EI-TC Economic Insights – Trends and Challenges Vol. 12 No.1 2023 95 - 103

Adaptive Microservices for Dynamic E-commerce: Enabling Personalized Experiences through Machine Learning and Real-time Adaptation

Gabriela Dobrița (Ene)

Bucharest University of Economic Studies, 6 Piata Romana, 1st District, Bucharest, Romania
https://orcid.org/0000-0002-5992-334X
e-mail: gabriela.ene02@gmail.com

Abstract: *In the rapidly evolving landscape of e-commerce, the need for personalized experiences and real-time adaptation has become paramount. This article proposes a framework for adaptive microservices in the e-commerce domain, leveraging machine learning techniques and real-time adaptation to enhance user experiences and drive business growth. The framework suggests implementing each e-commerce activity as an independent microservice, enabling modular development, scalability, and flexibility. Key microservices such as product catalog management, shopping cart management, order processing, and user management are identified and integrated into an architecture that fosters seamless communication and data exchange. To enable self-adaptation and personalization, a Monitor-Analyze-Plan-Execute (MAPE) loop is introduced. The monitoring phase involves integrating additional services to collect relevant data, including user interactions, search keywords, session data, and feedback. This data is leveraged in the analysis phase, where machine learning techniques are employed to preprocess and extract meaningful insights. Cosine similarity is utilized to calculate similarity scores between user preferences and available products, enabling the generation of personalized recommendations. The proposed approach focuses on connecting the product management microservice to a self-adaptive mechanism, ensuring that the existing infrastructure remains largely untouched. This enables companies to enhance their e-commerce platforms without undergoing significant overhauls or disruptions. Another advantage lies in the flexibility and scalability offered by the modular microservices architecture. Companies can selectively integrate the self-adaptive mechanism into the product management microservice, maintaining the autonomy and independence of other microservices. This flexibility facilitates scalability as businesses expand and encounter growing demands.*

Original research paper

Citation:
Dobrița, G. (2023). Adaptive Microservices for Dynamic E-commerce: Enabling Personalized Experiences through Machine Learning and Real-time Adaptation. *Economic Insights – Trends and Challenges, 12*(1), 95-103.
https://doi.org/10.51865/EITC.2023.01.08

JEL Classification:
O32;O25; L10;

Keywords: *Self-adaptive Recommendation System; Dynamic E-commerce; Microservices*

Introduction

In recent decades, e-commerce platforms have undergone significant transformations driven by advancements in technology, in-depth studies on consumer behavior, and the evolving demands of expanding businesses. Numerous e-commerce platforms have incorporated dedicated mobile applications, thereby facilitating rapid and streamlined product purchases for buyers. Moreover, personalized purchasing experiences have been seamlessly integrated, with marketing campaigns increasingly including tailored targeting strategies based on meticulous customer data analysis, integrating browsing history, preferences, and duration of product interactions.

Consequently, the data volume has surged, necessitating the adoption of sophisticated processing algorithms to profoundly improve the shopping experience and, by extension, streamline business operations. Noteworthy examples include recommendation systems, customer support chatbots, fraud detection mechanisms, inventory management systems, dynamic pricing algorithms, topic modeling techniques for discerning sentiment from customer reviews, and predictive analytics for anticipating demand patterns on specific products. Taken together, these use-cases highlight the extensive usage of machine learning algorithms and artificial intelligence frameworks in the creation and enhancement of modern e-commerce platforms.

Traditional monolithic architectures have exhibited constraints in scalability, displaying rigidity in adapting to decision-making processes and the integration of emerging technologies. By definition, a monolithic approach to an e-commerce platform entails its development as a cohesive entity, wherein all constituent components are consolidated within a singular codebase, sharing a unified database and infrastructure. The interconnected nature of these components renders the platform susceptible to the propagation of errors, whereby a fault occurring within a specific domain has the potential to reverberate throughout the entirety of the system, thereby introducing amplified risks and augmenting the expenditures associated with development and rigorous testing.

In contrast to the monolithic architecture paradigm, microservices have emerged as a highly favored approach in recent years. The microservices architecture provides resilience, scalability, and flexibility, effectively addressing the inherent limitations of monolithic architectures. The implementation methodology revolves around the partitioning of the application into smaller, self-contained entities, tailored to specific business activities or business domains. These discrete entities can be independently deployed, scaled, and developed. The cohesion among them is maintained through robust communication mechanisms such as REST APIs, event-driven systems, or messaging frameworks. Each microservice operates as an autonomous entity, with its own dedicated database. Any errors that may arise within one microservice remain confined, isolated from the operational integrity of other components, thereby minimizing risks and mitigating costs (Raj & Sadam, 2021). This architectural integrates very well with the DevOps culture (Waseem et al., 2020) and agile methodologies, facilitating accelerated development cycles and advanced testing capabilities.

Even with the incorporation of all these facets, navigating through vast datasets can prove to be a big challenge. The integration of product recommendation systems has become an almost indispensable requirement for customer retention and the augmentation of user engagement. These systems meticulously analyze vast datasets that include browsing histories, product views, time spent within specific sections of the website, as well as user preferences and interests. However, to truly enhance the efficacy and quality of services provided by an online shop, the integration of a self-adaptive recommendation system becomes imperative. This adaptive mechanism considers the fluidity of user preferences, the availability of stock, the integration of new products in the business line, the dynamic pricing landscape and even system restrictions. By continuously adapting to these ever-evolving factors, the self-adaptive

recommendation system ensures the delivery of pertinent, tailored recommendations that align precisely with users' evolving needs and desires, culminating in an elevated user experience.

Literature Review

In (Mendonca et al., 2021) a self-adaptive system is defined as a system that can dynamically monitor and adapt its behavior to preserve and enhance its quality attributes in uncertain operating conditions. The variations of self-adaptation are provided to a system by adding an additional layer that analyzes data regarding the effects of its actions, determines whether they exceed or remain within predefined constraints, and implements a set of modifications to restore the system's state or manage its behavior in a controlled manner. This additional layer simply entails running checks within the loop over the entire system (Seo et al., 2018). The MAPE (Monitor-Analyze-Plan-Execute) loops (Kephart & Chess, 2003) represent a common model used in self-adaptive systems. They enable the system to continuously monitor its state, analyze collected data, plan, and execute adaptation actions based on the analysis results. Monitoring involves gathering real-time information about the system, such as performance metrics, resource status, or user feedback. These data are analyzed in the analysis stage to identify anomalies, trends, or issues that require adaptation. In the planning stage, the system utilizes the analyzed information to develop strategies and adaptation actions. Here, decisions are made regarding the changes needed to improve performance and meet quality objectives. The execution stage involves implementing the planned actions to adjust the system accordingly. These actions may include resource scaling, parameter adjustments, or configuration changes, for example.

The MAPE loops operate in an iterative manner, where the system continuously adapts as it receives and processes new monitoring data. This model allows systems to react in real-time to environmental changes and fluctuating requirements, ensuring performance and reliability in uncertain conditions. Although architecture-based self-adaptation is appealing in theory, it cannot be always easily applied. One challenge is devising solutions that can effectively handle a wide range of systems with diverse characteristics. Additionally, there is a pressing need to explore approaches that minimize the costs associated with integrating external control mechanisms into a system. In (Garlan et al., 2004) those problems were addressed, and the Rainbow framework was introduced. The RAINBOW framework provides a comprehensive approach to designing and implementing adaptive systems. It consists of six key components:

- Reconfiguration: The ability to dynamically modify the system's structure, behavior, and configuration based on changing environmental conditions or goals.
- Architectural model: A formal representation of the system's architecture that captures its components, connectors, and their relationships, enabling analysis and reasoning about system behavior.
- Implementation: The actual realization of the system's components and connectors, incorporating mechanisms for monitoring, analyzing, and executing adaptations.
- Negotiation: The process of making decisions about system adaptations based on multiple stakeholders' requirements, goals, and constraints, considering trade-offs and conflicts.
- Observation: Continuous monitoring and collection of data about the system's internal and external states, including performance metrics, environmental conditions, and user feedback.
- Workbench: A set of tools and techniques that support the development, deployment, and management of self-adaptive systems, including simulation, analysis, and testing capabilities.

In (Cheng et al., 2009) authors outline the requirements for a benchmark environment to enable effective comparisons of the Rainbow framework with other self-adaptive techniques and assess

how well Znn.com meets those requirements, concluding that the rainbow framework proved to provide fast and efficient integration results. In the context of Znn.com, when the response time exceeds a certain threshold, the Rainbow self-adaptive system can take specific actions to improve performance.

Regarding adaptive recommendation systems, few studies describe their integration. In (Huang et al., 2018), a framework is proposed for recommendation suggestions and predicting user behavior in a library context. The framework consists of online and offline components. The online components involve data collection, preprocessing, and data mining. They generate recommendations by analyzing predefined rules by enabling adaptive online recommendation services. The study utilizes the university library as an example, employing the sliding window method, association rule algorithms, and recommendation set generation algorithms. The results demonstrate the effectiveness and feasibility of the proposed method through theoretical analysis and experimentation. In (Sunny et al., 2017), the implementation of a real-time recommendation of TV channels to viewers in real-time system using Apache Spark is described. Similar scenarios are treated and described in (Bui et al., 2022; H. Wan & Yu, 2020; S. Wan & Niu, 2020).

Auto adaptive microservices have the advantage of managing behavior in such a way that they can handle dynamic situations and dynamically allocate resources considering changing operating conditions. The development of such mechanisms also enables the dynamic adjustment of microservices configurations, the adoption of various scaling options, responses provided by the system and the variations of communication between them. In terms of implementation methodology, in (Oreizy et al., 1999), a set of questions is proposed that developers must consider when defining such a system. These questions are presented in the table below (Table 1), along with corresponding answers, aimed at developing a framework of methods and functionalities to enhance the service that handles products through automatic self-adaptation of the recommended product set and dynamic pricing based on specific product demand.

Table 1. Design guidelines for a self-adaptive microservice in the context of an e-commerce website

Question	Answer
Under what conditions does the system undergo adaptation?	The system undergoes adaptation when there is a need to enhance the functionality of the product catalog microservice with product recommendations and dynamic pricing. These conditions should be defined under the business goals, considering changes in user behavior or market trends.
Should the system be open-adaptive or closed-adaptive?	The system should be open-adaptive, allowing for the introduction of new product recommendation algorithms, pricing strategies, and adaptation plans during runtime.
What type of autonomy must be supported?	The system must support a range of autonomy levels, from fully automatic adaptation where the microservice independently adjusts the product recommendations and pricing, to human-in-the-loop adaptation.
Under what circumstances is adaptation cost-effective?	Adaptation is considered cost-effective when the benefits gained from improved product recommendations and dynamic pricing outweigh the costs associated with implementing and maintaining the adaptation mechanisms.
How often is adaptation considered?	Adaptation should be considered based on the specific needs and requirements of the e-commerce system. It can be implemented with policies ranging from continuous adaptation, where real-time user interactions and market changes trigger immediate adjustments, to periodic adaptation, where adaptation is performed at regular intervals based on predefined schedules or business cycles.
What kind of information must be collected to make adaptation decisions? How accurate and current must the information be?	The system needs to collect relevant information such as user preferences, browsing history, purchase patterns, market trends, and competitor analysis. The accuracy and currency of this information should be as up-to-date as possible to ensure accurate analysis and decision-making. Real-time data integration and monitoring mechanisms can be employed to gather the most recent information for adaptation.

Source: Made by author.

Proposed Methodology

Each activity required in an e-commerce system can be implemented as a microservice, as an independent service that handles a specific functionality of the ecommerce platform. The product catalog microservice can be responsible for managing and storing information about the available products, including their attributes, pricing, and inventory. The shopping cart microservice can handle the management of customers' selected items and their quantities, allowing users to add, remove, and update products in their cart. Another microservice could be the order management microservice, which handles the processing of customer orders, including order validation, payment processing, and order fulfillment. The user management microservice can handle user authentication, registration, and account management, ensuring secure access to the platform for both customers and administrators. Each of these microservices can be developed and deployed independently, allowing for easy scalability, modular development, and flexibility in adapting to changing business needs. The microservices can communicate with each other through APIs or messaging protocols to enable data exchange within the ecommerce platform.

An example of such an architecture for an e-commerce platform is presented in Figure 1.

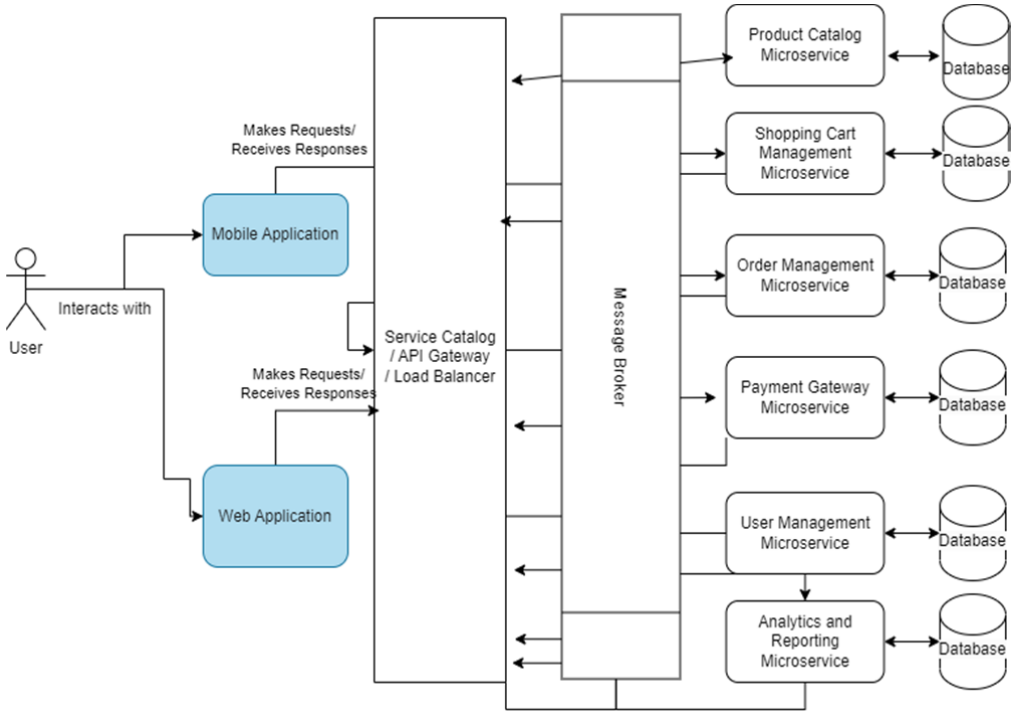

Fig. 1. Microservices architecture for e-commerce platform

Source: Made by author.

To improve and add self-adaptation to the product management microservice, we propose integrating a MAPE loop into the above architecture, as depicted in Figure 2.

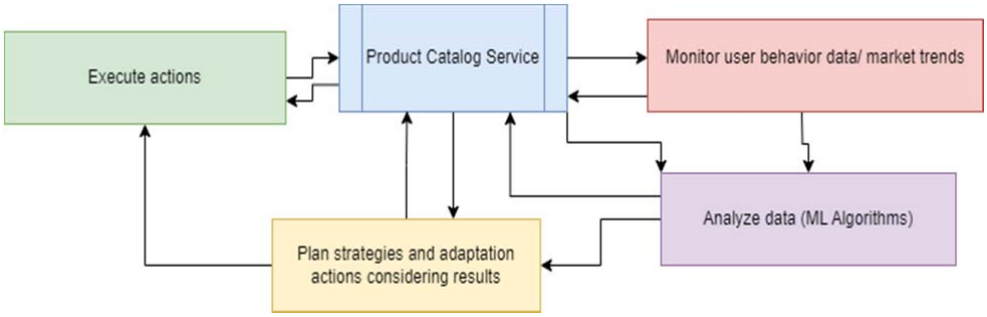

Fig. 2. Monitor-Analyze-Plan-Execute loop interaction with the product management service.

Source: Made by author.

In the monitoring phase, the goal is to integrate additional services under a plug-and-play system for collecting data relevant for the product management service. The following steps were followed for collecting relevant data while minimizing the impact on the platform's codebase and core architecture. Used data sources include server log files from where information about user interactions, search keywords, accessed pages, and session data were extracted. Additionally, tracking code can be injected into JavaScript scripts. Google Analytics provides a convenient mechanism for integration, but customizable solutions can also be built. Furthermore, an event tracking system can be integrated with other services. In this case, various events are considered, such as adding products to the cart, completing orders, or adding product reviews. Questionnaires with relevant questions can be included within or outside the platform to directly collect data on user preferences. Additionally, the classic A/B testing method can generate relevant data by presenting different versions of product presentation pages or pricing variants.

In the analysis phase, the collected data will be preprocessed to prepare it for building the recommendation model. Specifically, the similarity score will be calculated between the attributes related to the names and descriptions of the preferred products and the products added to the cart with the other similar attributes of products in the product catalog database. The textual data was cleaned by removing special characters, converting to lowercase, and handling stop words and, also, textual data was encoded into numerical representations, such as using TF-IDF. Feature vectors were constructed for the preferred products and the products in the catalog using their encoded representations and cosine similarity was used in order to calculate the similarity scores between the feature vectors. The higher the similarity score, the more similar the products are. The products were ranked based on their similarity scores and the first five were selected and returned as possible recommendations for the user. The entire approach is presented in Figure 3.

During the planning phase, different constraints were identified in which the recommendation system should be triggered, and the recommended products should be displayed to the user. These conditions were translated into events that triggered the dynamic adaptation of the product lists, such as adding or removing products from the database, as well as modifying the data related to these products. Other events included new user registrations, the start of promotions, and feedback provided by each user (product reviews, ratings, changes in searched product categories through accessed keywords). These events can be monitored using various mechanisms. Database Triggers can be used to detect events such as adding or removing products, modifying product data, or new user registrations. These triggers can be set up to execute certain actions or notify the system when specific changes occur for notifying the service in charge for data collection that needs to be provided for the recommendation system for the products to be correctly returned. Also, the recommendation system can be configured for webhook notifications from the product service and trigger the appropriate actions in response. If the e-commerce platform integrated with real-time data streams, stream processing

frameworks or technologies like Apache Kafka, Apache Flink, or Apache Spark can be used to analyze the stream and detect events based on predefined rules or patterns.

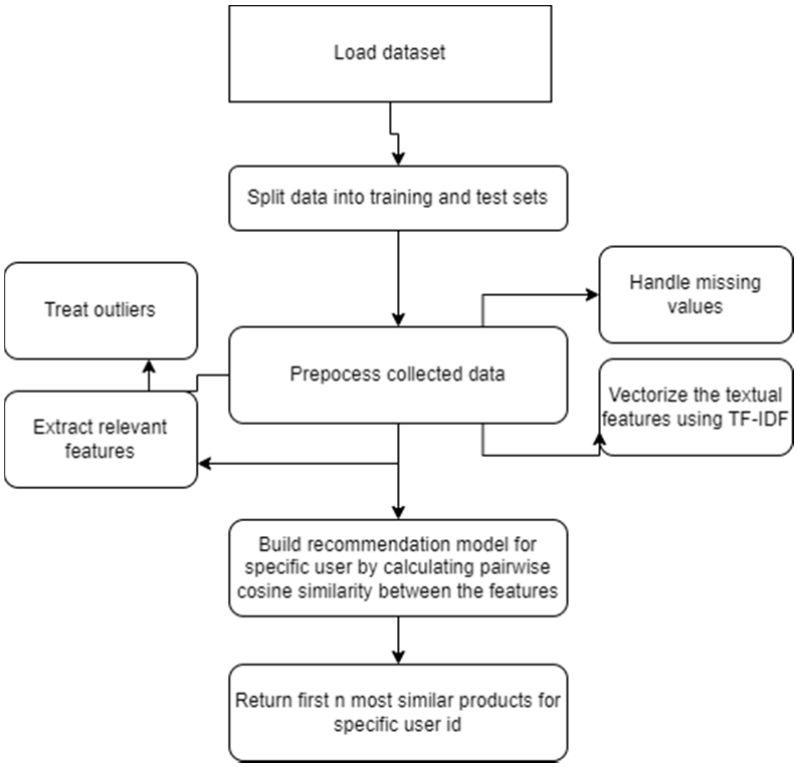

Fig. 3. Simple recommendation model using cosine similarity.

Source: Made by author.

Results and simulations

To measure the impact of adding the self-adaptation loop to the product management service, a series of scenarios were designed for a test environment with 10.000 products for which the response time was tracked. The description of the scenarios is as follows:

Single User: Measure the response time when a single user requests recommendations. The response time of the recommendation system was tested for a user with a small purchase history compared to a user with a large purchase history.

Concurrent Users: Simulate a scenario where multiple users are simultaneously requesting recommendations.

Peak Load: We also wanted to record the response time during peak usage periods when the system experiences high traffic.

Cold Start: Evaluate how quickly the system can generate relevant recommendations for users with minimal information.

Catalog Update: By making changes to the product catalog, such as adding new items or updating existing ones, the response time was tracked.

The results are presented in Table 2. The baseline response time refers to the average response time without the integration of the self-adaptive recommendation system.

Table 2. Response time comparison for different test scenarios

Test Scenario	No. Of Users	Average Response Time (ms) over baseline
Single User	1	100
Concurrent Users	10	360
Peak Load	20	420
Cold Start	1	180
Catalog Update	1	215

Source: Made by author.

The response time measurements indicate the self-adaptation loop does not introduce delays during system startup, or when the product catalog is updated. However, when concurrent users access the recommendation system the waiting time increases, because it handles data in an iterative manner. However, synchronous requests can significantly improve response time in the context of the described system. By utilizing asynchronous processing, the system can handle multiple requests concurrently, reducing overall response time and improving scalability. Additionally, the preprocessing steps involved in data preparation can be optimized to further enhance response time. Techniques such as parallel processing, distributed computing, or optimized algorithms can be employed to expedite data preprocessing and reduce the time required for feature extraction, encoding, and similarity calculations. Furthermore, using caching mechanisms can help improve response time for recurring requests. By storing and retrieving precomputed results or frequently accessed data, the system can avoid redundant calculations and serve responses more quickly.

Conclusions

This article proposes the implementation of an adaptive microservices architecture for dynamic e-commerce platforms. The microservices architecture allows for the modular development and scalability of the system, with each microservice handling specific functionalities such as product catalog management, shopping cart management, order processing, and user management. To enhance the product management microservice, a MAPE (Monitor-Analyze-Plan-Execute) loop was integrated into the architecture. This loop enables self-adaptation by monitoring relevant data, analyzing it, planning appropriate actions, and executing those actions. By leveraging machine learning and real-time adaptation, the recommendation system can provide personalized product recommendations to each user. This enhances the user experience by presenting relevant and tailored suggestions based on their preferences, browsing behavior, and feedback. Personalization increases user engagement, satisfaction, and the likelihood of making a purchase. It also increases cross-selling and up-selling opportunities. The self-adaptive recommendation system can be easily integrated into the existing architecture, providing flexibility to adapt and evolve as business needs change.

Acknowledgements

This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS-UEFISCDI, project number PN-III-P4-PCE-2021-0334, within PNCDI III.

References

1. Bui, S., Kettinger, W. J., & Park, I. (2022). Identity Sharing and Adaptive Personalization Influencing Online Repurchases. *Journal of Computer Information Systems, 62*(4). https://doi.org/10.1080/08874417.2021.1919939

2. Cheng, S. W., Garlan, D., & Schmerl, B. (2009). Evaluating the effectiveness of the rainbow self-adaptive system. *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009*. https://doi.org/10.1109/SEAMS.2009.5069082
3. Garlan, D., Cheng, S. W., Huang, A. C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, *37*(10). https://doi.org/10.1109/MC.2004.175
4. Huang, Z., Li, T., & Xiao, S. (2018). Research on Library Recommendation Reading Service System Based on Adaptive Algorithm. *Wireless Personal Communications*, *102*(2), 1963–1977. https://doi.org/10.1007/s11277-018-5249-9
5. Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41–50. https://doi.org/10.1109/MC.2003.1160055
6. Mendonca, N. C., Jamshidi, P., Garlan, D., & Pahl, C. (2021). Developing self-adaptive microservice systems: Challenges and directions. *IEEE Software*, *38*(2). https://doi.org/10.1109/MS.2019.2955937
7. Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovc, N., Quilici, A., Rosenblum, D. S., & Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, *14*(3), 54–62. https://doi.org/10.1109/5254.769885
8. Raj, V., & Sadam, R. (2021). Performance and complexity comparison of service oriented architecture and microservices architecture. *International Journal of Communication Networks and Distributed Systems*, *27*(1). https://doi.org/10.1504/IJCNDS.2021.116463
9. Seo, Y. D., Kim, Y. G., Lee, E., Seol, K. S., & Baik, D. K. (2018). Design of a smart greenhouse system based on MAPE-K and ISO/IEC-11179. *2018 IEEE International Conference on Consumer Electronics, ICCE 2018*, *2018-January*. https://doi.org/10.1109/ICCE.2018.8326276
10. Sunny, B. K., Janardhanan, P. S., Francis, A. B., & Murali, R. (2017). Implementation of a self-adaptive real time recommendation system using spark machine learning libraries. *2017 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems, SPICES 2017*. https://doi.org/10.1109/SPICES.2017.8091310
11. Wan, H., & Yu, S. (2020). A recommendation system based on an adaptive learning cognitive map model and its effects. *Interactive Learning Environments*, *2020*. https://doi.org/10.1080/10494820.2020.1858115
12. Wan, S., & Niu, Z. (2020). A hybrid e-learning recommendation approach based on learners' influence propagation. *IEEE Transactions on Knowledge and Data Engineering*, *32*(5). https://doi.org/10.1109/TKDE.2019.2895033
13. Waseem, M., Liang, P., & Shahin, M. (2020). A Systematic Mapping Study on Microservices Architecture in DevOps. *Journal of Systems and Software*, *170*. https://doi.org/10.1016/j.jss.2020.110798