Tizim arxitekturasi va loyihalash

Samarqand davlat pedagogika instituti
Abdullayev Ilxom Xujayorovich samexpoler@gmail.com
Nurmurodova Jasmina Davronbek qizi jasminanurmurodova14@gmail.com

Annotatsiya

Tizim arxitekturasi – bu murakkab texnik yoki dasturiy tizimning strukturaviy modelini belgilovchi konsepsiya bo‘lib, komponentlar, ularning o‘zaro bog‘lanishlari, vazifalari va tizim talablari asosida tuziladi. Arxitektura dizayni tizimning yanada barqaror, kengaytiriladigan va test qilinadigan bo‘lishini ta’minlashga xizmat qiladi. Loyihalash jarayoni talablarni aniqlashdan tortib, arxitektura modellari yaratish, dizayn uslublarini tanlash va arxitektura tamoyillarini qo‘llashni o‘z ichiga oladi. Zamonaviy tadqiqotlarda arxitektura dizayni tizim sifatini oshirishda muhim omil sifatida ko‘riladi, bu esa turli uslublar va naqshlar yordamida tizimning funktsional va noo‘rin funktsional talablarini qondirishga imkon beradi [1].

Kalit so‘zlar: tizim arxitekturasi, loyihalash, dasturiy ta’minot dizayni, arxitektura naqshlari, komponentlar, UML.

Tizim arxitekturasi va loyihalash bugungi kunda har bir dasturiy ta'minot yoki axborot tizimi uchun muhim rol o'ynaydi. Yaxshi ishlab chiqilgan tizim arxitekturasi tizimni barqaror, kengaytiriladigan va samarali qilishga yordam beradi.

Tizim arxitekturasining asosiy maqsadi tizimning barcha komponentlarini mantiqan to'g'ri birlashtirish va ularga mos funktsiyalarni taqdim etishdir. Arxitektura dizayni tizimning samarali ishlashini, uning oson kengayishini va ishlashda yuzaga keladigan muammolarga tezda yechim topish imkoniyatini ta'minlaydi [2]. Shuningdek, tizim arxitekturasining to'g'ri tanlanishi xavfsizlikni oshirish, resurslarni optimallashtirish va foydalanuvchi ehtiyojlarini to'liq qondirishda muhim ahamiyatga ega.

Loyihalash jarayoni tizim arxitekturasini yaratishning keyingi bosqichi sifatida, tizimning har bir komponentini detallashtirishni o'z ichiga oladi. Loyihalash – bu tizimning funksional qismlarini va ularning o'zaro aloqalarini aniq belgilash, shuningdek, tizimni amalga oshirishda qanday texnologiyalar va platformalar ishlatilishini tanlash jarayonidir. Tizim loyihalashda arxitektura qarorlaridan tashqari, tizimning barqarorligi, xavfsizligi, ma'lumotlar uzatish tezligi, foydalanuvchi tajribasi kabi muhim jihatlar ham inobatga olinadi.

Tizim arxitekturasini yaratishda va loyihalashda muvaffaqiyatli yondashuvlar ishlab chiqish har qanday tashkilotning muvaffaqiyatli axborot tizimi yaratishidagi asosiy omil hisoblanadi. Tizim arxitekturasini to'g'ri tanlash, uning barqarorligini, ishlashini va kelajakda tizimning rivojlanishini ta'minlaydi. Shu bois, tizim arxitekturasi va loyihalash jarayonlarining muhimligi har qanday sohada, ayniqsa texnologiya va axborot tizimlari bilan ishlovchi sohalarda juda katta ahamiyatga ega.

Arxitektura – tizim uchun reja yoki sxemadir. U tizimning murakkabligini boshqarish va elementlar o‘rtasida aloqa va muvofiqlashtirish mexanizmini o‘rnatish uchun abstraksiya taqdim etadi [4]. Arxitektura texnik va operatsion talablarni qondirishga mo‘ljallangan tuzilgan yechimni

belgilaydi, shu bilan birga umumiy sifat atributlarini, masalan, ish faoliyatini va xavfsizlikni optimallashtiradi. Bundan tashqari, bu dasturiy tizimni rivojlantirish bilan bog‘liq jiddiy qarorlarning to‘plamini o‘z ichiga oladi va har bir qaror sifat, texnik xizmat ko‘rsatish, ish faoliyati va yakuniy mahsulotning muvaffaqiyati bo‘yicha sezilarli ta'sir ko‘rsatadi.

Xususiyatlar kodning operatsion va texnik darajadagi kutishlarini tasvirlaydi. Ishlashning past nosozlikka chidamliligi, miqdoriylik va mas'uliyat kabi asosiy xususiyatlar muhim hisoblanadi. Shuningdek, ular sifat atributlari yoki naqshlar deb ham ataladi. **Microservices** (mikroservislar) - bu dasturiy dizayn naqshlaridan biridir, Event-Driven Pattern (voqeaga asoslangan naqsh), Serverless Pattern (serverga ehtiyoj sezmaslik naqshi) va boshqa ko‘plab naqshlar bilan birga. Mikroservislar naqshi Amazon va Netflix tomonidan qo‘llanilgandan so‘ng nom oldi va katta ta'sir ko‘rsatdi.

Serverless Arxitektura ikki asosiy guruhga bo‘linadi. Birinchisi, "Backend as a Service" (BaaS) yoki "Backend xizmat sifatida" deb ataladi. Ikkinchisi, "Functions as a Service" (FaaS) yoki "Xizmat sifatida funktsiyalar" deb ataladi. Serverless arxitektura sizga ish faoliyati va serverlar bilan bog‘liq vazifalarni hal qilishda ko‘p vaqtni tejashga yordam beradi va kodni ishlab chiqishda va serverlarni joylashtirishda yuzaga keladigan xatolarni tuzatishda yordam beradi [5].

Ushbu arxitektura **Voqea Ishlab chiqaruvchilari (Event Producers)** va **Vaqea Mijozlari (Event Consumers)** ga asoslanadi. Asosiy g‘oya tizimingiz komponentlarini ajratishdir, va har bir qism boshqa bir qismdan biror muhim voqea qo‘zg‘atilganida ishga tushadi. Misol sifatida, bir onlayn do‘kon tizimini olaylik, u ikki komponentdan iborat: birinchi – sotuv moduli, ikkinchisi esa sotuvchi moduli. Agar mijoz sotuv amalga oshirsa, sotuv moduli "buyurtma kutishda" degan voqeani yaratadi. Sotuvchi moduli "buyurtma kutishda" voqeasiga qiziqishi sababli, u ushbu voqeani eshitadi va qo‘zg‘atilgan taqdirda unga javob beradi. Sotuvchi moduli ushbu voqeani olgach, ba’zi vazifalarni bajaradi yoki hatto boshqa voqeani ishlab chiqarishi mumkin, masalan, sotuvchidan ko‘proq mahsulotni buyurtma qilish.

Mikroservislar arxitekturasi hozirgi kunda juda mashhur bo‘lib, u kichik, mustaqil va standart xizmatlarni ishlab chiqishga asoslanadi, bu xizmatlarning har biri o‘ziga xos muammoni hal qiladi yoki o‘zining maxsus vazifasini bajaradi. Ushbu modullar o‘rtasida yaxshi aniqlangan API orqali muloqot amalga oshiriladi va ular biznes maqsadlariga xizmat qiladi [6].

Dastur arxitekturasi dasturiy ta'minotning skeleti va yuqori darajadagi infratuzilmasi uchun javobgar bo‘lsa, dastur dizayni kod darajasidagi dizayn uchun javobgardir, ya'ni har bir modul nima qilayotgani, klasslarning doirasi, funksiyalarning maqsadlari va boshqalar. Har bir tuzilma kod qismlari, ular o‘rtasidagi aloqalar va har bir qismning va aloqaning xususiyatlaridan tashkil topadi. Tizim dizayni tizimning tashkil etilishi yoki tuzilishini tasvirlaydi va uning qanday ishlashini ko‘rsatadi. Dizayn qarorlari ketma-ketligi sifat, ishlash, texnik xizmat ko‘rsatish va tizimning umumiy muvaffaqiyatiga ta'sir qiladi. Tizim elementlar to‘plami bo‘lib, ular ma'lum funksiyalarni bajaradi [7].

Dasturiy ta'minot dizayni – bu dasturiy ta'minotning ishga tushirish komponentlarining ba'zi qismi bo‘yicha abstraksiya hisoblanadi. Komponentlar va ma'lumotlar o‘zaro aloqalarida kerakli xususiyatlar to‘plamini amalga oshirish uchun ishlatiladi.

Element – bu kompyuter kodining abstrakt birligi bo‘lib, ichki holat orqali ma'lumotlarni transformatsiya qiladigan interfeysi orqali ma'lumotlarni o‘zgartiradi. Masalan, transformatsiyalarni quyidagilar tashkil qiladi: yordamchi xotiradan xotiraga yuklash, ba'zi hisoblashlarni bajarish, maxsus formatga tarjima qilish, boshqa ma'lumotlar bilan inkapsulyatsiya qilish va hokazo.

Komponentlar – dasturiy ta'minot dizaynining eng oddiy va aniq tanilgan qismidir, ular jarayonlar, ma'lumotlar va ulanish komponentalarini o‘z ichiga oladi.

Ma'lumot nuqtasi – bu komponentdan uzatilgan yoki komponent tomonidan qabul qilingan ma'lumotdir. Masalan, byte-ketma-ketliklari, xabarlar, marshalled parametrlar va seriyalangan obyektlar, ammo ular doimiy saqlanadigan yoki element ichida yashirin bo‘lgan ma'lumotlar hisoblanmaydi [8].

Bu tamoyil har bir klass faqat bitta maqsadga, bir xil tashvishga va o‘zgarish uchun bir xil sababga ega bo‘lishi kerakligini bildiradi. Ya'ni, klassning o‘z vazifasini aniq belgilash va unga ortiqcha mas'uliyat yuklamaslik kerak. Har bir klass faqat bitta vazifani bajarishi va o‘zgarishi kerak bo‘lgan holatlar faqat bir joyda yuz berishi kerak.

Bu tamoyil klassni kengaytirish uchun ochiq, ammo o‘zgartirish uchun yopiq ekanligini bildiradi. Dizayner klassga yangi funktsiyalar qo‘shishi mumkin, ammo mavjud funksiyalarni o‘zgartirmasligi kerak, shunda bu kodni ishlatadigan boshqa qismlarini buzmasdan yangi imkoniyatlar qo‘shilishi mumkin.

Bu tamoyil dasturchilarga meros olishni to‘g‘ri yo‘l bilan ishlatishga yo‘l ko‘rsatadi, bunda ilova mantiqini hech qanday nuqtada buzmaslik kerak. Ya'ni, agar "XyClass" nomli bola klassi "class" nomli ota klassidan meros olsa, bola klassi ota klassining funktsiyasini qayta yozmasligi kerak, shunda bu ota klassning ishlashini o‘zgartirmaydi. Shu tarzda, siz **Class** obyekti bilan ishlashda ham **Complexity** obyekti bilan ishlaganingiz kabi ilova mantiqini buzmasdan foydalanishingiz mumkin.

Biz bir nechta interfeyslarni amalga oshirishimiz mumkin, so‘ngra kodni shunday tashkil etishimiz kerakki, klass hech qachon o‘z vazifasiga mos bo‘lmagan funktsiyani amalga oshirishga majbur bo‘lmasin. Ya'ni, interfeyslarni to‘g‘ri tasniflash orqali har bir interfeysni faqat kerakli funktsiyalarni taqdim etishga mo‘ljallashimiz kerak [9].

Agar siz dasturiy ta'minotning testlash va modullashuvini ta'minlash uchun TDD (Test-Driven Development) yondashuvini qo‘llagan bo‘lsangiz, unda kodni ajratib, mustaqil qilishning muhimligini bilasiz. Boshqacha qilib aytganda, agar "Purchase" nomli klass "Users" klassiga bog‘langan bo‘lsa, unda **User** obyekti uning ichki ko‘rinishini "Purchase" klassidan tashqarida taqdim etishi kerak.

Dasturiy dizayn tizimning tuzilishini namoyish etadi, shu bilan birga uning amalga oshirilish tafsilotlarini ham ko‘rsatadi. Dizayn shuningdek, tizim ichidagi elementlar va qismlar qanday

o‘zaro aloqada bo‘lishini ham e'tiborga oladi. Dastur dizayni tizimning amalga oshirilish tafsilotlariga chuqur kirib boradi. Dizayn qarorlari ma'lumotlar tuzilmalarini va algoritmlarni tanlash yoki har bir komponentning amalga oshirilishini belgilashga taalluqlidir. Arxitektura va dizayn qarorlari ko‘pincha bir-biriga to‘g‘ri keladi [10]. Dizayn va arxitektura o‘rtasida aniq farqni belgilash uchun qattiq qoidalar va qoidalar mavjud emas. Biroq, ularni birlashtirish mantiqiydir. Ba'zi holatlarda qarorlar ko‘proq dizaynni o‘rganishga yo‘naltirilgan bo‘lishi mumkin, boshqa holatlarda esa qarorlar dizayn va uning qanday amalga oshirilishini tushunishga qaratilgan bo‘ladi.

Mening tadqiqotimga va yuqorida muhokama qilingan nuqtalarga asoslanib, xulosa qilib aytganda, dasturiy dizayn uchun hech qanday aniq qoidalar yoki me'yorlar yo‘q. Muhim jihat shundaki, dizayn – bu uslub, ammo barcha uslubni o‘rganish kerak emas. Amalda dizayner arxitektura dizayni (arxitektura uslubi) va batafsil dizayn (no-arxitektura dizayni) o‘rtasidagi chizig‘ini belgilaydi. Barcha holatlar uchun ishlaydigan qoidalar yoki ko‘rsatmalar mavjud emas, garchi ushbu farqni rasmiylashtirishga urinilsa ham

Foydalanilgan adabiyotlar

[1] (Evelyn J. Barry et. al. 2003) Evelyn J. Barry, Chris F. Kemerer, Sanda A. Slaughter, On the Uniformity of Software Evolution Patterns, IEEE, 2003, pp. 106 – 113

[2] (Kruchten 2000) Philippe Kruchten, “Rational Unified Process – An Introduction”, Addison-Wesley, 2000.

[3] Software Engineering by M.Jaiswal, and S. J. Patel , Thakur publisher, ED 2014, chapter Design Concepts; and Architectural Design pg.100-108, Component-Level design.pg. 122-126.

[4] (Kruchten 1995) Philippe Kruchten, The 4+1 View Model of Architecture, IEEE, 1995, pp. 42 – 50

[5] https://www.sciencedirect.com/topics/computerscience/software-architecture.

[6] R. N. Taylor, *Software Architecture and Design*, Springer, 2019.

[7] W. L. Pantoja Yépez, "Training software architects suiting software industry needs," *Springer*, 2024. Available: https://link.springer.com/article/10.1007/s10639-023-12149-x

[8] I. Fatima, S. Khan, and J. Ahmed, "Software Architecture Assessment for Sustainability: A Case Study," *Springer*, 2023. Available: https://www.springer.com/article/10.1007/s12567-023-00626-7

[9] S. Farshidi, A. Z. Shahrabi, and M. Ali, "Capturing Software Architecture Knowledge for Pattern-Driven Design Decisions," *Elsevier Journal of Systems and Software*, vol. 181, no. 2, 2023. Available: https://www.sciencedirect.com/science/article/pii/S0164121220301552