https://orcid.org/0000-0002-9374-2833
https://orcid.org/0000-0001-6559-1508

UDC 004.273

H.T. SAMOYLENKO*, A.V. SELIVANOVA*

MICROSERVICE ARCHITECTURE OF THE E-COMMERCE SYSTEM

*State University of Trade and Economics, Kyiv, Ukraine

Анотація. *У статті розглянуто основні вимоги до мікросервісної архітектури в інформаційних системах. Обґрунтовано доцільність використання мікросервісів для інформаційних систем електронної торгівлі. Концепція мікросервісної архітектури полягає в тому, що програмні додатки розробляються як набір незалежних дрібних модульних сервісів і орієнтована на можливості та пріоритети бізнесу. Охарактеризовано особливості взаємодії між сервісами та показники, що впливають на вибір архітектурних рішень. Мікросервісна архітектура відзначається можливістю використовувати різні технології та платформи для окремих сервісів. Зокрема, це дозволяє адаптувати технічні рішення під потреби кожного конкретного сервісу. Визначено функціональні, нефункціональні та бізнес-вимоги до системи електронної торгівлі, запропоновано архітектуру для вебсервісу, що базується на клієнт-серверній та мікросервісній архітектурі, яка дозволяє горизонтально та вертикально масштабувати серверну частину додатка. Мікросервісна архітектура передбачає створення окремих сервісів відповідно до окремих функцій підприємства, що дозволить подальше масштабування та розширення, оскільки кожен мікросервіс може бути розгорнутий незалежно. Сформовано перелік бізнес-вимог для мікросервісів інформаційної системи електронної торгівлі, визначено вимоги для їх подальшої реалізації. Запропоновано архітектуру застосунку системи електронної торгівлі, що складатиметься з мікросервісів, які разом утворюють комплексну систему електронної торгівлі, обґрунтовано інструменти для практичної реалізації. Така архітектура відображає переваги мікросервісного підходу до розробки, оскільки дозволяє ефективно відповідати на зміни в бізнес-потребах, надає можливість масштабування та спрощує підтримку системи, що робить її більш гнучкою й підготовленою до викликів сучасного бізнесу та технологій.*

Ключові слова: *системи електронної торгівлі, архітектура інформаційних систем, мікросервіси, мікросервісна архітектура.*

Abstract. *The article explores the fundamental requirements for microservice architecture in information systems and justifies its suitability for e-commerce information systems. The concept of microservice architecture is based on developing software applications as a collection of independent modular services aligned with business capabilities and priorities. The characteristics of service interaction and factors influencing architectural decisions are discussed. Microservice architecture's flexibility to employ different technologies and platforms for individual services is highlighted, allowing the adaptation of technical solutions to the unique needs of each service. Functional, non-functional, and business requirements for e-commerce systems are defined, and an architectural approach for web services is proposed. This approach is based on a client-server and microservice architecture, enabling both horizontal and vertical scaling of the server-side components. The architecture involves creating separate services to correspond to distinct business functions, facilitating further scalability and expansion since each microservice can be deployed independently. A list of business requirements for microservices in an e-commerce information system, along with the specifications for their implementation, has been formed. An app architecture for the e-commerce system is presented, comprising a collection of microservices that collectively constitute a comprehensive e-commerce system, and the tools for practical implementation are justified. This architecture effectively embodies the advantages of the microservices development approach, allowing the system to dynamically adapt to changing business needs. It provides scalability and simplifies system maintenance, rendering it more agile and prepared to meet the challenges of modern business and technology.*

Keywords: *e-commerce systems, architecture of information systems, microservices, microservice architecture.*

DOI: 10.34121/1028-9763-2024-1-70-76

1. Introduction

Informatization is a significant factor in the impact of technology on e-commerce. The wide application of information technologies in the company's activities ensures the efficiency of its work. Over the past several years, the architectural information system has undergone significant changes, and new approaches and technologies have appeared. One of the choices that developers must make is the choice of architecture, after which it has a decisive impact on the application's performance, scalability, and maintainability.

The aim of the article is to investigate the requirements for designing a microservice architecture for an e-commerce information system and to explore the issues that arise during their implementation.

2. Results of the research

Each type of software architecture has its own benefits and drawbacks. Monolithic applications are easy to deploy because they typically need to be installed on a single server, but such applications have tight internal connections, resulting in limited scalability because they expand as a single unit [1]. Microservice architecture is an innovative way of developing software systems, the essence of which is that software applications are developed as a set of independent, small modular services [2]. Each service performs its own unique process and communicates with others using a well-defined and easy interaction mechanism. The way in which services interact with each other is determined by the requirements of the application; there are various protocols and data exchange formats for this purpose. HTTP/REST is a popular choice for many microservice architectures due to its simplicity and standardization. REST uses HTTP methods such as GET, POST, PUT, and DELETE to perform operations on resources and uses URL paths to identify those resources.

Software designed as microservices can naturally be divided into a series of service components. Each of these services can be deployed, configured, and redistributed independently without compromising the integrity of the overall program [2, 3]. Microservice architecture focuses on business capabilities and priorities, differing from the traditional monolithic development approach. Microservices handle requests, process them, and formulate responses. They have smart endpoints that process information, execute logic, and gateways through which information is disseminated. This approach enables efficient resource utilization and ensures system flexibility and scalability. A distinctive feature of microservices architecture is the ability to use various technologies and platforms for different services. Monitoring microservices plays a key role in detecting and rectifying faults. Tracking metrics and the state of each service helps anticipate failure risks and make rapid decisions to restore system functionality. When designing a microservice architecture, it's crucial to coordinate the methods of interaction between services. There are several key indicators that developers must consider when selecting architectural solutions:

• Interaction style: determines which mechanism of interprocessor communication (IPC) is used between services (it can be HTTP/REST, gRPC, AMQP (Advanced Message Queuing Protocol).

• Detection: determines how the service client learns its IP address or other details necessary for interaction (through registration and detection services, DNS (Domain Name System) or other mechanisms).

• Reliability: how reliable interaction between services is ensured, taking into account the possible unavailability of some of them (mechanisms of re-requests, slow-to-fast retries, or the use of targeted time-out schemes).

• Transactional messaging: how event publishing and messaging integrate with database transactions that update business information.

• External API: defines how application clients will interact with services (REST API, GraphQL, or other interaction interfaces).

There are three main types of requirements that should be highlighted when developing a software product:

1. Business requirements. These are high-level statements about the goals, objectives, and needs of the system. They reflect the desired results and are aimed at achieving business goals.

2. Non-functional requirements describe the general characteristics of the system, such as response time and reliability. They are often called quality attributes and arise from corporate policies, user requirements, budget constraints, etc. These requirements are not directly related to a specific system function.

3. Functional requirements describe the expected behavior of the system and the specific functions that must be implemented by developers. They specify how the product should behave in specific situations and are key to enabling users to complete their tasks.

Functional requirements must be clear and understandable to the development team and all stakeholders. Non-functional requirements are defined by taking into account various factors that are not related to specific system functions. They are important for optimizing the software product. A business requirement (or business domain) represents a task that the organization solves, contributing to the achievement of its strategic goals. For example, processing a basket of orders in an online store is one of the business requirements that allows users to purchase goods via the Internet. Every commercial organization has many such business requirements, which together form its overall business function. Microservice architecture realizes these business opportunities by fully automating their execution. The system is decomposed according to these business requirements, and corresponding services are created to satisfy them. There are two main patterns for decomposing an application into microservices: by business opportunities and by problem areas.

Business capability-oriented decomposition: microservices are grouped based on the business capabilities or functionality they provide. Each service is responsible for a specific business function or opportunity and reflects the company's business structure.

Decomposition by problem areas (domain-driven design): microservices are grouped based on the problem areas they serve. Each service specializes in a specific part of the system or a subject area and helps distinguish different concepts and entities in the system.

When decomposing the system, the principle of single responsibility is applied. The system is broken down based on business requirements, and separate services are created to meet these requirements. An approach to software design based on domain modeling is known as subject-oriented design. Within the subject-oriented design, there is a concept of «bounded context», which defines a part of the subject area where terms and concepts have a specific meaning. An enterprise may have multiple bounded contexts, each of which may include multiple business opportunities [4].

3. Materials and methods

After analyzing the subject area of e-commerce systems, a set of functional requirements for the e-commerce system was formulated:

– the system should provide the ability to register a new user and authorize an existing user;

– the system should provide the possibility of client authorization through social networks and accounts;

– the system must provide access keys to the HTTP API to users;

– the system should provide the ability to view product lists with the option of sorting and review evaluation criteria and the latest actions of customers;

– the system should allow users to add new products via HTTP API, create new actions and associate them with products, and register new customers through a redirect request to social networks or accounts.

Non-functional requirements include performance, security, and implementation requirements.

Performance requirements:

• vertical and horizontal scalability: the system must be able to scale vertically (adding resources to existing servers) and horizontally (adding new servers to the system) to ensure stable operation even with increasing load;

• processing a conditional number of requests at the same time: the system must be able to process a large number of requests from users at the same time without losing performance and speed of response.

Security requirements:

• security of user passwords: user passwords must be protected from reading and must not be stored in an open form;

• protection against browser and server attacks: the system must have mechanisms to protect users from browser attacks such as CSRF (Cross-Site Request Forgery) and XSS (Cross-Site Scripting), as well as from server attacks such as DDoS (Distributed Denial of Service);

• input validation and filtering: the system must validate and filter all input data coming from users to prevent SQL injections and other types of attacks.

Implementation requirements:

• minimization of the client application code: the client application should have a minimal amount of code, which reduces its loading and facilitates a quick response to requests;

• support for common browsers and their current versions: the system must be optimized to support common browsers (e.g., Google Chrome, Mozilla Firefox, Safari) to ensure compatibility and ease of use for users.

In microservice architecture, each service is responsible for performing a specific function or service within the system. This approach allows considering business requirements for each service separately. The following list of business requirements for e-commerce system microservices is proposed:

1. Registration and authorization:

• registration: allows users to create accounts by entering personal details and contact information;

• authorization: checks and confirms the user’s identification to grant access to the personal account;

• social media authentication: allows users to use their social media accounts to log in to the site.

2. Product catalog service:

• storage and presentation of information: stores data about goods, such as name, description, price, image, rating, and other characteristics; provides an interface for viewing goods by buyers;

• accounting of goods: keeping records of the number of goods in the warehouse;

• availability: checks the availability of goods for sale and restocking if necessary.

3. Order management service:

• placing orders: enables customers to place orders by selecting products and specifying a delivery address;

• status of orders: provides the ability to track the status of orders, from receipt to shipment;

• payment: processing payments for orders through various payment methods such as bank cards, electronic money, etc.

4. Delivery management service:

• delivery status: provides the ability to track the movement of goods during delivery, including the date of departure and arrival;

• delivery options: shows different delivery options to users with corresponding prices and terms.

5. HTTP API for users:

• requests API: provides access to various user requests, including order details, purchase history, etc.

6. Expansion and scaling service:

• scaling of services: automatically scales individual microservices in case of overload, ensuring stable operation of the system even under high load.

Together, these services form a comprehensive e-commerce system that enables visitors and users to make purchases online.

4. Designing

The interaction of the browser web client and user applications in the e-commerce system is based on the client-server architecture. Client computers provide an interface that allows users of these devices to request services from the server and display the results that the server sends. Ideally, the server provides a standardized and transparent interface to clients that helps encapsulate clients from system details such as hardware and software that provide these services [5]. For example, in this case, the HTTP API is used as an interface for interaction between clients and the server, which makes clients less dependent on specific details of the server system implementation (Fig. 1). Each of the microservices can be tailored to specific enterprise functions, allowing for further scaling and extensibility as each microservice can be deployed independently.

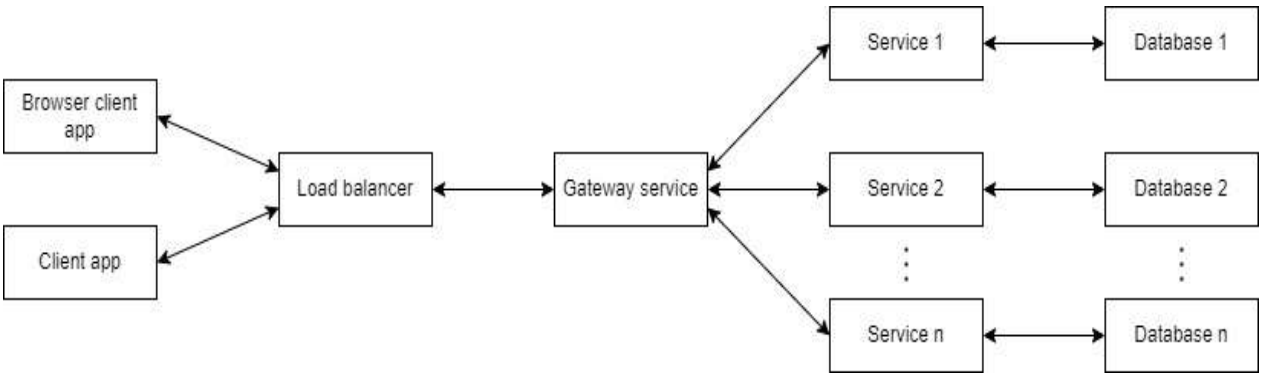

Figure 1 – Proposed architecture of the e-commerce system

The proposed architecture of the e-commerce system consists of the following levels: clients, load balancer, gateway service, services, and databases. Clients that interact with the system can be of different types, including client applications that access recommendations, as well as user web applications [5]. The load balancer is located between the gateway service and the clients, and its main function is to distribute the load between different parts of the system. The gateway service is where the business logic resides and acts as an intermediary between clients and various services, for example, those responsible for authorization processes. Services are responsible for various functional services and provide them for clients. Each service has its own database; this architecture ensures transactional requests, which means that each request or sequence of requests is perceived by the database as a single and indivisible block of operations.

For further practical implementation, it is proposed to develop an e-commerce system application consisting of 6 microservices: product catalog service, order management service, delivery management service, API gateway service, the expansion and scaling service, as well as the registrar service (discovery service) (Fig. 2). It is advisable to develop the client part as a separate program on a single-page application of the Vue.js framework.

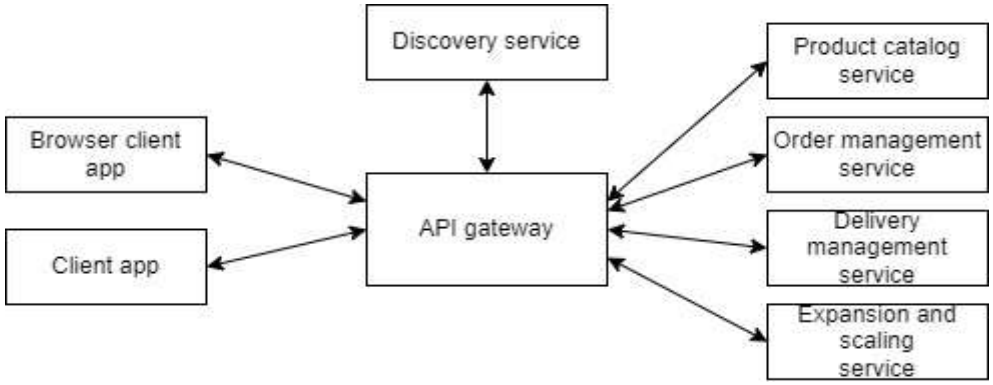

Figure 2 – Proposed application architecture

The presence of Service Discovery allows you to use a declarative REST client, which allows you to wrap HTTP communication between microservices in a Java interface, which makes the code much easier to read [6, 7]. Service Discovery and Feign are often used with the client load balancer – Ribbon. For example, if a conditional microservice has many replicas, then the traffic must be evenly distributed among these replicas. So the service that calls the other one must import spring-cloud-starter-ribbon and specify the load balancing strategy via a short configuration. The registrar service can be implemented using the Service Discovery pattern, which provides an opportunity to use the Feign declarative REST client within the system [7]. This service monitors the status of other services and provides developers with a convenient tool in the form of a dashboard to detect possible system malfunctions. In the future, it will also be possible to implement a client-side Ribbon load balancer. It is advisable to create the API Gateway service in order to provide the system with a single point of entry, thereby reducing possible points for potential attacks. In addition, this service includes user authentication and authorization processes. This architecture reflects the advantages of a microservices approach to application development and implements such common patterns as Service Discovery and API Gateway [5]. Interaction between services will be based on synchronous HTTP requests and asynchronous messaging through the RabbitMQ broker. Zuul provides an opportunity to implement the API gateway pattern, which is a single point of entry into the application [8]. This library provides an interface for configuring appropriate routings to other microservices by service name in the Spring Cloud system or by URL. Zuul also provides the ability to apply filters to a request: before sending it to the microservice, after returning a response from the microservice, and when an error occurs in requests [9].

5. Conclusions

In recent years, the architecture of information systems has undergone significant changes, and new approaches and technologies have appeared. Microservice architecture breaks down complex monolithic applications into a set of autonomous services that can be developed and scaled independently of each other. This gives more flexibility and scalability to the system. The work defines both functional and non-functional requirements for the e-commerce system. It is also proposed to use a microservice architecture for the e-commerce system, which allows horizontal

and vertical scaling of the server part. An overview of possible technical implementations based on the proposed architectural solution has been carried out.

REFERENCES

1. ISO/IEC 15288. Systems and software engineering – System life cycle processes. [Valid from 2008-03-18]. 70 p. (International standard).
2. Configure ESLint. URL: https://eslint.org/docs/user-guide/configuring.
3. Pattern: Monolithic Architecture. URL: https://microservices.io/patterns/monolithic.html.
4. Paulk M.C., Weber C.V., Curtis B., Chrissis M.B. et al. The Capability Maturity Model: Guidelines for Improving the Software Process. Boston: AddisonWesley, 2015. 456 p.
5. Amazon API Gateway. URL: https://aws.amazon.com/api-gateway.
6. Spring Cloud. URL: https://spring.io/projects/spring-cloud.
7. Service Registration and Discovery. URL: https://spring.io/guides/gs/service-registration-and-discovery.
8. Spring REST with a Zuul Proxy. URL: https://www.baeldung.com/spring-rest-with-zuul-proxy.
9. Design Patterns Typescript. URL: https://refactoring.guru/design-patterns/typescript.

Стаття надійшла до редакції 15.10.2023