Increasing Web-Design Effectiveness Based on Backendless Architecture

Kostiantyn Morozov[0000-0002-6299-4181], Ievgen Sidenko[0000-0001-6496-2469], Galyna Kondratenko[0000-0002-8446-5096], Yuriy Kondratenko[0000-0001-7736-883X]

Intelligent Information Systems Department, Petro Mohyla Black Sea National University, 68th Desantnykiv Str., 10, Mykolaiv, 54003, Ukraine,
morozovknstn@gmail.com, ievgen.sidenko@chmnu.edu.ua,
halyna.kondratenko@chmnu.edu.ua, yuriy.kondratenko@chmnu.edu.ua

Abstract. This paper discusses web-system design and development techniques. For examples would be taken existing methods, approaches and products for the development and design of web systems, such as: monolithic and multi-layer architecture, microservice architecture, serverless architecture, BaaS (Backend as a service) approach. Based on the analysis of these approaches, a new approach will be developed, called DADO (direct approach to data obtaining), on the basis of which the backendless architecture will be created, which is designed to save architects and developers of web systems from many problems and additional costs that lead to other approaches.

Keywords: web-system, client-server, monolithic architecture, multi-layer architecture, microservice architecture, serverless architecture, BaaS, DADO, backendless.

1 Introduction

Nowadays, the problem of proper and effective design is becoming more important than the problem of effective development or support of any web system [1]. This is largely due to the global spread of web applications or applications that somehow work in conjunction in web applications. The load on web systems is growing every day, in an ever-changing world, sudden changes in customer specifications are becoming the norm [2].

With the global spread of the Internet and access to any information virtually in the world, specialties that can be studied without graduating specialized educational institutions have begun to flourish. One such specialty is software development. This entails the saturation of the market with many diverse specialists. Thus, finding high-quality personnel for developer positions is not a big problem. Using various patterns, libraries and frameworks also helps development. However, the initial design of any system is not an easy task, the understanding of which comes with experience and can’t be taught through university classes or video tutorials. That is why the design task is the most important for any system, in large part because changing the system settings during its development can be costly or not possible at all [3-5].

In this work, the popular approaches used in web design and development will be reviewed, their main advantages and disadvantages will be singled out, as well as cases of use will be described. Based on them, with the study of the development of ideas for designing systems, an own approach will be developed, devoid of most of the shortcomings listed in the paper.

2 Related Works and Problem Statement

Within the terms of the work, the following types of systems engineering approaches will be reviewed: monolithic architecture, multi-layer architecture, microservice architecture, serverless architecture and Backend as a Service approach [6].

Monolithic architecture is an approach built on the client-server architectural model (Fig. 1) [7] and based on the idea of the indivisibility of server software. The so-called "monolith" is a single software unit that ensures the operation of the entire system without containing any subsystems or separate components.

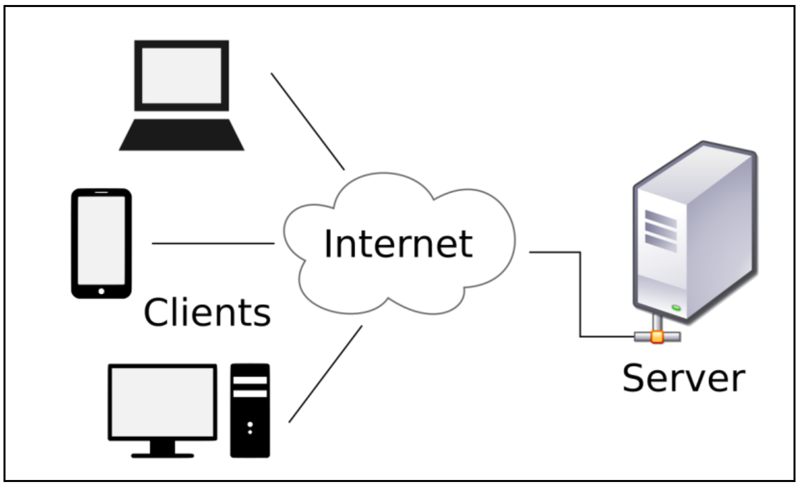


Fig. 1. Typical client-server architectural model [3, 8]

Within the boundaries of this architectural approach, all requests that go to the server (whether requests to a page or asynchronous) will inapplicably fall into one block and will be processed in one data stream. Typically, such an architecture is characterized by the presence of a single database (with the ability to use backup) and monolith not breaking into logical parts or subsystems for separation of duties.

The advantages of this approach are as follows [3]:

1. Due to the absence of need of dividing a single block into smaller parts, the time required to develop the system is reduced;

2. A system designed using a monolithic architecture does not require additional measures for servicing, that is, it does not require many servers or additional software. Such a system can be hosted in a single docker container on one computer and use one single build machine and not require a lot of resources.

However, there are obvious disadvantages in this approach [8, 9]:

1. Due to the lack of dividing the server into subsystems, system support can become very complicated over time. Since such software is usually distributed as a single assembly, the code base tends to become much cluttered, becoming less and less supportable and testable. Also, frequent changes in technical requirements can lead to the fact that the system is redone by force for the necessary needs;
2. In this regard, the scalability of the system becomes extremely low, and often such a system ceases to withstand the increasing load;
3. Since the entire system consists of one large module, a failure in the operation of this module can lead either to a slowdown or to the complete inoperability of the entire system, thereby the fault tolerance of the monolith is extremely low.

Based on the points listed above, we can conclude that this architectural model is suitable for projects in which the development speed prevails over scalability and scalability, or which does not have big plans for expansion, but should be developed in an extremely short time.

Next, a *layered architecture* will be reviewed. It is a development of the idea of monolithic architecture, except that there are so-called layers - subsystems that exist on the principle of separation of responsibilities and usually fulfill three main roles [3, 10]:

– data layer is a layer whose purpose, based on the name, is data manipulation, usually directly through a database or cloud storage. Such layers are characterized by a clear division of objects and classes by roles (client classes, manager classes), as well as the frequent use of various design patterns (repository, unit of work, date transfer object);
– business layer (or business logic layer) is the main layer of the system that implements the technical requirements of the customer. It is this layer that is responsible for how to manage the data that came from the data layer, how to process and combine them, what to convert to, and in what form to give to the client;
– presentation layer is a layer responsible for transferring data to the client. Usually it is a layer that carries out the final manipulations with the data that came from the business layer, makes decisions about how and in what form to combine them, what users with what rights they can be shown to, under what conditions to show certain data and so on. Examples of a presentation layer are controllers from ASP .NET Mvc.

However, the number of layers does not limit with these three layers. The designer’s task is to determine the most optimal number of layers and the tasks that these layers will perform. For example, you can have several business layers that will be divided according to the principle of separation of responsibilities or will be designed for different clients (desktop application, web or mobile). Sometimes instead of layers there may be separate services located on different machines. Then this is called service-oriented architecture [2, 3].

In comparison with monolithic architecture, multilayer has undeniable advantage [10, 11]:

1. The code base ceases to be closely connected with one assembly, acquiring flexibility: now instead of one large module there are at least three smaller ones, the number of coupling inside which is less than in one large one. This means that making changes to such submodules becomes much easier: now changes in one place of the system will affect much less than other places in this system. In this connection, the support of a multilayer system is much simpler than monolithic.

However, there are disadvantages [3, 11]:

1. The fragmentation of one large module by the principle of separation of responsibilities can lead to duplication of code, since the transfer of data between layers may require the same objects, or, when using the so-called data transfer object, the code base can significantly increase;
2. However, there is a more serious problem: due to the fact that the boundaries, for example, between data manipulation logic and business logic, or business logic and presentation logic, is often blurred, serious problems can arise in the logical construction of the system, as a result of which the presentation layer may execute a lot of business logic, or the data layer will begin to process data where it is not needed. This, in turn, can lead to support problems, changes in system structure and scalability.

As a conclusion, it can be said that a multilayer architecture is suitable in many cases when a project can have a big load, and when there are plans for its long-term support.

Next comes the *microservice architecture* [7, 9, 12, 13]. It is the next round in the evolution of multi-layering, combining service-oriented and partitioning the system into subsystems. Microservices are atomic assembly services, united by the principle of business requirements or the logic of the tasks performed, which communicate with each other remotely and can work autonomously from the rest. Each such microservice has its own database, which stores data coming to other services using asynchronous communication channels, such as RabbitMQ, EventGrid or Kafka.

The following briefly describes how the microservice system works. The client (browser, mobile or desktop application) makes a request for any of the microservices. If the client only requests data without changes being made, the service only gives the necessary data (usually in JSON format). However, if a client sends a request to add, delete or modify data, the following happens: the data changes in the database of the service for which the request occurred, then an event is sent through asynchronous communication channels that are tracked by several (or not a single) microservices that perform certain manipulations with data in its database. For example, the Users service received a request to delete a user with id "12345", the service sent an event that the Gifts service received. Now the Gifts service knows that the user has been deleted, and the next gift request for this user will return a 404 response. This principle of operation provides two of the most important features of microservices: fault tolerance and data integrity.

The advantages of microservices are as follows [9]:

1. Since microservices are quite atomic and independent, the number of connections between them is minimal, which greatly simplifies making changes to the system;
2. Since services are usually independent of each other, failure of one (or even several) of them may not affect the operation of the system as a whole;
3. Microservices scale well due to easy integration with various load balancers or proxies.

However, this approach has its drawbacks [12]:

1. The workflow described above is a recommendation from Microsoft [7]. Its implementation will require much more resources than for the development of a monolithic or multi-layer application [9];
2. Consequently, the cost of servicing microservices is increasing. Now, in addition to the need to service one large server, there is a need to service several small ones, as well as host a separate database for each of them;
3. Since microservices are loosely coupled to each other, testing them together can cause significant difficulties, especially if they use asynchronous messaging, which often may not be tested at local developer machine at all.

In the end, microservices are well suited for complex projects that are designed for a large audience, or that work in conjunction with many types of clients (mobile, browsers, computer games). Also, due to its high fault tolerance, they can be used for surveillance systems, medical software, or for highly loaded sites.

The result of the development of microservices is a *serverless architecture* [5, 14-16]. Its essence is that instead of services, even more atomic entities appear, the so-called lambda functions that communicate with each other usually through simple HTTP requests and do not require a separate database or separate machines for hosting them.

The benefits of this approach are as follows [14, 15]:

1. Simplicity in development since there is no need to develop and support large systems, development and support, therefore, are greatly simplified and accelerated;
2. Easy maintenance and deployment - lambda functions do not need dedicated servers to work, as serverless service providers do this.

But serverless has also disadvantages [5, 16]:

1. Since serverless computing services are provided by third parties, the project cost will increase due to the costs of these services;
2. The performance of lambda functions is lower than in the approaches considered above. This is because functions do not exist as a software entity until a request is

made to it. With each request, a function is created and destroyed at its end. This slows down the operation of a full-weighted system.

In conclusion, it worth stating that serverless architecture can hardly make up the backbone of an entire system, but it is well suited as an architecture for a subsystem. For example, lambda functions in production go well with microservices.

The last in this paper will be the *BaaS technology (architecture)* [17-20]. Its essence is that the company provides a ready-made server solution for use. This very solution may have the necessary minimum functionality that developers need: user management, push notifications, multimedia streaming, and so on.

A clear advantage of using this technology is that there is no need to write your own server part, since in fact the finished system will be at the disposal of developers.

However, this approach has obvious disadvantages [3, 17]:

1. Usually the services for using this server are paid, which again increases the cost of the project;
2. The impossibility of making changes to the server code and database is one of the most serious problems of baas, since any deviation of requirements from the services of the company providing the service can seriously harm the whole project. Some companies allow developers to implement their own code, but this contradicts the "backend as a service" approach; in addition, it is simply impossible to test such a code [18];
3. In addition, the inability to manually deploy the server and change its configuration depending on the load. If there is a need for scaling, this service will have additional costs.

This approach is suitable for small projects that will have limited functionality and will not have plans for expansion. Usually as an addition to some other product, which takes all the attention of the customer [2].

At the end of the section, intermediate results on the considered approaches can be conducted and the main task can be formulated. So, the solutions were considered: monolithic architecture, multi-layer architecture, microservice architecture, serverless architecture and BaaS. These approaches have typical advantage-disadvantage pairs. So, monolithic architecture is easy to develop and relatively cheap to maintain, but difficult to scale and maintain. Microservices and serverless, by contrast, are easy to maintain, but are paid and in some cases are difficult to implement and maintain [5].

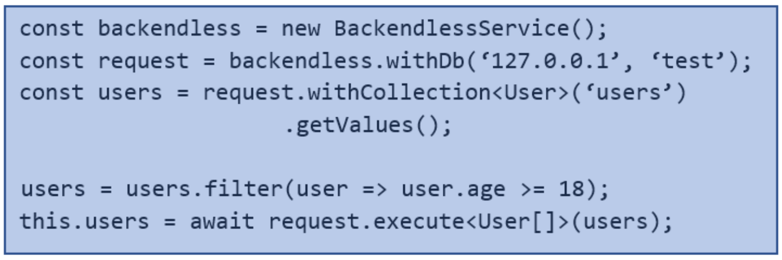
The main task will be to create an approach that will be easy to develop and support, will not require special means of maintenance, and will also not have a high cost. It is also worth considering scalability, fault tolerance and the ability to withstand high loads. In addition, it is worth considering that the world of web development is growing extremely fast, and any lack of approach to system design may simply disappear over time.

3 Concepts of Backendless Architecture

In this section, we will consider a solution developed based on previously discussed methods and approaches to design of web systems.

The architecture, called *backendless (or DADO, Direct Approach to Data Obtaining)* [21-23], is based, as you might guess from the name, based on the complete removal of the server side as an intermediary between the client and the data (database, Redis cache, blob storage) or other data sources. This achieves several effects at once: firstly, the development speed increases markedly, since the development and support of the server side are completely removed from the development cycle. Secondly, productivity is growing: the server, as an additional layer between the client and the data, delays the execution of each request and, therefore, slows down the operation of the whole system.

To implement this approach, a TypeScript language framework was developed that will deal with all queries to the database using a special API, which is very similar to the ordinal JavaScript code that many developers not acquainted with data base technologies are used to (Fig. 2).



```
const backendless = new BackendlessService();
const request = backendless.withDb(‘127.0.0.1’, ‘test’);
const users = request.withCollection<User>(‘users’)
                    .getValues();

users = users.filter(user => user.age >= 18);
this.users = await request.execute<User[]>(users);
```

Fig. 2. TypeScript language framework

The above code performs a backendless request to the local database named ‘test’ and gets all users that are 18 y.o. or older. In the end, developer executes the query that can be any long.

In the center of backendless are the so-called backendless requests, which are made from the client application (Angular, Reactjs, Vuejs) and go directly to the data source (this work takes as a basis work with the database). Thus, the business logic, unique to the server, is divided into a client and a database management system. This leads to an increase in the role of stored procedures or similar structures for a database. A side effect may be an increase in the size of the client application due to the removal of some part of the business logic there.

Summarizing, the advantages of this approach over those considered earlier are as follows:

1. Accelerate development - removing from the development cycle such a large part as the server leads to a significant acceleration of the entire development;

2. Reducing the cost of the project - all the approaches discussed earlier have their own costs for servicing the machines that run the server subsystem. backendless, in turn, is not only a completely free approach, it can also help save money by speeding up development and the absence of the need for a large number of developers to implement it;
3. Increased productivity - as mentioned earlier, removing the server from the data stream will significantly speed up the system as a whole;
4. Simplification of development and support - the absence of a server should facilitate development, since there will be no need for complex system design.

However, there are drawbacks to the approach:

1. The need for a separate client subsystem - such a subsystem can be an application that uses client rendering (Fig. 3) [24], such as Angular or ReactJS. Its essence lies in the fact that navigation on the site and other changes on the pages are made entirely on the client side, and all server requests are AJAX requests. As opposed to client rendering, there is server rendering (Fig. 4) [25];

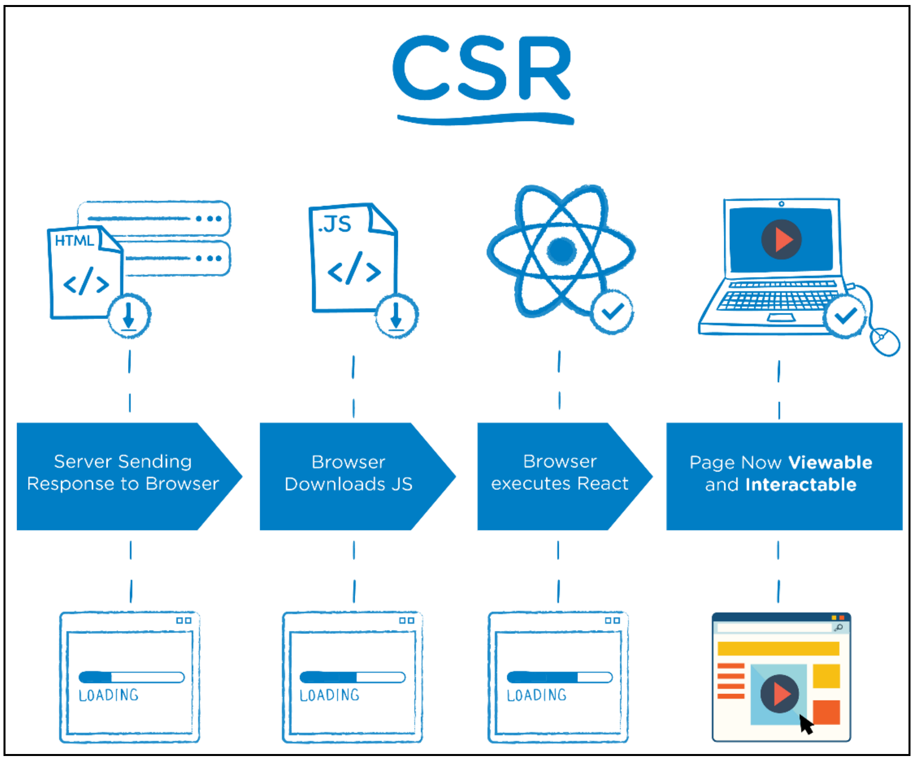


Fig. 3. Scheme of client rendering [3]

2. Putting business logic on the client side - the principle of separation of responsibilities states that each class or component should be responsible for only one function or be strictly separated from the rest logically. The client-server model fully fulfills this requirement, where the client is responsible for displaying data, and the server is responsible for receiving and processing it. However, in some cases, this principle can be neglected, since usually the concepts of business logic are rather vague, and the display of data in a certain way can also be considered business logic. A more significant flaw in bringing logic to the client is to increase the size of the client application, which may be unacceptable for certain users who, for example, have weak Internet. In this case, the competent design of the client subsystem comes to the fore.

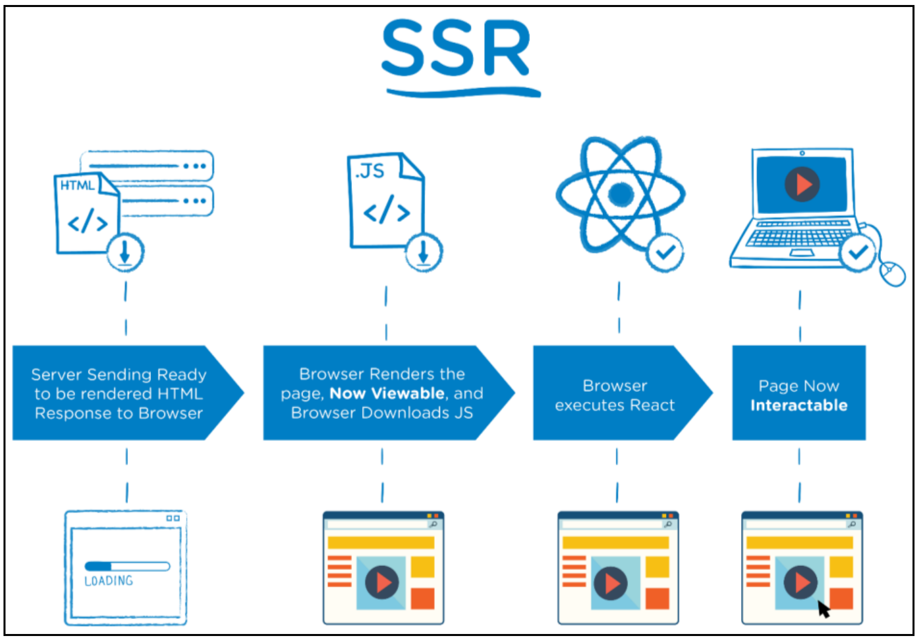

SSR
HTML
.JS
Server Sending Ready to be rendered HTML Response to Browser
Browser Renders the page, Now Viewable, and Browser Downloads JS
Browser executes React
Page Now Interactable
LOADING

Fig. 4. Scheme of server rendering [3]

This paper examines the backendless architecture for working with the MongoDB database as part of a DADO approach to data access. The following is a comparative analysis of each of the considered architectures with backendless (Table 1). Table 1 shows qualitative comparisons of architectures by criteria, since these estimates are difficult to quantify, because much depends on the project. This simplifies the evaluation process, and also avoids linking the assessment to specific project decisions. So, for example, it is quite difficult to quantify architectures by criterion “Speed of development”, because projects can be different in scale, number of developers, their level, etc. Therefore, the estimates in this table are generalized.

Table 1. Comparative analysis of each of the considered architectures with backendless

Criteria / Architectures	Easy to development	Level of maintenance	Level of support	Speed of development
Monolithic architecture	High	High	Low	High
Layered architecture	Medium	High	Medium	High
Microservice architecture	Low	Low	High	Medium
Serverless architecture	Medium	High	High	High
BaaS architecture	High	High	Low	High
Backendless architecture	High	High	High	High

Criteria / Architectures	Flexibility to change technical requirements	Development price	Development prospects
Monolithic architecture	Low	Low	None
Layered architecture	Medium	Low	None
Microservice architecture	High	Medium	Medium
Serverless architecture	High	High	Low
BaaS architecture	Low	High	Medium
Backendless architecture	Medium	Low	High

Summing up, backendless architecture can be useful for startups, where often budget expenses can be more important than system memory costs. Also, for small or educational projects that do not have much coverage. Due to its weak development, backendless will not be able to withstand heavy loads or provide large projects with all the necessary functionality (statistics, caching, etc.).

4 Conclusions

This work presented such principles of designing web systems as monolithic architecture, multi-layer architecture, microservice architecture, serverless architecture and BaaS technology. The strengths and weaknesses of each of them were analyzed, use cases were deduced. It was also analyzed what reasons can stop developers from using one or another approach.

As a result, a new approach to the development and design of web systems was developed, which was called the backendless architecture, or the DADO approach, which says about the complete rejection of the intermediate steps in working with data, which speeds up the application, makes it easier and less confusing to overall, and also allows developers to manipulate the database directly, without using specialized query languages.

References

1. Henderson-Sellers, B., Lowe, D., Haire, B.: OPEN Process Support for Web Development. Annals of Software Engineering 13, 163-201 (2002). DOI: 10.1023/A:1016549527480.

2. Kautz, K., Madsen, S.: Web Development. In: Linger, H. et al. (eds) Constructing the Infrastructure for the Knowledge Economy, pp. 495-505. Springer, Boston, MA (2004). DOI: 10.1007/978-1-4757-4852-9_37.
3. Zambon, G., Sekler, M.: Beginning JSP, JSF, and Tomcat Web Development. Apress, New York (2007). DOI: 10.1007/978-1-4302-0465-7.
4. Putrady, E.: Practical Web Development with Haskell. Apress, Berkeley (2018). DOI: 10.1007/978-1-4842-3739-7.
5. Layka, V.: Learn Java for Web Development. Apress, Berkeley (2014). DOI: 10.1007/978-1-4302-5984-8.
6. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond. Addison-Wesley Professional, Boston (2010). ISBN: 978-0-13-248861-7.
7. Villamizar, M., Garcés, O., Ochoa, L. et al.: Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. SOCA 11, 233-247 (2017). DOI: 10.1007/s11761-017-0208-y.
8. Oquendo, F., Leite, J., Batista, T.: Client Server Architectural Style. In: Software Architecture in Action. Undergraduate Topics in Computer Science, pp. 179-187. Springer, Cham (2016). DOI: 10.1007/978-3-319-44339-3_14.
9. Saransig, A., Tapia, F.: Performance Analysis of Monolithic and Micro Service Architectures – Containers Technology. In: Mejia, J., Muñoz, M., Rocha, Á., Peña, A., Pérez-Cisneros, M. (eds) Trends and Applications in Software Engineering. CIMPS 2018. Advances in Intelligent Systems and Computing, vol. 865, pp. 270-279. Springer, Cham (2018). DOI: 10.1007/978-3-030-01171-0_25.
10. Liu, L., Özsu, M. (eds): Layered Architecture. In: Encyclopedia of Database Systems. Springer, Boston, MA (2009). DOI: 10.1007/978-0-387-39940-9_2941.
11. Datta, U., Lewis, L.: A Layered Architecture for Capacity Planning in Hybrid Networks. In: Lazar, A., Saracco, R., Stadler, R. (eds) Integrated Network Management. Springer, Boston, MA (1997). DOI: 10.1007/978-0-387-35180-3_71.
12. Torre, C., Wagner, B., Rousos, M.: .NET Microservices: Architecture for Containerized .NET Applications. Redmond, Washington (2019).
13. Solesvik, M., Kondratenko, Y.: Architecture for Collaborative Digital Simulation for the Polar Regions. In: Kharchenko, V., Kondratenko, Y., Kacprzyk, J. (eds) Green IT Engineering: Social, Business and Industrial Applications. Studies in Systems, Decision and Control, vol. 171, pp. 517-531. Springer, Cham (2019). DOI: 10.1007/978-3-030-00253-4_22.
14. Vemula, R.: Integrating Serverless Architecture. Apress, Berkeley, CA (2019) DOI: 10.1007/978-1-4842-4489-0.
15. Kondratenko, Y., Kozlov, O., Gerasin, O., Topalov, A., Korobko, O.: Automation of Control Processes in Specialized Pyrolysis Complexes Based on Web SCADA Systems. In: 9th Int. Conf. IDAACS. vol. 1, pp. 107-112 (2017). DOI: 10.1109/IDAACS.2017.8095059
16. Baldini, I. et al.: Serverless Computing: Current Trends and Open Problems. In: Chaudhary, S., Somani, G., Buyya, R. (eds) Research Advances in Cloud Computing, pp. 1-20. Springer, Singapore (2017). DOI: 10.1007/978-981-10-5026-8_1.
17. Kim, C.: A Study of Utilizing Backend as a Service (BaaS) Space for Mobile Applications. In: Lee, R. (eds) Computer and Information Science. ICIS 2019. Studies in Computational Intelligence, vol. 849, pp. 225-236. Springer, Cham (2020).
18. Gropengießer, F., Sattler, K.: Database Backend as a Service: Automatic Generation, Deployment, and Management of Database Backends for Mobile Applications. Datenbank Spektrum 14, 85-95 (2014). DOI: 10.1007/s13222-014-0157-y.

19. Kushneryk P., Kondratenko Y., Sidenko I.: Intelligent dialogue system based on deep learning technology. In: 15th International Conference on ICT in Education, Research, and Industrial Applications: PhD Symposium (ICTERI 2019: PhD Symposium), vol. 2403, pp. 53-62, Kherson, Ukraine (2019).
20. Shurbin, O., Kondratenko, G., Sidenko, I., Kondratenko, Y.: Computerized System for Cooperation Model's Selection based on Intelligent Fuzzy Technique. In: 1st International Workshop on Information-Communication Technologies & Embedded Systems, vol. 2516, pp. 206-217, Mykolaiv, Ukraine (2019).
21. Backendless REST API Documentation. [Online]. Available: https://backendless.com/docs/rest/doc.html.
22. Piller, M.: Backendless Pro is Now Available With Docker Architecture (2018). [Online]. Available: https://backendless.com/the-on-premise-version-is-now-available-with-docker-architecture/.
23. Backendless SDK for .NET API Documentation. [Online]. Available: https://backendless.com/docs/dotnet/.
24. Breux, G.: Client-side vs. Server-side vs. Pre-rendering for Web Apps (2020). [Online]. Available: https://www.toptal.com/front-end/client-side-vs-server-side-pre-rendering.
25. Quax, P., Liesenborgs, J., Barzan, A. et al.: Remote rendering solutions using web technologies. Multimed Tools Appl 75, 4383-4410 (2016). DOI: 10.1007/s11042-015-2481-0.