chiqarishning to‘liq integratsiyalashishiga imkon bermayapti. Bu esa o‘tkazilayotgan izlanishlarning samaradorligini pasaytirmoqda.

Mavjud muammolarni hal qilish, shuningdek, innovatsion faoliyatni qo‘llab-quvvatlash, innovatsion g‘oyalar, ishlanmalar va texnologiyalar, ilmiy yutuqlarni joriy etishni rag‘batlantirish maqsadida 2017-yil 29-noyabrdagi O‘zbekiston Respublikasi Prezidenti tomonidan “O‘zbekiston Respublikasi Innovatsion rivojlanish vazirligini tashkil etish to‘g‘risida”gi Farmon qabul qilindi [6].

Mazkur farmon strategik rejalashtirish tizimini yaratish, davlat boshqaruvining innovatsion shakllarini amalga kiritish, fan va innovatsiya faoliyatini rivojlantirishning zamonaviy infratuzilmasini shakllantirish, investitsiyalarni keng jalb qilish, huquqiy bazani takomillashtirish, ilmiy-tadqiqot va innovatsiya faoliyatini qo‘llab-quvvatlash, uni rag‘batlantirish, ijtimoiy va iqtisodiy hayotning dolzarb sohalariga ilg‘or texnologiyalarni faol joriy etish kabi mamlakat innovatsion rivojlanishining asosiy yo‘nalishlari belgilandi [5].

Innovatsiyani hayotga joriy qilishdan maqsad biror bir ijobiy natijaga erishishdir. Bundan shuni anglash mumkinki, innovatsiya sohasi o‘z-o‘zidan investitsiya sohasi bilan chambarchas bog‘liqligi sababli ta’lim tizimiga ham sarmoyalar kiritilib kelinmoqda. Ta'lim inson rivojlanishi uchun muhim ahamiyatga ega. U shaxsning shakllanishi, hayotiy intilishlari va e'tiqodlarining shakllanishi, odamlarning ma'naviy kamoloti uchun zamin yaratadi. Ta’limda innavatsion g’oyalarni shakllantirish esa ma’naviy kamolotning ham shakllanishi kuchaytiradi. O’quvchilar bilimini oshiradi. Bilimli o’quvchilar o’z navbatida rivojlanib kelayotgan vatanimizni, iqtisodi rivojlangan davlatlar qatoriga olib chiqadi.

Foydalanilgan adabiyotlar ro‘yxati:

1. Prezidentimiz Shavkat Mirziyoyevning 2018 yil 28 dekabrdagi Oliy Majlisga Murojatnomasi. http://uza.uz.
2. Yoldoshev N.Q. va boshqalar. «Innovatsion menejment». Darslik. TDIU. 2011y. — 312 b
3. Alibekov, D. (2020). Socio-philosophical basis of educational system development. ISJ Theoretical & Applied Science, 10 (90), 24-26. Soi: http://s-o-i.org/1.1/TAS-10-90-6 Doi: https://dx.doi.org/10.15863/TAS.2020.10.90.6
4. Alibekov, D. (2021). СОЦИАЛЬНО-ФИЛОСОФСКИЕ ОСНОВЫ РАЗВИТИЯ СИСТЕМЫ ОБРАЗОВАНИЯ. *Журнал музыки и искусства*, *2*(2).

Internet saytlar:

5. https://mineconomy.uz/
6. lex.uz.

MONOLITHIC VS MICROSERVICES ARCHITECTURE

Saparov Khamdam Baxtiyor o'g'li[1], *Matyaqubov Bobur Qutlimurat o'g'li*[2] *Xursandbek Sherxonov Sheripboy o'g'li*[3]

[1,2,3] *Master of Urgench branch of Tashkent University of Information Technologies named after Muhammad al-Khwarizmi*

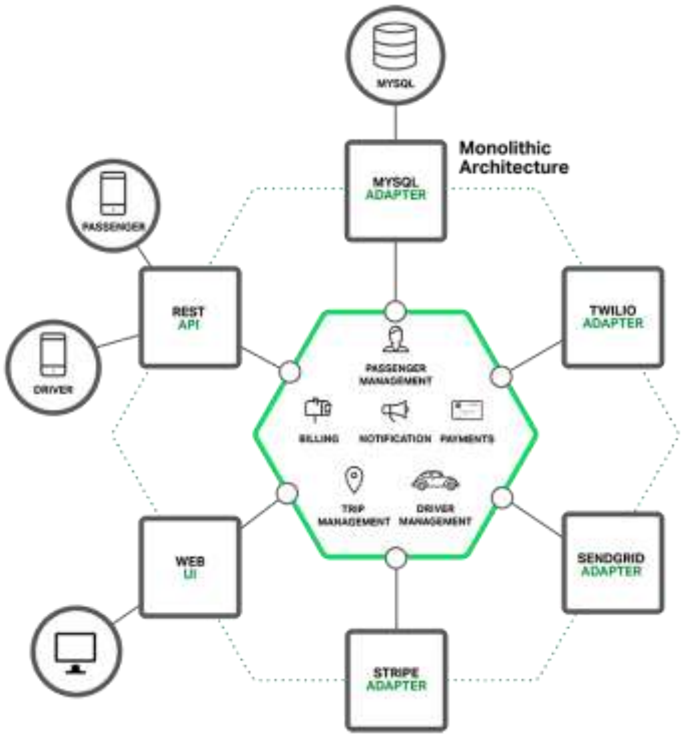
Annotation. Microservices are currently getting a lot of attention: articles, blogs, discussions on social media, and conference presentations. They are rapidly heading towards the peak of inflated expectations on the Gartner Hype cycle. At the same time, there are skeptics in the software community who dismiss microservices as nothing new. Naysayers claim that the idea is just a rebranding of SOA. However, despite both the hype and the skepticism, the Microservices Architecture pattern has significant benefits – especially when it comes to enabling the agile development and delivery of complex enterprise applications.

Keywords : Monolithic Applications**,** Microservices , REST API, Spring Boot, EC2, RPC, Tomcat, Jetty, Similarly, Rails and Node.js, UI with Selenium, SaaS applications, SOA, VM or a Docker container, UI services, API Gateway.

Building Monolithic Applications

Let’s imagine that you were starting to build a brand new taxi-hailing application intended to compete with Uber and Hailo. After some preliminary meetings and requirements gathering, you would create a new project either manually or by using a generator that comes with Rails, Spring Boot, Play, or Maven. This new application would have a modular hexagonal architecture, like in the following diagram: (Picture.1.) [1]

At the core of the application is the business logic, which is implemented by modules that define services, domain objects, and events. Surrounding the core are adapters that interface with the external world. Examples of adapters include database access components, messaging components that produce and consume messages, and web components that either expose APIs or implement a UI.



Picture.1. Monolithic architecture.

Despite having a logically modular architecture, the application is packaged and deployed as a monolith. The actual format depends on the application’s language and framework. For example, many Java applications are packaged as WAR files and deployed on application servers such as Tomcat or Jetty. Other Java applications are packaged as self-contained executable JARs. Similarly, Rails and Node.js applications are packaged as a directory hierarchy.

Applications written in this style are extremely common. They are simple to develop since our IDEs and other tools are focused on building a single application. These kinds of applications are also simple to test. You can implement end-to-end testing by simply launching the application and testing the UI with Selenium. Monolithic applications are also simple to deploy. You just have to copy the packaged application to a server. You can also scale the application by running multiple copies behind a load balancer. In the early stages of the project it works well.

Marching Towards Monolithic Hell

Unfortunately, this simple approach has a huge limitation. Successful applications have a habit of growing over time and eventually becoming huge. During each sprint, your development team implements a few more stories, which, of course, means adding many lines of code. After a few years, your small, simple application will have grown into a monstrous monolith. To give an extreme example, I recently spoke to a developer who was writing a tool to analyze the dependencies between the thousands of JARs in their multi-million line of code (LOC) application. I'm sure it took the concerted effort of a large number of developers over many years to create such a beast.[3]

Once your application has become a large, complex monolith, your development organization is probably in a world of pain. Any attempts at agile development and delivery will flounder. One major problem is that the application is overwhelmingly complex. It's simply too large for any single developer to fully understand. As a result, fixing bugs and implementing new features correctly becomes difficult and time consuming. What's more, this tends to be a downwards spiral. If the codebase is difficult to understand, then changes won't be made correctly. You will end up with a monstrous, incomprehensible big ball of mud.

Another problem with a large, complex monolithic application is that it is an obstacle to continuous deployment. Today, the state of the art for SaaS applications is to push changes into production many times a day. This is extremely difficult to do with a complex monolith since you must redeploy the entire application in order to update any one part of it. The lengthy start-up times that I mentioned earlier won't help either. Also, since the impact of a change is usually not very well understood, it is likely that you have to do extensive manual testing. Consequently, continuous deployment is next to impossible to do.

Monolithic applications can also be difficult to scale when different modules have conflicting resource requirements. For example, one module might implement CPU-intensive image processing logic and would ideally be deployed in Amazon EC2 Compute Optimized instances. Another module might be an in-memory database and best suited for EC2 Memory-optimized instances. However, because these modules are deployed together you have to compromise on the choice of hardware.

Microservices – Tackling the Complexity

Many organizations, such as Amazon, eBay, and Netflix, have solved this problem by adopting what is now known as the Microservices Architecture pattern. Instead of building a single monstrous, monolithic application, the idea is to split your application into set of smaller, interconnected services.

A service typically implements a set of distinct features or functionality, such as order management, customer management, etc. Each microservice is a mini-application that has its own hexagonal architecture consisting of business logic along with various adapters. Some microservices would expose an API that's consumed by other microservices or by the application's clients. Other microservices might implement a web UI. At runtime, each instance is often a cloud VM or a Docker container.

The Microservices Architecture pattern significantly impacts the relationship between the application and the database. Rather than sharing a single database schema with other services, each service has its own database schema. On the one hand, this approach is at odds with the idea of an enterprise-wide data model. Also, it often results in duplication of some data. However, having a database schema per service is essential if you want to benefit from microservices, because it ensures loose coupling.

The Benefits of Microservices

The Microservices Architecture pattern has a number of important benefits. First, it tackles the problem of complexity. It decomposes what would otherwise be a monstrous monolithic application into a set of services. While the total amount of functionality is unchanged, the application has been broken up into manageable chunks or services. Each service has a well-defined boundary in the form of an RPC- or message-driven API. The Microservices Architecture pattern enforces a level of modularity that in practice is extremely difficult to achieve with a monolithic code base. Consequently, individual services are much faster to develop, and much easier to understand and maintain.[1]

The Drawbacks of Microservices

Like every other technology, the Microservices architecture has drawbacks. One drawback is the name itself. The term *microservice* places excessive emphasis on service size. In fact, there are some developers who advocate for building extremely fine-grained 10–100 LOC services. While small services are preferable, it's important to remember that they are a means to an end and not the primary goal. The goal of microservices is to sufficiently decompose the application in order to facilitate agile application development and deployment.[2]

Summary

Building complex applications is inherently difficult. A Monolithic architecture only makes sense for simple, lightweight applications. You will end up in a world of pain if you use it for complex applications. The Microservices architecture pattern is the better choice for complex, evolving applications despite the drawbacks and implementation challenges.

References

1. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith by Sam Newman Paperback Released November 2019[100-108 p]
ISBN: 9781492047841
2. Building Microservices: Designing Fine-Grained Systems by Sam Newman Paperback [64-108 p]
3. Monolith to Microservices: Refactoring Approaches compared: Transforming Applications to could-ready Software Architectures by Jonas Fritzsch /Apr 24, 2018 [64-80 p]
4. Building Event-Driven Microservices: Leveraging Organizational Data at Scale 1st Edition by Adam Bellemare 2020 [189-192 p]

ZAMONAVIY TEXNOLOGIYALARNI KUTUBXONA FAOLIYATIGA TADBIQ QILISH

Tojiyev Alisher Hasan o'g'li
O'zbekiston Milliy universiteti Jizzax filiali o'qituvchisi

Annotatsiya: *Bugungi kunda intellektual tizimlarni kutubxonalarda joriy qilish va foydalanish kutubxona sohasini yangi bosqichga olib chiqadi. Tizim yordamida ma'lumotlarini to'plash, saqlash, nazorat qilish va tartibga solish muhim ahamiyat kasb etadi. Bu orqali kutubxonalarda o'qilayotgan kitoblar, janrlar, yozuvchi va shoirlarga bo'lgan talab, talabalarning qiziqishlari haqidagi dolzarb axborotlarga ega bo'lish, talabalarni o'qishi, qiziqishi, faolligi, dunyoqarashi kabi muhim ko'rsatkichlar bo'yicha turli guruhlarga ajratish, xulosalar chiqarish imkoniyati paydo bo'ladi.*

Kalit so'zlar: *texnologiya, kutubxona, axborot, statistika, tizim.*

Bugungi kunga kelib ijtimoiy-siyosiy, iqtisodiy va ma'naviy-madaniy sohalarda tub o'zgarishlar jarayoni har qachongidan ham jadallashgan. Kundan-kunga axborot kommunikatsiya texnologiyalarining hayotimizdagi roli oshib bormoqda. Mamlakatimizda ulkan yuksalishlar qatorida kutubxona tizimi xalqaro standartlar asosida rivojlanib, tizim faoliyati takomillashayotganiga guvoh bo'lmoqdamiz.

Kundalik turmushimizda deyarli barcha sohalarda ma'lumotlar bilan ishlaymiz. Ma'lumotlarni to'plash, saqlash va uzatishda axborot tizimi orqali katta natijalarga erisha olamiz. Ushbu tizimning vazifasi inson ro'lisiz ma'lumotlarni tahlil qilishdan iborat[5]. Zamonaviy texnologiyalar yordamida kutubxona sohasini yangi bosqichga olib chiqish mumkin. Kutubxona faoliyati kitobxon va kutubxona xodimi o'rtasidagi munosabatlarga asoslanadi. Ushbu faoliyatni tizimlashtirish vositasi yordamida to'plab boriladigan ma'lumotlarni tahlil qilish orqali kutubxona faoliyati va kitobxonlarga turli xil xulosalar berish mumkin[3]. Shunday ekan shu va shu kabi zamonaviy tizimlar yaratish, hayotga tadbiq qilish dolzarb masalalarga aylandi[1].

Ushbu intellektual tizim yordamida kutubxonada kitoblarni joylashtirish va undan qidirish tizimi kutubxonalarda kitoblar bilan ishlashda ma'lumotlarni to'plash, saqlash, nazorat qilish va tartibga solish muhim ahamiyat kasb etadi. Bu orqali kutubxonalarda o'qilayotgan kitoblar, janrlar, yozuvchi va shoirlarga bo'lgan talab, kitobxonlarning qiziqishlari haqidagi dolzarb axborotlarga ega bo'lish, kitobxonlarni o'qishi, qiziqishi, faolligi, dunyoqarashi kabi