the problems of adaptability and intelligent improvement of the architecture, which the authors plan to investigate in separate studies.

Also important in the context of studying e-Banking class systems is work [7], which focuses on the requirements for the design and user experience (UX) of electronic bank account statements. The authors conduct a systematic review of UX principles and formulate recommendations for the application of these solutions for statements in e-Banking systems, emphasizing the key factors of their usefulness and convenience. The work does not concern architectural aspects or the construction of the MSA in particular. It defines the main functional requirements that should be taken into account when designing the relevant services in e-Banking systems. First of all, statement generation services, which are central to our study.

A separate area of research concerns modern approaches to monitoring and ensuring observability of distributed systems. In work [8], a combination of Prometheus, Grafana, Loki and Alerta tools is analyzed to achieve increased transparency and rapid response to anomalies in Kubernetes clusters. It was shown that the use of these tools has a positive effect on such

software quality attributes as reliability, availability and performance. Despite the significant practical contribution to the field of DevOps, the work does not consider modern intelligent decision-making mechanisms but is focused exclusively on the technical features of the use of modern monitoring tools. Its results can be used to increase the reliability, availability, and performance of systems built on the basis of MSA. However, they do not solve the problems of intelligent architecture design and the features of their application directly in e-Banking class systems, which can be done in further research.

The directions of using artificial intelligence (AI) in the processes of design and maintenance of information systems (ISs) built on the basis of MSA are also studied. In paper [9], an analysis of modern AI methods was performed and their relationship between their use and improving the quality attributes of MSA was determined. The authors focus only on one stage of the software life cycle – DevOps, which is the main drawback of that study. The authors note this issue as one that they plan to investigate further.

Modern research on software development processes for the financial sector is mostly focused on the issue of transition from monolithic architectures to microservice solutions. MSA makes it possible to improve the indicators of scalability, productivity, reliability, and maintainability, as well as ensure continuity of service. However, despite significant progress in methods of system decomposition, the use of architectural patterns and the emergence of various technological solutions, the issue of integrating knowledge-based models and intelligent methods into the processes of design and maintenance of MSA remains insufficiently studied. This also applies to e-Banking class systems.

Our review of the literature [1, 4–9] reveals that there is a systemic gap in current studies. Despite significant progress in the use of MSA, architectural patterns, and various technological solutions, there are practically no approaches that combine MSA with knowledge-based models and intelligent methods, which would be able to support the adoption of architectural decisions, automate the configuration of services, and ensure adaptive development of the system in accordance with changes in business requirements. The issue of formalizing domain knowledge in the context of e-Banking and using such models to define the boundaries of microservices, select architectural patterns, or optimize the interaction of system components also remains unexplored in [1, 4–9].

3. The aim and objectives of the study

The purpose of our work is to develop and evaluate the effectiveness of the intelligent modeling and technological tools necessary for building an MSA for a typical e-Banking system, which integrates an intelligent forecasting module (IFM) [10]. This will give the opportunity to provide better quality attributes of software systems of this class, such as scalability, performance, and reliability.

To achieve the goal, the following tasks were set:

– to identify the limitations of existing approaches to building an MSA when applied in the e-Banking domain;

– to systematize the conceptual principles of designing an MSA, identify architectural patterns suitable for integration into e-Banking systems, and justify their adaptation taking into account knowledge-oriented approaches and AI-driven design;

– to design the architecture and develop prototype of a fragment of the e-Banking system, in particular IFM [10], which is built on the basis of an MSA and is a vivid example of the proposed intelligent modeling and technological tools for a typical e-Banking system;

– to conduct experimental studies to assess the quality of the designed prototype and the approach proposed in paper [10] to building user account statements in general according to some indicators defined in [10], also comparing them with the corresponding indicators of implementation in a monolithic architecture.

4. Materials and methods

The object of our study is the design and maintenance processes of e-Banking systems built using MSA, taking into account the need to ensure an appropriate level of software quality indicators.

The principal hypothesis assumes that the combination of MSA architectural principles and knowledge-based approaches [10], including intelligent forecasting methods, makes it possible to improve the quality of architectural solutions. At the same time, the delay in performing critical operations should be reduced and the system should be adaptable to changes in loads and business requirements.

The study puts forward several assumptions aimed at determining the real operating conditions of systems of this class. First, e-Banking systems operate in a highly loaded environment. Second, users often make requests for the generation of account statements, but these requests are uniform in time, especially when it comes to large-volume statements. Thirdly, the use of modern design patterns allows for the qualitative implementation of the architectural principles of MSA, such as domain modeling [10], which allows for a reasoned definition of the boundaries of future services.

In general, the design of MSA for e-Banking systems is based on a certain set of methods and approaches. They allow for the decomposition of the domain, design high-quality interaction between the selected components, and provide the possibility of integrating intelligent mechanisms for further support of the proposed architectural solutions. The basic principle is the decomposition of the system into services with clearly defined boundaries of responsibility (bounded context). The system design stage is one of the first according to the typical software life cycle. The use of such an approach at the design stage allows for minimizing dependences between services, increasing their autonomy of operation, and simplifying future system scaling. For the e-Banking domain, this allows for the separation of functionality related to transaction processing, account management, customer profile maintenance, statement generation, and other critically important processes. That, in turn, ensures one of the important modern requirements for systems of this class: the absence of impact of new modules on the quality and logic of the already implemented functionality.

The leading role in the design of MSA is played by domain modeling (Domain-Driven Design – DDD), which is used to define the boundaries of services and formalize the business logic of the domain. In the context of e-Banking systems, this means building models that reflect the key information entities of the domain: users, accounts, transactions, statements, payment instruments; as well as the principles and order of their interaction in the system. DDD helps ensure that the proposed architectural solutions correspond to the given domain and reduce the risk of duplication of functionality between different services [11]. The second important aspect

is the choice of the method and protocols for communication between services. In banking systems, it is impossible to limit ourselves to synchronous calls via REST (Representational State Transfer) or gRPC (Google Remote Procedure Calls) because they create additional delays at high loads and also require more resources. The use of event-oriented approaches using message brokers should unload the system, namely, transmit events asynchronously and provide better resistance to peak loads. This approach is advisable to use for background tasks, such as, for example, generating statements or processing transaction statistics.

It is worth noting the architectural patterns of MSA, which provide reusable solutions for typical integration and service management tasks. Below are modern architectural patterns that are advisable to use to solve the tasks of our study:

– API Gateway provides centralized routing of all requests to and within the system, which also allows for an easy and high-quality implementation of various access controls to ensure the appropriate level of security [1, 12];

– Database per Service provides for the allocation of a separate data store for each microservice, which ensures data storage autonomy [1];

– Saga Pattern is responsible for managing distributed transactions in MSA and helps solve the issue of rolling back transactions in the event of various failures, which is one of the main difficulties of MSA in comparison with other reference system architectures [13];

– Circuit Breaker helps solve the issue of early notification of the unavailability and/or inoperability of other system services, which helps increase system reliability and implement failure isolation mechanisms [1];

– Event Sourcing implements mechanisms for preserving the history of changes and restoring the state of the system, which is especially important for any financial systems, including e-Banking class systems [1].

The use of these patterns allows for the formation of more flexible, stable, and secure architectures but, due to the strict requirements for transactional integrity and data protection in the e-Banking domain, they require additional adaptation and improvement.

In addition to the above methods, various intelligent approaches are increasingly being used for the tasks of architecture design and support, as well as deployment and maintenance processes [9]. Work [14] presents a classification of such methods and the main challenges of their application. Intelligent approaches are able to take into account the accumulated knowledge about the system operation, identify hidden patterns in user behavior and predict load changes, which is of practical importance for the evolution of MSA. AI-architecture design methods cover the stages of requirements analysis and creation of a conceptual system structure. Their goal is to automate the initial stages of development: from the analysis of various text specifications to determining the boundaries of future services. AI-deployment and maintenance methods are designed to adapt the operation of systems already deployed in a productive environment and ensure their stable functioning under conditions of possible changes in the productive environment. AI-support of system development involves the use of analytical methods in the process of system maintenance for continuous improvement of the architecture and increasing its stability. One of the promising and poorly researched areas of development of intelligent approaches in the construction of MSA is knowledge-based design. It involves the use of formalized knowledge models in the processes of decomposition, configuration and maintenance of systems. Unlike purely algorithmic solutions, this approach is based on the accumulation, structuring, and reuse of knowledge obtained in previous projects or during the operation of the system.

Our work focuses on the application of a knowledge-based approach in combination with MSA in the domain of account statements, which are actively formed by users of e-Banking systems, as well as on the development and research of model and technological tools for its implementation in systems of this class. In [10], the use of a new IFM module for predicting user actions in the system when building account statements was proposed and functional requirements for it were determined. The main intelligent method used in IFM is the SARIMA time series analysis method. It should provide prediction of the time when a user will access the system to build the next statement based on accumulated data about the previous actions of this user in the system and his ontological profile.

An integral part of design is the verification of built prototypes. For experimental research and assessment of their quality, there are various methods of software testing. Among them, the following main ones can be distinguished:

– verification of the prototype's operability using unit and module testing;

– load testing: what number of resources is involved under different load profiles, what is the system response time, what is the percentage of errors, what is the percentage of critical load of the central processor (CPU);

– fault tolerance testing: average recovery time after failures, average time between failures.

It is worth noting that the above testing methods are aimed at assessing the architectural approaches used and are not aimed at assessing the business logic of the system.

5. Results of research into the processes of design and maintenance of e-Banking systems

5. 1. Determining limitations in the application of existing approaches to building a microservice architecture in the e-Banking domain

Typical principles of MSA design involve a clear separation of domains, separate data storage, minimal areas of responsibility of services and independent life cycles of components. The analysis of existing approaches to IS decomposition revealed that typical principles of MSA design perform well in systems that do not have high requirements for speed and consistency of transactions. Decomposition into too small services leads to the fact that the implementation of a significant part of key business operations requires significant inter-service interaction in this case. This applies to such operations as the formation of various types of payments, checking the correctness of details, processing transactions or generating a statement. Such calls require additional overhead, both in terms of resources and time, although in essence they perform minimal business logic. Examples include the following operations that are necessary for the formation and execution of a payment: searching for account balance, obtaining counterparty details, obtaining user settings. Formally, they should be implemented in separate microservices since these same functions are also required for many other functional parts of the e-Banking system. At the same time, such an implementation will negatively affect performance indicators. This is especially true for operations such as mass import of payments, where it is mostly impossible to make a

single call, and accordingly, overhead costs grow in arithmetic progression. In contrast, examples of similar operations can be given, but the complexity and support of which are too high not to be carried out in separate microservices. This may be a microservice that performs cryptographic operations, works with OTP codes (One-Time Password) or authorization.

Based on the above, it is possible to formally, in the form of a formula, determine the basic principle of system decomposition into services, which can be applied when designing e-Banking systems

$$DD = f(TRC, NC, OLC), \quad (1)$$

where *DD* is the depth of decomposition, *TRC* is the complexity of transaction coordination (from the point of view of databases) performed in the domain, *NC* is the number of network inter-service requests, *OLC* is the complexity of supporting the business logic of "joint operations".

As a result, we can say that the specificity of the e-Banking domain lead to the need to adapt approaches to decomposition in MSA and necessarily take into account the negative impact on system quality indicators. If the negative impact on the speed of execution of certain operations is significantly greater than the benefit that can be obtained from reducing code duplication, then the above operations should be implemented several times. That is, within each service where they are needed.

5. 2. Systematization of conceptual principles for def signing microservice architecture of e-Banking systems taking into account knowledge-oriented approaches

To determine and systematize the conceptual principles of designing MSA for e-Banking class systems, the product architecture from two Ukrainian IT companies specializing in software development was analyzed. The first company is TOV "CIES" (CS) [15], which is a product IT company, and, accordingly, the architecture of its software applications, in addition to the basic requirements of e-Banking class systems, also takes into account the requirements for variability. The second is an IT company that develops software applications for a specific Ukrainian bank. The architecture analysis was carried out by studying the architectural documentation of the developed software applications, which are built on the basis of MSA, as well as by interviewing software architects and technical leaders at departments/development teams of these companies.

Below is a list of defined conceptual principles and features of their adaptation to e-Banking class systems:

1. DDD architectural style. The principle of separating a complex system into contexts is clearly combined with the concept of MSA and gives better results when involving a large number of stakeholders who can provide high-quality information about the system from different points of view. Using DDD in MSA, it is quite easy to implement functionality that will implement knowledge-oriented methods that can be transferred to a separate microservice that will provide API to all other services of the system. But it is mandatory to take into account the limitations of the use of DDD, which were listed earlier.

2. Communication protocol between services into which the system is divided using DDD. When used in e-Banking systems, both synchronous and asynchronous calls should be combined. Synchronous calls should be applied to operations that must immediately return the result of execution, for example, authorization in the system. Asynchronous calls should be used if there is no urgent need to immediately return the result, or obtaining the result can be implemented by adapting functional requirements, for example, push notifications or generating large statements, as described in [10]. For asynchronous messaging, the use of message brokers is mandatory [16].

3. Using architectural patterns that will ensure compliance with the requirements of MSA and the specified software quality attributes for e-Banking systems. The API Gateway pattern should be used to implement access control mechanisms and authentication tools for any inter-service interaction in the system. The Database per Service pattern should be used in a limited way due to the high degree of interdependence of stored data in e-Banking systems, which can be associated with several different information entities. And high requirements for data consistency make it impossible to duplicate them in different databases (DBs). Saga Pattern was defined as one of the mandatory mechanisms for building MSA in e-Banking systems. Systems of this class have very high requirements for transaction management, and they are especially complex in terms of handling failures with subsequent rollback of the entire chain of operations. Many operations require either "rollback of the entire transaction" if possible or designing the system in such a way that there are compensation mechanisms in case of impossibility of full physical rollback of the transaction to the database. The implementation of the Circuit Breaker pattern should be performed using the tools of the Kubernates system, which has built-in mechanisms for monitoring the health of each running microservice. The Event Sourcing pattern should be implemented by implementing event logs, and from the point of view of architecture and software tools – by tools used for tracing, such as Zipkin and OpenTelemerty.

4. Implementation of intelligent approaches in the prop cesses of design and support of e-Banking systems based on MSA. For better processing of a significant amount of domain knowledge, AI-architecture design tools should be applied, namely the NLP (Natural Language Processing) method [14, 17]. It should automate the process of identifying information entities and their relationships, which will be useful for domain modeling in the future. There are also various tools for automated construction of the main types of diagrams [18]. Tools such as ChatUML or PlantUML allow one to quite conveniently build different types of UML (Unified Modeling Language) diagrams based on text entered by the user. Another direction of implementation of intelligent approaches is AI-deployment and support of MSA components, which are aimed at adapting the work of already deployed systems and ensuring their stable functioning. One of the methods of this class is the case-based reasoning (CBR) method, the feasibility of using which for adaptive configuration management of software microservices was proven in [19].

5. 3. Design of architecture and a prototype for a fragment of the e-Banking system based on microservice architecture

During the research, the architecture was designed and a prototype of a fragment of the e-Banking system was programmatically implemented, which is limited to the domain of "formation of account statements". Functional requirements for this domain were defined in [10], which proposed an improved algorithm for constructing user account statements. It is assumed to use the IFM module, which implements the application of knowledge-oriented methods, to ensure the proper level of quality attributes of e-Banking systems by improving two quality attributes: performance and reliability. The main components involved in the work of IFM are shown in Fig. 1 in the form of a package diagram [20].

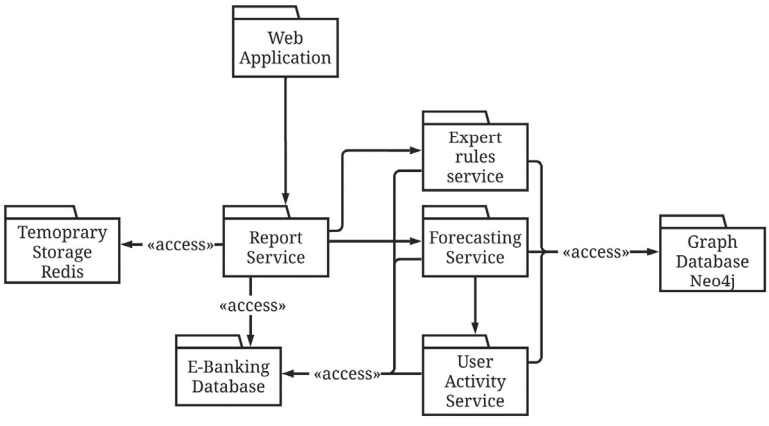

Fig. 1. Package diagram of component services in the intelligent forecasting module

Fig. 2 shows the deployment diagram [20], which depicts the physical architecture of the specified fragment of the e-Banking system. The diagram shows the division of the system into microservices. Both microservices directly for the domain "formation of account statements" and some service microservices that are somehow involved in the processes of the domain are presented. The diagram also contains microservices that are intended for the implementation of the architectural patterns defined above. For each microservice, a communication protocol is defined based on the previously defined principles of inter-service interaction. All microservices are deployed in Docker containers under the control of the Kubernates orchestration system, which provides load balancing and configuration management of microservices. For the tasks of collecting metrics and monitoring, the Prometheus and Grafana systems are used, which allow real-time receipt and visualization of information about the state of the service, as well as reporting on anomalies [8].

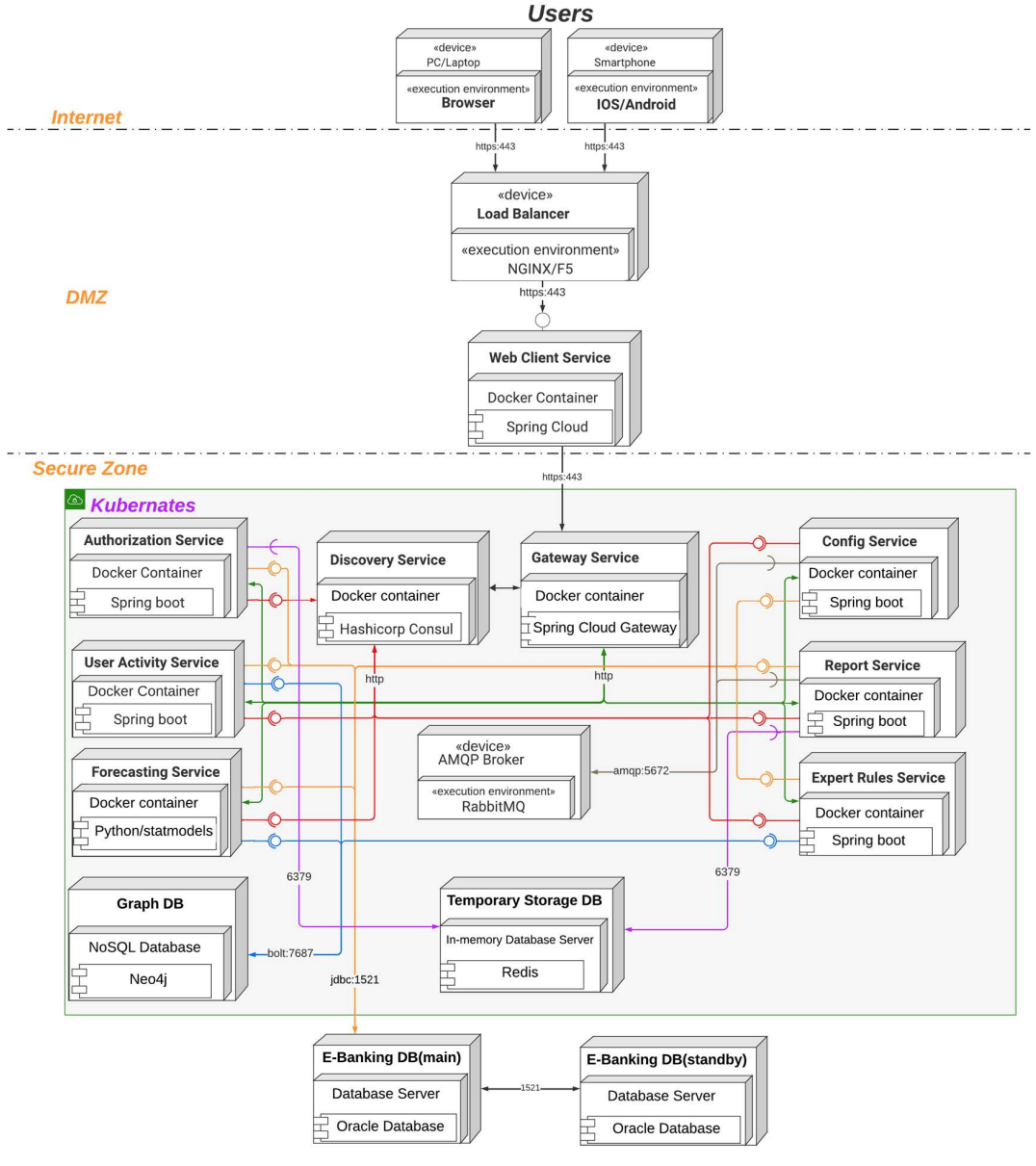

Fig. 2. Diagram of the deployment of an e-Banking system based on microservice architecture

The issue of the selected technology stack that was used in the development of the prototype deserves special attention. All microservices, except Forecasting Service, were written in the Java programming language, which is the leading development tool for high-load enterprise solutions. The Python language and the Statmodels library were used to write Forecasting Service [10, 21]. The Spring technology stack was chosen as the basic framework, as well as its Spring Cloud extension. They have all the tools for implementing the basic principles of MSA and key architectural patterns in the developed software [22]. For example, Spring Cloud Gateway fully implements the API Gateway pattern, which was defined as mandatory for use. To ensure asynchronous interaction between microservices, the RabbitMQ message broker and the Redis in-memory database were used, with Redis to temporarily store additional information of a large (as for RabbitMQ) volume to such an asynchronous message.

5. 4. Results of the experimental study on the quality assessment of the proposed approach

The purpose of the experiment was to assess the impact of the proposed architecture on key software quality attributes, such as performance, scalability, and reliability, and to compare them with similar indicators for the same functionality in a monolithic architecture. Testing was performed within the functionality implemented in the proposed prototype.

The experiments were conducted using the test environment of the Ukrainian IT company TOV "CIES" (CS) [15], which specializes in the development of banking software. To collect metrics, 20 test runs were performed with emulation of the client load on the system in 10 parallel requests per second for 2 hours. At the same time, statements of various volumes were generated.

The following metrics were measured:

1) request execution time;

2) throughput;

3) CPU load of Report Service – the main statement geng eration service;

4) percentage of requests that ended with an error;

5) average recovery time after failures.

Table 1 gives a comparison of the average statement generation time for the current typical algorithm and the proposed microservice solution. The current algorithm is synchronous and implemented in the e-Banking system, which is built on the basis of MA. The proposed microservice solution provides a fully asynchronous statement generation mode, which is additionally divided into subtasks. The total statement generation time for the proposed solution was calculated as the sum of the execution time of each of the subtasks.

Table 1

Statement formation time under load

Number of pages	Time for a monolith-based solution (ms)	Time for a microservices-based solution (ms)
1	212	169
5	790	570
10	1410	995
20	3750	2880
50	8100	5100
100	16150	7160
150	23800	11900
500	80500	41700
850	121500	76100

The calculation of the throughput indicator per unit of time is performed as the ratio of the number of parallel sessions to the response time. Table 2 gives the increase in the corresponding indicator. It should also be noted that when the number of parallel requests to the system is reduced to 1, it was observed that the throughput is almost the same.

Table 2

Statement generation throughput under load

Number of pages	Throughput for a monolith-based solution (request/second)	Throughput for a microservices-based solution (request/second)	Throughput gains for a microservices-based solution
1	47.17	59.17	25.4%
5	12.66	17.54	38.5%
10	7.09	10.05	41.7%
20	2.67	3.47	30.0%
50	1.23	1.96	59.3%
100	0.62	1.40	125.0%
150	0.42	0.84	100.0%
500	0.12	0.24	100.0%
850	0.08	0.13	62.5%

Fault tolerance testing showed that the total time required to restore the service running under Kubernetes does not exceed 2 minutes. Moreover, more than 1 minute was taken by internal Java and Spring tasks that are performed to initialize the JRE and the context of the web application. Provided that human intervention is not required to resolve the failure, the service restores its performance within this time.

Table 3 gives the percentage of requests that ended in an error.

Table 3

Percentage of requests that ended in error

Number of pages	Error rate for a monolith-based solution	Error rate for a microservices-based solution
1	0.01%	0.01%
5	0.02%	0.02%
10	0.08%	0.03%
20	0.15%	0.04%
50	0.3%	0.07%
100	0.5%	0.11%
150	0.8%	0.17%
500	2%	0.39%
850	5%	0.83%

During the load testing experiment, CPU (Central processing unit) load metrics were also obtained and collected, as shown in Fig. 3. The Prometheus system was used to obtain the metrics, and the Grafana system was used to collect, store, and visualize the metrics.

As can be seen from the charts above, the implementation of the prototype based on MSA provides a more even distribution of resources and almost no critical CPU load even under significant load. On average, a reduction in the CPU load by 12–15% was achieved, and the time of critical CPU load was also significantly reduced, which, according to the recommendations, is determined at 80% [23]. This minimizes the risk of server failures and increases the reliability of e-Banking systems as a whole. Thus, the data from Tables 1, 2 show the positive impact of the proposed MSA approach on the scalability and performance of the e-Banking system; data from Table 3, together with the charts in Fig. 3, serve as indicators of increasing the reliability of its operation.

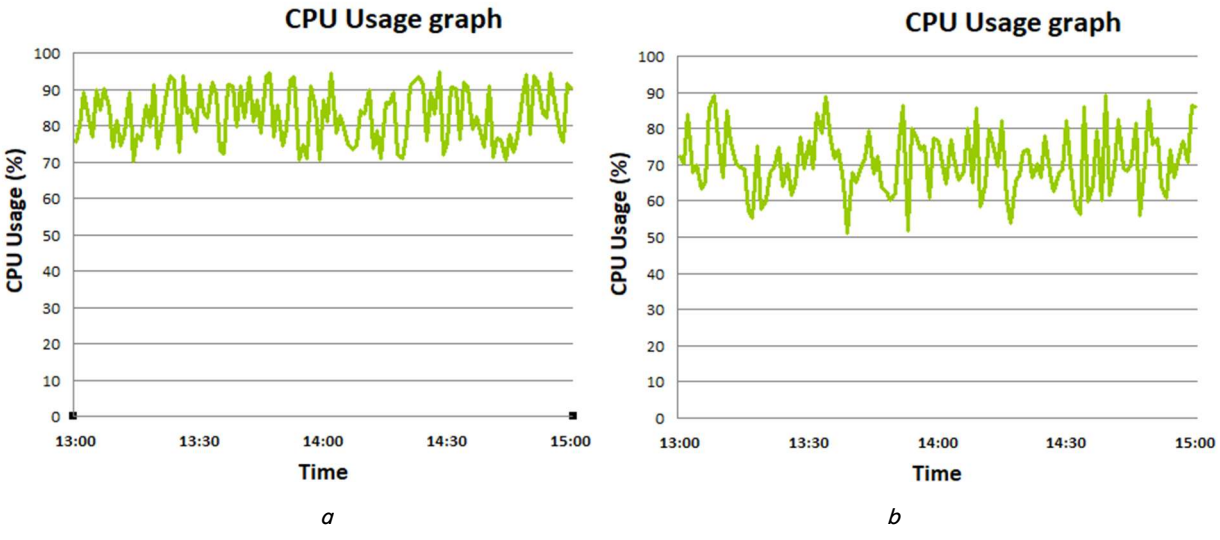

Fig. 3. CPU usage graph: *a* — for monolithic architecture; *b* — for microservice architecture

6. Discussion of the results of research into the process of building a microservice architecture for e-Banking systems

Unlike previous studies [4, 5], which mainly defined general approaches to the design of MSA, the architecture in our work takes into account the peculiarities of the functioning of e-Banking systems. This was achieved by adapting architectural patterns and intelligent methods to the requirements of a specific domain.

The results of the study have confirmed our hypothesis. Conventional approaches to the decomposition of software systems are aimed at the maximum separation of domains and areas of responsibility of services. Our results demonstrated that such approaches do not provide an appropriate level of quality when applied in e-Banking class systems. As the analysis of the peculiarities of the functioning of e-Banking systems revealed, this is explained by the high degree of interdependence of domain data, the requirements for transactional consistency, and the need to perform critical business operations with minimal delay. It was determined that excessive decomposition of the domain area leads to a significant increase in inter-service communication, which negatively affects the performance and reliability of the system.

A prototype of a fragment of the e-Banking system was designed, the architecture of which is shown in Fig. 1, 2; the results of its experimental studies demonstrate that the proposed approach provides a significant improvement in the specified target indicators of system quality. The approach involves using MSA in combination with architectural patterns and intelligent methods. An increase in the productivity indicator (Table 1), a decrease in the delay of operations (Table 2), and an increase in the reliability indicator (Table 3, Fig. 3) were recorded. This was achieved due to distributed processing and the implementation of asynchronous execution of part of the operations. The results confirm that the combination of MSA and knowledge-oriented methods is an effective and high-quality way to overcome previously unexplored problems. The proposed model-technological toolkit (Fig. 2), compared to that proposed in [1], is adapted for use in the system of knowledge-oriented models and intelligent methods. This allowed for the better implementation of the designed MSA (Fig. 2 ,1) and provided appropriate quality attributes of e-Banking systems software. The principles of designing microservice architectures for e-Banking systems were further developed by forming a collection of design patterns that provide the possibility of ensuring higher quality attributes of the functioning of such systems, using a set of quantitative metrics for their assessment.

The results of our study were used in the implementation of the research project at the Ministry of Education and Science of Ukraine "Conceptual models, methods, and technologies for designing adaptive information systems based on knowledge-oriented approaches and software development tools" (No. DR: 0121U110310) in the period from 2022 to 2024. Their testing was carried out in the projects by the Ukrainian IT company "CIES LTD" (the city of Kharkiv). The results were also used in the educational process for lectures, laboratory, and practical work of students majoring in "Computer Science" in the disciplines "Design of service-oriented software systems", "Design of distributed information systems", "Development and maintenance of problem-oriented software systems", and others.

However, the results have certain limitations. The research was carried out only within one target domain of the e-Banking system, namely the formation of account statements. But there are many other complex business processes in the system, such as currency transactions, electronic document management, mass payroll, etc. They may require new constraints on the approaches to decomposition and architecture design as a whole. In addition, the experiment was conducted in a controlled test environment, and the application of the proposed solutions in real banking systems requires additional checks, as it may encounter additional external factors.

The disadvantage of this study is the assumption that only MSA is a way to solve certain problems. Currently, research on the so-called post-MSA is beginning to be conducted in the world.

This research in the future implies the need to investigate other domain areas of e-Banking systems and determine the possibility of applying the proposed approach for them. Alternatively, to propose an adaptation of this approach taking into account the features of these domains.

7. Conclusions

1. The limitations of existing approaches to building MSA when applied in the e-Banking domain have been identified. Formally, in the form of a formula, the basic principle of system decomposition into services was defined, which can be applied in the design of e-Banking systems. The formula shows the relationship between the depth of system decomposition into individual services and such indicators as the complexity of transaction coordination (from the point of view of databases) performed in the domain, the number of network inter-service requests, and the complexity of supporting the business logic of "joint operations".

2. The conceptual principles of MSA design have been systematized; architectural patterns suitable for integration into e-Banking systems were identified, taking into account their specific requirements. In addition, the principles of applying knowledge-oriented and intellectual methods in the processes of design and architecture support were defined.

3. The architecture was designed and a prototype of the e-Banking system fragment was built, which includes IFM that implements a knowledge-oriented approach, namely the SARIMA time series analysis method. It was essentially proved that the combination of MSA, architectural patterns, and intelligent methods provides a significant improvement in the performance indicator, reduces transaction execution delays, and increases the reliability of the system.

4. Experimental studies of the designed prototype of a fragment of a typical e-Banking system, which is responsible for the functionality of building user account statements, were conducted. The results of our study confirmed that the knowledge-oriented intelligent approach to building user account statements proposed in [10] in combination with the advantages of MSA eliminates certain shortcomings and is a promising way to build modern e-Banking systems, which aim to meet modern requirements for software quality indicators. A reduction in the CPU load by 12–15% was achieved; the time of critical CPU load was also significantly reduced. At the same time, the percentage of errors in the execution of the operation of constructing a large-sized statement was reduced by 4 times on average. This minimizes the risk of system failures and makes it possible to draw a motivated conclusion about the increase in the reliability index of e-Banking systems. An increase was also obtained for the system performance index: the statement construction time was reduced by half on average, and the throughput by 50–60% on average. The results show that the proposed MSA for a typical e-Banking system and the model and technological tools necessary for its implementation meet the requirements that are imposed on systems of this class. It is also more effective in comparison with the corresponding implementations using monolithic architectures.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study, as well as the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

All data are available, either in numerical or graphical form, in the main text of the manuscript.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

Authors' contributions

Tymur Daas: Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing; **Mykola Tkachuk:** Conceptualization, Writing – review & editing, Supervision.

References

1. Deshpande, R. A. (2025). Application Of Spring Boot Microservice Architecture for Scaling Banking Applications. The American Journal of Engineering and Technology, 07 (09), 152–158. https://doi.org/10.37547/tajet/volume07issue09-09
2. Daas, T. I. (2025). Design principles and tools to develop of microservice architecture for e-banking systems. Materialy XVI Mizhnarodnoi naukovo-praktychnoi konferentsiyi "Priorytetni shliakhy rozvytku nauky i osvity". Lviv, 51–55. Available at: http://www.lviv-forum.inf.ua/save/2025/29-30.09/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA.pdf
3. Newman, S. (2021). Building Microservices. O`Reilly Media, 612. Available at: https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/
4. Anh, V. N. H. (2024). An Architectural View Model for Designing and Implementing Microservices-based Systems: Use Case in FinTech. Procedia Computer Science, 237, 667–674. https://doi.org/10.1016/j.procs.2024.05.152
5. Meiappane, A., Prasanna Venkatesan, V., Jegatheeswari, V., Kalpana, B., Sarumathy, U. (2013). Pattern Based Adaptive Architecture For Internet Banking. IJITE, 3 (1-2), 287–293. https://doi.org/10.48550/arXiv.1312.2325
6. Elrashidy, M. R., Mansour, H. (2025). Microservices Architecture in Fintech: A Case Study on Scalable Loan Processing with Classical Design Patterns. 2025 Intelligent Methods, Systems, and Applications (IMSA), 382–387. https://doi.org/10.1109/imsa65733.2025.11167186
7. Brosens, J., Kruger, R. M., Smuts, H. (2018). Guidelines for designing e-statements for e-banking. Proceedings of the Second African Conference for Human Computer Interaction: Thriving Communities, 1–6. https://doi.org/10.1145/3283458.3283461

8. Pai, K., Srinivas, B. J. (2024). Enhanced Visibility for Real-time Monitoring and Alerting in Kubernetes by Integrating Prometheus,Prometheus, Grafana, Loki, and Alerta. Interantional journal of scientific research in engineering and management, 08 (06), 1–5. https://doi.org/10.55041/ijsrem35639
9. Moreschini, S., Pour, S., Lanese, I., Balouek, D., Bogner, J., Li, X. et al. (2025). AI Techniques in the Microservices Life-Cycle: a Systematic Mapping Study. Computing, 107 (4). https://doi.org/10.1007/s00607-025-01432-z
10. Daas, T. I., Tkachuk, M. V. (2025). Implementation of time series analysis methods and domain modeling in developing anintelligent forecasting module in the internet banking system. Information Processing Systems, 3 (182), 34–43. https://doi.org/10.30748/soi.2025.182.04
11. Daas, T. I. (2023). Towards domain modeling approach to software development for bank information systems. Materialy XXIII Vseukrainskoi naukovo-tekhnichnoi konferentsiyi molodykh vchenykh, aspirantiv ta studentiv «Stan, dosiahnennia ta perspektyvy informatsiynykh system i tekhnolohiy». Odesa, 183–185. Available at: https://card-file.ontu.edu.ua/items/93ca1c32-c54c-46d7-9c89-a16f1cd5aab9
12. Tomic, M., Dimitrieski, V., Vjestica, M., Župunski, R., Jeremić, A., Kaufmann, H. (2022). Towards Applying API Gateway to Support Microservice Architectures for Embedded Systems. 12th International Conference on Information Society and Technology (ICIST 2022). Kopaonik, 86–91. Available at: https://www.researchgate.net/publication/361952256_Towards_Applying_API_Gateway_to_Support_Microservice_Architectures_for_Embedded_Systems
13. Daraghmi, E., Zhang, C.-P., Yuan, S.-M. (2022). Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture. Applied Sciences, 12 (12), 6242. https://doi.org/10.3390/app12126242
14. Daas, T. I., Tkachuk, M. V. (2025). Intelektualni pidkhody do rozrobky ta suprovodu mikroservisnykh arkhitektur: klasyfikatsiya, osnovni vyklyky ta dosvid zastosuvannia. Zbirnyk naukovykh prats mizhnarodnoi naukovo-tekhnichnoi konferentsiyi «Intelektualni tekhnolohii u mizhdystsyplinarnykh doslidzhenniakh» (ITMD-2025). Kharkiv, 101–104.
15. CS. Available at: https://csltd.com.ua/
16. Ćatović, A., Buzađija, N., Lemes, S. (2022). Microservice development using RabbitMQ message broker. Science, Engineering and Technology, 2 (1), 30–37. https://doi.org/10.54327/set2022/v2.i1.19
17. Yalla, P., Sharma, N. (2015). Integrating Natural Language Processing and Software Engineering. International Journal of Software Engineering and Its Applications, 9 (11), 127–136. https://doi.org/10.14257/ijseia.2015.9.11.12
18. Abdelnabi, E. A., Maatuk, A. M., Abdelaziz, T. M., Elakeili, S. M. (2020). Generating UML Class Diagram using NLP Techniques and Heuristic Rules. 2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), 277–282. https://doi.org/10.1109/sta50679.2020.9329301
19. Tkachuk, M., Zinoviev, D. (2024). Development and investigation of an algorithmic model for adaptive management of software microservices configuration. Information Processing Systems, 2 (177), 107–111. https://doi.org/10.30748/soi.2024.177.12
20. OMG Unified Modeling Language TM (OMG UML). Version 2.5 (2015). Available at: https://www.omg.org/spec/UML/2.5/PDF
21. Statsmodels. Available at: https://www.statsmodels.org/stable/index.html
22. Spring Framework. Available at: https://spring.io/projects/spring-framework
23. Oracle Cloud Infrastructure Documentation. Monitoring. Available at: https://docs.oracle.com/en-us/iaas/Content/Monitoring/Concepts/alarmsbestpractices.htm