ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

მარიამ მერებაშვილი

პროგრამული უზრუნველყოფის არქიტექტურის შერჩევის პრინციპები

ინფორმაციული ტექნოლოგიები

ნაშრომი შესრულებულია ინფორმაციული ტექნოლოგიების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

გელა ბესიაშვილი

ტექნიკის მეცნიერებათა კანდიდატი

ასისტენტ პროფესორი

თბილისი

2015

ანოტაცია

ნაშრომში განხილულია პროგრამული უზრუნველყოფის შემუშავების პროცესში არქიტექტურის სწორად შერჩევის როლი და მნიშვნელობა, აგრეთვე ის ძირითადი საფრთხეები და პრობლემები, რომლებიც პროგრამული უზრუნველყოფის არქიტექტურის არასწორმა პროექტირებამ შეიძლება გამოიწვიოს. ნაჩვენებია არქიტექტურის შექმნის მიზნები, განვითარების პერსპექტივები და ჩამოყალიბებულია პროექტირების ძირითადი პრინციპები, რომელთა გათვალისწინება აუცილებელია ხარისხიანი პროგრამული უზრუნველყოფის შესამუშავებლად. წარმოდგენილია ძირითადი ატქიტექტურული სტილები და მათი გამოყენების უპირატესობები და ნაკლოვანებები.

კაცობრიობის განვითარების დღევანდელ ეტაპზე, ნებისმიერი დარგისა თუ საწარმოს წარმატებული ფუნქციონირებისათვის უდიდესი მნიშვნელობა აქვს , მათთვის ხარისხიანი, მოხერხებული, საიმედო და უსაფრთხო პროგრამული უზრუნველყოფის შექმნას. ეკონომიკის განვითარებაში ერთ-ერთ უმნიშვნელოვანეს როლს თამაშობს საბანკო სექტორი, რომელთა წარმატებული ფუნციონირება წარმოუდგენელია ხარისხიანი პროგრამული უზრუნველყოფის გარეშე. ნაშრომში განხილულია კლიენტის პროფაილის (საბანკო პროგრამის) არქიტექტურის შერჩევის მეთოდები პროგრამული არქიტექტურის პროექტირების ძირითადი პრინციპების გათვალისწინებით და შერჩეული არქიტექტურის ძირითადი უპირატესობები. ნაჩვენებია, პროგრამული უზრუნველყოფის სწორად შერჩეული არქიტექტურის მეშვეობით რამდენად მარტივდება და იზრდება კლიენტის მომსახურების სისწრაფე, შესაბამისად იზრდება მწარმოებლურობა და მცირდება მომსახურების დანახარჯები.

Annotation

This article deals with the process of the role and value of developing software architecture rightly; also the main dangers and problems that can lead if architecture of software is protected incorrect. There is a shows the goals of architecture, development perspectives, designing basic principles for the development perspectives, whitch must be foreseen when the quality of software development is needed. There is shows the main styles of architecture and it's advantages and disadvantages.

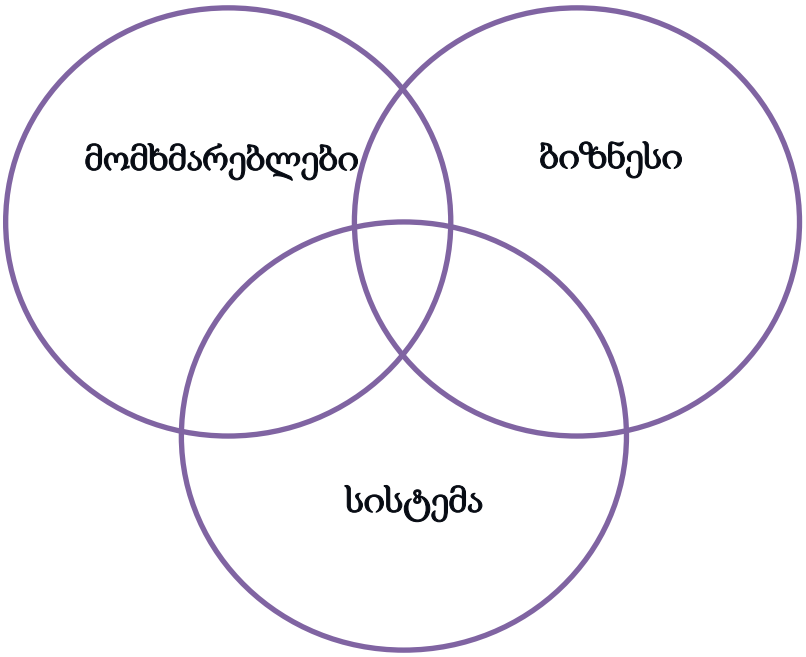
In the present stage of human development, it very important to create the high-quality, convenient, reliable and secure software for successful development in any kind of enterprise. the banking sector plays grate role in the economic development, which successful working is inconceivable without high-quality software . there is considered how to choose the software architecture of of client's profile(banking program), within foreseen maine princeps of project. There is shown how facilitates and increases advantage in customer service the if architecture of softwear is choosed coractly.

შესავალი

მას შემდეგ, რაც პროგრამული უზრუნველყოფის შემუშავება ჩვენი ცხოვრების განუყოფელ ნაწილად იქცა, მასზე მოთხოვნილება მნიშვნელოვნად გაიზარდა. დღესდღეობით მაღალი ხარისხი წარმოადგენს პროგრამული უზრუნველყოფის აუცილებელ კომპონენტს. ხოლო პროგრამული უზრუნველყოფის მაღალი ხარისხის მიღწევა შესაძლებელია, მხოლოდ სწორად შერჩეული არქიტექტურის საშუალებით.

პროგრამული უზრუნველყოფა , როგორც ნებისმიერი სხვა სრუქტურა,იგება მყარ ფუნდამენტზე. საკვანძო სცენარების არასწორმა განსაზღვრამ , საერთო საკითხების არასწორმა პროექტირებამ ან ძირითადი გადაწყვეტილებების გრძელვადიანი შედეგების გამოვლენის უუნაროობამ შეიძლება საფრთხე შეუქმნას მთელ პროგრამას. თანამედროვე ინსტრუმენტები და პლათფორმები ამარტივებენ აპლიკაციის შექმნის ამოცანას, მაგრამ ვერ აღმოფხვრავენ მათი პროექტირების აუცილებლობას კონკრეტული სცენარებისა და მოთხოვნების საფუძველზე. არასწორად შემუშავებული არქიტექტურა იწვევს პროგრამული უზრუნველყოფის არასტაბილურობას, შეუძლებელს ხდის არსებული ან მომავალი ბიზნეს ლოგიკების მხარდაჭერას, წარმოიქმნება პრობლემები დანერგვასა და მართვისას.

სისტემების პროექტირება უნდა ხდებოდეს მომხმარებლის, სისტემის (ITინფრასტრუქტურის) და ბიზნეს მიზნების მოთხოვნების გათვალიწინებით. თითოეული ამ კომპონენტისათვის განისაზღვრება საკვანძო სცენარები და გამოიყოფა ხარისხის მნიშვნელოვანი პარამეტრები (მაგ: საიმედოობა ან მაშტაბურობა). თუ შესაძლებელია , აუცილებლია თითოეულ ამ სფეროში შემუშავდეს და გათვალისწინებულ იქნას წარმატების მაჩვენებლები



ნახ. 1

პროგრამული უზრუნველყოფის ატქიტექტურის შემუშავება უნდა მოხდეს შემდეგი ძირითადი საკითხების გათვალიწინებით:

- როგორ გამოიყენებს მომხმარებელი აპლიკაციას?
- როგორ მოხდება აპლიკაციის დანერგვა და მომსახურება ექსპლუატაციის დროს?
- როგორი მოთხოვნებია წამოყენებული აპლიკაციის ხარისხის მიმართ ,ისეთი ,როგორიცაა უსაფრთხოება, მწარმოებლურობა, პარალელური დამუშავების შესაძლებლობა და კონფიგურაცია ?
- როგორ უნდა მოხდეს აპლიკაციის პროექტირება, რომ ხანგრძლივი დროის განმავლობაში ის რჩებოდეს მოქნილი და მოსახერხებელი მომსახურებაში?
- ძირითადი არქიტექტურული მიმართულებები, რომლებსაც შეუძლიათ გავლენა მოახდინონ აპლიკაციაზე ახლა ან მისი დანერგის შემდეგ?

1. არქიტექტურის მიზნები

არქიტექტურის მიზანია გამოავლინოს მოთხოვნები, რომლებიც გავლენას ახდენს აპლიკაციის სტრუქტურაზე. კარგი არქიტექტურა ამცირებს ბიზნეს რისკებს, რომლებიც დაკავშირებულია ტექნიკური გადაწყვეტილების მიღებასთან. კარგი სტუქტურა მნიშვნელოვნად მოქნილია, რათა გაუმკლავდეს ტექნოლოგიების ბუნებრივ განვითარებას, როგორც მოწყობილობებისა და პროგრამული უზრუნველყოფის სფეროში, ასევე სამომხმარებლო სცენარებში და მოთხოვნებში. არქიტექტორმა უნდა გაითვალისწინოს მიღებული საპროექტო გადაწყვეტილებების საერთო ეფექტი, კომპრომისები ხარისხის ატრიბუტებს შორის (როგორიცაა მწარმოებლურობა და უსაფრთხოება) და კომპრომისები, რომლებიც აუცილებელია სამომხმარებლო, სისტემური და ბიზნეს-მოთხოვნების შესასრულებლად.

აუცილებელია გვახსოვდეს, რომ არქიტექტურამ უნდა შეძლოს:

- გახსნას სისტემის სტრუქტურა, მაგრამ დამალოს რეალიზაციის დეტალები.
- გამოყენებისა და სცენარების ყველა ვარიანტის რეალიზება.
- შეძლებისდაგვარად პასუხობდეს სხვადასხვა დაინტერესებული მხარეების ინტერესებს.
- შეასრულოს როგორც ფუნქციონალური, ასევე ხარისხობრივი მოთხოვნები.

2. არქიტექტურა დღეს და ხვალ

მნიშვნელოვანია განისაზღვროს ძირითადი ფაქტორები, რომლებიც ახდენენ არქიტექტურული გადწყვეტილებების ფორმირებას დღეს და მოახდენენ გავლენას იმაზე, თუ როგორ შეიცვლება არქიტექტურული გადწყვეტილებები მომავალში. ეს ფაქტორები განისაზღვრება მომხმარებელთა სურვილების მიხედვით, აგრეთვე ბიზნესის მოთხოვნებით შედარებით სწრაფი გადაწყვეტილებების მიღებაზე, სამუშაოს სტილისა და სამუშაო პროცესების ცვალებადობის უკეთესი მხარდაჭერით, აგრეთვე პროგრამული უზრუნველყოფის დიზაინის გაუმჯობესებული ადაპტირებით.

განვიხილოთ შემდეგი ძირითადი მიმართულებები:

- მომხმარებლისათვის უფლებამოსილების მინიჭება. მომხმარებლისათვის უფლებამოსილების მინიჭების მხარდაჭერის დიზაინი უნდა იყოს მოქნილი, მორგებული და ორიენტირებული მომხმარებელზე. აპლიკაციები უნდა დაპროექტდეს მომხმარებელთა დონის მიხედვით. ნუ უკარნახებთ , არამედ მიეცით შესაძლებლობა მომხმარებლებს თავად განსაზღვრონ აპლიკაციასთან ურთიერთქმედების სტილი, ამასთან არ გადატვირთოთ არასაჭირო ფუნქციებით და პარამეტრებით, რამაც შეიძლება დააბნიოს მომხმარებელი. განსაზღვრეთ საკვანძო სცენარები და გააკეთეთ ისინი უკიდურესად მარტივი. უზრუნველყავით ინფორმაციის ძებნისა და აპლიკაციის გამოყენების სიმარტივე.
- მოქნილი დიზაინი. სულ უფრო პოპულარული ხდება მოქნილი დიზაინი, რომელიც იყენებს სუსტ კავშირებს, რაც განმეორებითი გამოყენების შესაძლებელობას იძლევა და ამარტივებს მხარდაჭერას. არქიტექტურა მოდულების მიერთების შესაძლებლობით დანერგვის შემდეგ გაფართოების შესაძლებლობას იძლევა. სხვა სისტემებთან ურთიერთქმედების უნზრუნველსაყოფად შეიძლება გამოყენებულ იქნას სერვისზე

ორიენტირებული ტექნიკის უპირატესობების, ისეთის, როგორიცაა SOA (*Service-oriented Architecture*)

3. პროექტირების ძირითადი პრინციპები

პროგრამული უზრუნველყოფის არქიტექტურის შემუშავებისას აუცილებელია გავითვალისწინოთ პროექტირების ძირითადი პრინციპები:

- **ფუნქციების დანაწილება.** აპლიკაციის ცალკეულ კომპონენტებად დაყოფა, რომელთაც შეძლებისდაგვარად მინიმალური ფუნქციონალური თანაკვეთა ექნებათ.
- **პასუხისმგებლობის ერთადერთობის პრინციპი.** ყოველი ცალკე აღებული კომპონენტი თუ მოდული პასუხისმგებელია მხოლოდ ერთ კონკრეტულ თვისებაზე/ფუნქციაზე ან ერთმანეთთან დაკავშირებულ ფუნქციებზე.
- **მინიმალური ცოდნის პრინციპი.**კომპონენტისთვის ან ობიექტისთვის არ უნდა იყოს ცნობილი სხვა კომპონენტების შიდა დეტალები.
- **აპლიკაციაში არ უნდა მოხდეს ფუნქციონალურობის დუბლირება.** ნებისმიერი სახის ფუნქციონალურობა უზრუნველყოფილი უნდა იყოს ერთი კომპონენტით. ეს ამარტივებს მის ოპტიმიზაციას ფუნქციონალურობის ცვლილების შემთხვევაში. ფუნქციონალურობის დუბლირებამ შეიძლება გაართულოს ცვლილებების შეტანა და აპლიკაცია გახადოს ნაკლებად გასაგები.

4. ძირითადი არქიტექტურული სტილები

შემდეგ ცხრილში მოყვანილია არქიტექტურული სტილების სია და თითოეული მათგანის მოკლე აღწერა.

არქიტექტურული სტილი	აღწერა
კლიენტ/სერვერი (*Client/Server*)	სისტემა იყოფა ორ ნაწილად, სადაც კლიენტი ასრულებს მოთხოვნებს სერვერზე. უმეტეს შემთხვევაში სერვერის როლს ასრულებს მონაცემთა ბაზა, ხოლო აპლიკაციის ლოგიკა წარმოდგენილია შენახული პროცედურებით (stored procedure)
კომპონენტური არქიტექტურა (*Component-Based Architecture*)	აპლიკაციის დიზაინი იშლება ფუნქციონალურ ან ლოგიკურ კომპონენტებად , რომლებიც იძლევიან განმეორებით გამოყენების შესაძლებლობას.
მრავალშრიანი არქიტექტურა (*Layered Architecture*)	აპლიკაციის ფუნქციონალური მხარეები ნაწილდება მრავალშრიან ჯგუფებად
n-დონიანი/3-დონიანი არქიტექტურა (*N-Tier / 3-Tier*)	ფუნქციონალი გამოიყოფა ცალკეულ სეგმენტებში, მეტწილად მრავალშრიანი არქიტექტურის ანალოგიურია, მაგრამ მოცემულ შემთხვევაში სეგმენტები ფიზიკურად განსხვავებულ კომპიუტერებზეა განაწილებული .
ობიექტზე ორიენტირებული არქიტექტურა	ობიექტზე-ორიენტირებული არქიტექტურა დაფუძნებულია

(Object-Oriented)	აპლიკაციის ან სისტემის პასუხისმგებლობის დანაწილებაზე დამოუკიდებელ, განმეორებითი გამოყენებისათვის გამოსადეგ ობიექტებად, რომელთაგან თითოეული მოიცავს მონაცემებსა და ქცევებს ამ ობიექტების შესახებ.
სერვისზე ორიენტირებული არქიტექტურა *(Service-Oriented Architecture) (SOA)*	აღწერს აპლიკაციას, რომელიც განსაზღვრავს და მოიხმარს ფუნქციუნალს სერვისების სახით , კონტრაქტების და შეტყობინებების დახმარებით.

4.1. კლიენტ/სერვერის არქიტექტურა

კლიენტ/სერვერის არქიტექტურა აღწერს სისტემის განაწილებას , რომელიც შედგება ცალკეული კლიენტისა და სერვერისა და მათი დამაკავშირებელი ქსელისგან. კლიენტ/სერვერის სისტემის უმარტივესს ფორმას წარმოადგენს ორდონიანი არქიტექტურა. ეს არის სერვერული აპლიკაცია , რომელსაც პიდაპირ მიმართავს უამრავი კლიენტი.

ისტორიულად კლიენტ/სერვერის არქიტექტურა წარმოადგენს desktop აპლიკაციას გრაფიკული სამომხმარებლო ინტრფეისით (user interface, UI) .ის მონაცემების გაცვლისათვის ურთიერთქმედებს მონაცემთა ბაზების სერვეთან . მონაცემთა ბაზებში ბიზნეს ლოგიკა განთავსებულია შენახულ პროცედურებში. თუ განვიხილავთ უფრო განზოგადებულად , კლიენტ/სერვერის არქიტექტურა აღწერს ურთიერთობას კლიენტსა და ერთ ან რამდენიმე სერვერთან , სადაც აგზავნის ერთ ან რამდენიმე მოთხოვნას , ელოდება პასუხს და მიღების დროს ამუშავებს მათ.

დღეისათვის , კლიენტ/სერვერის არქიტექტურის მაგალითს შეიძლება წარმოადგენდეს Web აპლიკაცია, რომელიც გაშვებულია ინტერნეტში ან ინტრანეტში.

კლიენტ/სერვერის არქიტექტურის ძირითად უპირატესობას წარმოადგენს:

- მაღალი უსაფრთხოება. ყველა მონაცემი ინახება სერვერზე, რომელიც , როგორც წესი უზრუნველყოფს უსაფრთხოების უფრო მაღალ კონტროლს, ვიდრე კლიენტის კომპიუტერები.
- მონაცემებთან ცენტრალიზებული წვდომა. რადგან მონაცემები , მხოლოდ სერვერზე ინახება, მონაცემებთან წვდომისა და განახლების ადმინისტრირება უფრო მარტივია, ვიდრე სხვა ნებისმიერ სტილში.
- მომსახურების სიმარტივე. გამოთვლითი სისტემის როლები და პასუხისმგებლობები განაწილებულია რამდენიმე სერვერს შორის, რომლებიც ურთიერთქმედებენ ერთმანეთთან ქსელის საშუალებით. ამის მეშვეობით, კლიენტმა არ იცის და მასზე არ მოქმედებს სერვერზე მიმდინარე მოვლენები (შეკეთება, განახლება ან გადაადგილება).

მიუხედავად ამისა, კლიენტ/სერვერის ტრადიციულ ორდონიან არქიტექტურას აქვს ბევრი ნაკლი, მათ შორის, აპლიკაციის მონაცემებისა და ბიზნეს ლოგიკის მჭიდრო კავშირის ტენდენცია სერვერზე. ამან შეიძლება უარყოფითი გავლენა იქონიოს სისტემის გაფართოებასა და მაშტაბირებაზე,რაც ნეგატიურას აისახება სისტემის საიმედოობაზე . ამ პრობლემის გადასაწყვეტად კლიენტ/სერვერის არქიტექტურა განვითარდა უფრო უნივერსალურ სამდონიან არქიტექტურად, რომელშიც აღმოფხვრილია ზოგიერთი ნაკლოვანება , რომელიც ორდონიანი არქიტექტურისთვისაა დამახასიათებელი.

4.2. კომპონენტური არქიტექტურა

კომპონენტი - ეს არის მზა პროგრამული ობიექტი , რომელიც არსულებს მკაფიოდ განსაზღვრულ ფუნქციათა ნაკრებს და შეუძლია ფუნქციონირება როგორც

დამოუკიდებლად , ასევე შესრულების პროცესში სხვა კომპონენტებთან გაერთიანება. მაგალითად კომპონენტი „კალკულატორი“ შეიძლება გამოყენებულ იქნას დამოუკიდებლად და არაფრით არ განსხვავდებოდეს Windows-ის მიერ მოწოდებული იგივე სახელის პროგრამისაგან , ან შეიძლება თვალის დახამხამებაში მოხდეს მისი ინტეგრაცია ფინანსური პაკეტის შემადგენლობაში(რომელიც , თავის მხრივ აგრეთვე შეიძლება წარმოადგენდეს კომპონენტს).

კომპონენტური არქიტექტურა აღწერს პროექტირებისა და სისტემების შემუშავების მიდგომას პროგრამული უზრუნველყოფის პროექტირების მეთოდების გამოყენებით. ამ შემთხვევაში ძირითადი ყურადღება ექცევა დიზაინის ცალკეულ ფუნქციონალურ და ლოგიკურ კომპონენტებად განაწილებას. აღნიშნული კომპონენტები წარმოადგენენ მკაფიოდ განსაზღვრულ ინტერფეისებს , რომლებიც შეიცავენ მეთოდებს, მოვლენებს და თვისებებს. მოცემულ შემთხვევაში აბსტრაქციის დონე გაცილებით მაღალია, ვიდრე ობიექტზე ორიენტირებული შემუშავებისას და არ ხდება ყურადღების კონცენტრაცია ისეთ საკითხებზე , როგორიცაა კავშირის პროტოკოლები და საერთო მდგომარეობა.

კომპონენტური სტილის ძირითადი პრინციპია ისეთი კომპონენტების გამოყენება, რომლებსაც აქვთ შემდეგი თვისებები:

- განმეორებითი გამოყენების შესაძლებლობა. როგორც წესი , კოპონენტების პროექტირება ხდება მათი განმეორებითი გამოყენების შესაძლებლობით სხვადასხვა სცენარებში და აპლიკაციებში, თუმცა ზოგიერთი კომპონენტი იქმნება სპეციალურად კონკრეტული ამოცანის გადასაწყვეტად.
- შენაცვლებადობა. კომპონენტები ადვილად შეიძლება შიცვალოს სხვა მსგავსი კომპონენტებით.
- კონტექსტისგან დამოუკიდებლობა. კომპონენტების პროექტირება ხდება სხვადასხვა გარემოში და კონტექსტში სამუშაოდ. სპეციალური ცნობები , ისეთი როგორიცაა მონაცემები მდგომარეობის შესახაბ , კომპონენტმა არ უნდა შეცვალოს.

- გაფართოება. კომკონენტმა შეიძლება გააფართოვოს არსებული კომპონენეტები ახალი ქცევის უზრუნველსაყოფად.
- ინკაფსულაცია. კომპონენტები წარმოადგენენ ინტერფეისებს, რომელიც შესაძლებლობას აძლებს გამომძახებელ მხარეს გამოიყენოს მათი ფუნქციონალურობა, ამასთან არ გაამჟღავნოს შიდა პროცესების დეტალები ან შიდა ცვლადები და მდომარეობა.
- დამოუკიდებლობა. კომპონენტები პროექტირდება სხვა კოპონენტებთან მინიმალური დამოკიდებულებით. ამრიგად, კომპონენტები შეიძლება განლაგდეს ნებისმიერ მისაღებ გარემოში სხვა კომპონენტებზე და სისტემებზე ზემოქმედების გარეშე.

როგორც წესი, აპლიკაციებში გამოიყენება სამომხმარებლო ინტერფეისის კომპონენტები(მათ ხშირად უწოდებენ მართვის ელემენტებს), ისეთი როგორიცაა ცხრილები და ღილაკები, აგრეთვე დამხმარე ან მომსახურე კომპონენტები, რომლებიც წარმოადგენენ სხვა კომპონენტებში გამოყენებული ფუნქციების ჯგუფს. გავრცელებული კომპონენტების სხვა ტიპს მიეკუთვნება რესურსტევადი კომპონენტები, რომლებთანაც წვდომა ხორციელდება იშვიათად და რომლეთა აქტივაცია სრულდება „ზუსტად დროში “ (just-in-time, JIT) (ჩვეულებრივ, გამოყენება სცენარებში შორეული ან განაწილებული კომპონენტებით); კომპონენტები რიგითობით, რომელთა მეთოდების გამოძახება ხდება ასინქრონულად , შეტყობინებების რიგითობის გამოყენებით, მათი შენახვისა და გადაგზავნისათვის.

განვიხილოთ კომპონენეტური არქიტექტურული სტილის ძირითადი უპირატესობები:

- დანერგვის სიმარტივე. კომპონენტების არსებული ვერსიები შეიძლება შეიცვალოს ახალი თავსებადი ვერსიებით სხვა კომპონენტებზე ან მთლიანად სისტემაზე გავლენის მოხდენის გარეშე.

- დაბალი ღირებულება. სხვა მწარმოებლების კომპონენტების გამოყენება შესაძლებლობას იძლევა გადანაწილდეს დანახარჯები შემუშავებასა და მომსახურებაზე.
- შემუშავების სიმარტივე. მოცემული ფუნქციონალობის უზრუნველსაყოფად კომპონენტები ახდენენ ფართოდ ცნობილი ინტერფეისების რეალიზებას, რაც საშუალებას იძლება ჩატარდეს შემუშავება სისტემის სხვა ნაწილებზე ზეგავლენის გარეშე.
- განმეორებითი გამოყენების შესაძლებლობა. კომპონენტების მრავალჯერადი გამოყენება ნიშნავს შემუშავებისა და მომსახურების ხარჯების განაწილებას რამდენიმე აპლიკაციას ან სისტემებს შორის.
- გამარტივება ტექნიკური თვალსაზრისით. კომპონენტები ამარტივებენ სისტემას კომპონენტების კონტეინერისა და მისი სერვისების გამოყენებით. კონტეინერის მიერ წარმოდგენილი სერვისების მაგალითად შეიძლება მოვიყვანოთ კომპონენტების აქტივაცია, სიცოცხლის ციკლის მართვა, მეთოდების რიგითობის ორგანიზაცია, მოვლენებისა და ტრანზაქციების დამუშავება.

4.3. მრავალშრიანი არქიტექტურა

მრავალდონიანი არქიტექტურა უზრუნველყოფს ფუნქციუნალურად დამოკიდებული აპლიკაციების დაჯგუფებას სხვადასხვა შრეებში, რომლებიც აგებულია ვერტიკალურად , ერთმანეთის ზემოთ. ყოველი შრის ფუნქციონალობა გაერთიანებულია საერთო როლითა და პასუხისმგებლობით. შრეები სუსტად არის ერთმანეთთან დაკავშირებული და მათ შორის ხორციელდება მონაცემთა გაცვლა. აპლიკაციის სწორი დაყოფა შრეებში უზრუნველყოფს ფუნქციონალობის მკაცრ განაწილებას, რაც თავის მხრივ უზრუნველყოფს მოქნილობას , აგრეთვე მომსახურების სიმარტივეს.

მრავალშრიანი არქიტექტურა აღწერილია, როგორც განმეორებითი გამოყენების გადაბრუნებული პირამიდა, რომელშიც თითოეული შრე აერთიანებს უშუალოდ მის ქვეშ განლაგებული დონის პასუხისმგებლობასა და აბსტრაქციას. შრეებად მკაცრი განაწილების დროს ერთი შრის კომპონენტებს შეუძლიათ ურთიერთქმედება, მხოლოდ იმავე შრის კომპონენტებთან ან იმ შრის კომპონენტებთან , რომლებიც განლაგებულია პირდაპირ მოცემული შრის ქვემოთ. შრეებად შედარებით თავისუფალი განლაგების დროს, კომპონენტებს შეუძლიათ ურთიერთქმედება იგივე ან ყველა ქვემოთ მდებარე შრეებთან.

აპლიკაციის შრეები ფიზიკურად შეიძლება განლაგდეს ერთ კომპიუტერზე (ერთ დონეზე) ან განაწილდეს სხვადასხვა კომპიუტერებზე (n-დონეზე) და სხვადასხვა დონის კომპონენტებს შორის კავშირი ხორციელდება მკაცრად განსაზღვრული ინტეფეისების საშუალებით. მაგალითად , ტიპიური web აპლიკაცია შედგება წარმოდგენის შრისაგან (ფუნქციონალობა, დაკავშირებული UI-თან), ბიზნეს შრისაგან (ბიზნეს-წესების დამუშავება) და მონაცემთა შრისაგან (ფუნქციონალობა, დაკავშირებული მონაცემთა წვდომასთან, რაც პრაქტიკულად სრულად რეალიზდება მონაცემთა წვდომის მაღალდონიანი ინფრასტრუქტურის დახმარებით).

განვიხილოთ მრავალშრიანი არქიტექტურის გამოყენებით პროექტირების საერთო პრინციპები:

- აბსტრაქცია. მრავალშრიანი არქიტექტურა წარმოადგენს ერთ მთლიან სისტემას , რომელიც ამასთანავე უზრუნველყოფს საკმარის დეტალებს ცაკლეული შრეების როლისა და პასუხისმგებლობისა და მათ შორის დამოკიდებულების გასარკვევად.
- ინკაპსულაცია. პროექტირების პროცესში არ არის აუცილებელი რაიმე დაშვებების გაკეთება მონაცემთა ტიპების, მეთოდებისა და თვისებების ან რეალიზაციის შესახებ , რადგან ყველა ეს დეტალი დამალულია შრის ჩარჩოებში.
- მკაფიოდ განსაზღვრული ფუნქციონალური შრეები. ფუნქციონალობის განაწილება შრეებს შორის ძალიან მკაფიოა. ზედა შრეები , ისეთი როგორიცაა

წარმოდგენის შრე , აგზავნიან ბრძანებას ქვედა შრეებში, როგორიცაა ბიზნეს შრე და მონაცემთა შრე. ზედა შრეებს შეუძლიათ ამ შრეებში წარმოქმნილ მოვლენებზე რეაგირება, ამასთან უზრუნველყოფენ მონაცემთა გადაცემის შესაძლებლობას შრეებს შორის ზემოთ და ქვემოთ.

- მაღალი გამართულობა. თითოეული შრისათვის პასუხისმგებლობის საზღვრების მკაფიოდ განსაზღვრა და შრეში მხოლოდ იმ ფუნქციონალობის გარანტირებული ჩართვა , რომელიც პირდაპირ არის დაკავშირებული მის ამოცანებთან უზრუნველყოფს მაქსიმალურ გამართულობას შრის ჩარჩოებში.
- განმეორებითი გამოყენების შესაძლებლობა. ქვედა და ზედა შრეებს შორის დამოკიდებულების არარსებობა უზრუნველყოფს მათი განმეორებითი გამოყენების შესაძლებლობას სხვა სცენარებში.
- სუსტი დამოკიდებულება. შრეებს შორის სუსტი დამოკიდებულების უზრუნველსაყოფად კავშირი მათ შორის დაფუძნებულია აბტრაქციასა და მოვლენებზე.

მრავალშრიანი აპლიკაციების მაგალითად შეიძლება განვიხილოთ ბიზნეს აპლიკაცია (line-of-business, LOB) ,ისეთი როგორიცაა ბუღალტრული აღრიცხვისა და დამკვეთების მართვის სისტემები; საწარმოების web აპლიკაციები და web გვერდები; საწარმოების desktop და smart კლიენტები აპლიკაციის ცენტრალიზებული სერვერებით ბიზნეს ლოგიკის განთავსებისათვის.

მრავალშრიანი აპლიკაციის გამოყენების შესაძლებლობა აუცილებელია განვიხილოთ , თუ ჩვენს განკარგულებაშია სხვა აპლიკაციებში განმეორებითი გამოყენებისათვის შესაფერისი მზა დონეები; თუ გვაქვს აპლიკაციები , რომლებიც გვთავაზობენ შესაფერის ბიზნეს პროცესებს სერვისების ინტერფეისების მეშვეობით; ან თუ იქმნება რთული აპლიკაცია და წინასწარი პროექტირება ითხოვს დანაწილებას, რომ ჯგუფებმა მოახდინონ ფოკუსირება ფუქციონალობის სხვადასხვა ნაწილზე. მრავალშრიანი არქიტექტურა ასევე მართებულია, თუ აპლიკაცია მხარს უჭერს

სხვადასხვა ტიპის კლიენტსა და მოწყობილობას ან თუ საჭიროა რთული ან მორგებადი ბიზნეს წესებისა და პროცესების რეალიზაცია.

4.4. n-დონიანი / 3-დონიანი არქიტექტურა

n-დონიანი და 3-დონიანი არქიტექტურა ბევრ რამეში ანალოგიურია მრავალშრიანი არქიტექტურის, სადაც ფუნციონალობა განაწილებული სეგმენტებზე , მაგრამ მოცემულ შემთხვევაში ეს სეგმენტები ფიზიკურად შეიძლება სხვადასხვა კომპიუტერზე იყოს განლაგებული, მათ უწოდებენ დონეებს.

n-დონიანი არქიტექტურის მახასიათებლებს წარმოადგენს აპლიკაციის დეკომპოზიცია(დაყოფა , დანაწილება), სერვისული კომპონენტები და მათი განაწილებული განლაგება, რაც უზრუნველყოფს მაღალ მასშტაბირებას, ხელმისაწვდომობას, მართვადობას და რესურსების ეფექტურ გამოყენებას. ყოველი დონე სრულიად დამოუკიდებელია სხვა დანარჩენი დონეებისგან, გარდა იმათისა , რომლებიც უშუალოდ მის ზემოთ და ქვემოთ არის განთავსებული. n-ურ დონეს მოეთხოვება მხოლოდ იმის ცოდნა, როგორ დაამუშავოს n+1 დონის მოთხოვნა, როგორ გადასცეს ეს მოთხოვნა n-1 დონეს (თუ ასეთი არსებობს) და როგორ დაამუშაოს მოთხოვნის შედეგები. მასშტაბურობის უკეთესი უზრუნველყოფისათვის კავშირი დონეებს შორის, როგორც წესი, ასინქრონულია. n- დონიან არქიტექტურას , როგორც წესი , აქვს არანაკლებ სამი ცალკეული ლოგიკური ნაწილი, რომელთაგან თითოეული ფიზიკურად განლაგებულია სხვადასხვა სერვერებზე. თითოეული ნაწილი პასუხიამგებელია გარკვეულ ფუნქციონალობაზე. მრავალშრიანი მიდგომის გამოყენების დროს შრე განლაგდება დონეზე, თუ ამ შრით წარმოდგენილი ფუნქციონალობა გამოიყენება ერთზე მეტი სერვისით ან დონის აპლიკაციით.

n-დონიანი/3-დონიანი არქიტექტურული სტილის მაგალითად გამოდგება ტიპიური ფინანსური web-აპლიკაცია უსაფრთხოების მაღალი მოთხოვნებით. მაგალითად, ტიპიური გადატვირთული კლიენტი , რომელშიც წარმოდგენის შრე განლაგებული

კლიენტების კომპიუტერებზე, ხოლო ბიზნეს შრე და მონაცემებთან წვდომის შრე დანლაგებული ერთ ან მეტ სერვერულ დონეებზე.

n-დონიანი/3-დონიანი არქიტექტურული სტილის ძირითად უპირატესობებს წარმოადგენს :

- მხარდაჭერის მოხერხებულობა. დონეები არ არის ერთმანეთზე დამოკიდებული, რაც იძლევა განახლებებისა და ცვლილებების შესრულების შესაძლებლობას აპლიკაციაზე ზემოქმედების გარეშე.
- მასშტაბურობა. დონეების ორგანიზება ხდება შრეების განლაგების საფუძველზე, ამიტომაც აპლიკაციის დამასშტაბება საკმაოდ მარტივია.
- მოქნილობა. თითოეული დონის დამასშტაბება და მართვა შესაძლებელია შესრულდეს დამოუკიდებლად, რაც უზრუნველყოფს მოქნილობის ამაღლებას.
- ხელმისაწვდომობა. აპლიკაციებს შეუძლიათ გამოიყენონ მოდულური არქიტექტურა, რომელიც სისტემაში ადვილად მასშტაბირებადი კომპონენტების გამოყენების შესაძლებლობას იძლევა, რაც ამაღლებს ხელმისაწვდომობას.

3-დონიანი არქიტექტურის გამოყენება სასურველია , როდესაც იქმნება აპლიკაცია ორგანიზაციის შიდა ქსელისათვის. სადაც ყველა სერვერი განლაგებული იქნება დახურულ ქსელში; ან ინტერნეტ აპლიკაცია, რომლის უსაფრთხოების მოთხოვნები არ კრძალავს ბიზნეს ლოგიკის განლაგებას web სერვეზე ან აპლიკაციის სერვერზე.

4.5. ობიექტზე-ორიენტირებული არქიტექტურა

ობიექტზე-ორიენტირებული არქიტექტურა დაფუძნებულია აპლიკაციის ან სისტემის პასუხისმგებლობის დანაწილებაზე დამოუკიდებელ, განმეორებითი გამოყენებისათვის გამოსადეგ ობიექტებად, რომელთაგან თითოეული მოიცავს მონაცემებსა და ქცევებს ამ ობიექტების შესახებ. ობიექტზე-ორიენტირებული პროექტირების სისტემა განიხილება არა როგორც ქვე პროგრამებისა და პროცედურული ბრძანებების ნაკრები , არამედ როგორც ურთიერთმოქმედი

ობიექტების ნაკრები. ობიექტები გამიჯნულია, დამოუკიდებელია და სუსტად დაკავშირებული; მონაცემთა გაცვლა მათ შორის ხდება ინტერფეისების საშუალებით სხვა ობიექტების მეთოდების და თვისებების გამოძახებისა და შეტყობინებების გაგზავნა/მიღების გზით. ობიექტზე-ორიენტირებული არქიტექტურული სტილის ძირითად პრინციპებს წარმოადგენს:

- აბსტრაქცია. შესაძლებლობას იძლევა განაზოგადოს რთული ოპერაცია , ამავე დროს შეინახოს ოპერაციის ძირითადი მახასიათებლები. მაგალითად, აბსტრაქტული ინტერფეისი შეიძლება იყოს ფართოდ ცნობილი აღწერა, მხარდაჭერილი მონაცემებთან წვდომის ოპერაციის მარტივი მეთოდების გამოყენებით, ისეთის როგორიცაა Get(მიღება) და Update (განახლება). აბსტრაქციის სხვა ფორმაა მეტამონაცემები , რომლებიც გამოიყენება სტრუქტურირებული მონაცემების ორი ფორმატის თანხვედრის უზრუნველსაყოფად.
- კომპოზიცია. ობიექტები შეიძლება წარმოიქმნას სხვა ობიექტებით და სურვილისამებრ დამალოს ეს შიდა ობიექტები სხვა კლასებისგან ან წარმოადგინოს ისინი , როგორც მარტივი ინტერფეისები.
- მემკვიდრეობითობა. ობიექტებმა შეიძლება მემკვიდეობით მიიღონ სხვა ობიექტების ელემენტები და გამოიყენოს საბაზო ობიექტის ფუნქციონალობა ან მოახდინოს ხელახალი განსაზღვრა ახალი ქცევის რეალიზაციისათვის. უფრო მეტიც მემკვიდრეობითობა ამარტივებს მომსახურებას და განახლებას, რამდენადაც საბაზო ობიექტში შეტანილი ცვლილებები ავტომატურად ვრცელდება მის მემკვიდერე ობიექტებზე.
- ინკაპსულაცია. ობიექტები წარმოადგენენ ფუნქციონალობას მხოლოდ მეთოდების, თვისებებისა და მოვლენების საშუალებით და მალავენ შიდა დეტალებს, როგორიცაა მდგომარეობა და ცვლადები, სხვა ობიექტებისაგან. ეს მარტივებს ობიექტების განახლებას ან შეცვლას და შესაძლებლობას იძლევა შესრულდეს ეს ოპერაციები სხვა ობიექტებზე და კოდზე ზეგავლენის გარეშე, საჭიროა მხოლო უზრუნველყოფილ იქნას თავსებადი ინტერფეისები.

- პოლიმორფიზმი. საშუალებას იძლევა გადაიტვითოს ბაზური ტიპის ქცევა, რომელიც მხარს უჭერს ოპერაციებს აპლიკაციებში, ახალი ტიპების რეალიზაციის გზით, რომლებიც არინ ურთიერთშემცვლელები არსებული ობიექტისათვის.
- გამოყოფა. ობიექტები შეიძლება გამოყოფილი იქნას მომხმარებლისაგან აბსტრაქტული ინტერფეისის განსაზღვრის გზით, რომელიც რეალიზებულია ობიექტით და გასაგებია მომხმარებლისთვის. ეს შესაძლებლობას იძლევა უზრუნველყოფილი იქნას ალტერნატიული რეალიზაციები ინტერფეისის მომხმარებლებზე ზეგავლენის გარეშე.

როგორც წესი ობიექტზე-ორიენტირებული სტილი გამოიყენება ობიექტური მოდელის აღწერისას , რომელიც მხარს სჭერს რთულ სამცნიერო და ფინანსურ ოპერაციებს.

ობიექტზე-ორიენტირებული არქიტექტურული სტილის ძირითად უპირატესობას წარმოადგენს:

- განმეორებითი გამოყენების შესაძლებლობა. პოლიმორფიზმისა და აბსტრაქციის საშუალებით შეიძლება განმეორებითი გამოყენების უზრუნველყოფა.
- ტესტირება. უზრუნველყოფს ტესტირების გაუმჯობესებას ინკაფსულაციის მეშვეობით.
- გაფართოებადობა. ინკაფსულაცია, პოლიმორფიზმი და აბსტრაქცია გარანტიას იძლევა, რომ ცვლილება მონაცემთა წარმოდგენაში გავლენას არ მოახდენს ინტერფეისებზე,რომლებიც ობიექტებით არიან წარმოდგენილნი.
- მაღალი გამართულობა. ობიექტში მხოლოდ ფუნციონალურად ახლო მეთოდებისა და ფუნქციების განთავსება და ფუნქციების სხვადასხვა ნაკრებისათვის სხვადასხვა ობიექტის გამოყენებით შეიძლება მიღწეულ იქნას მაღალი დონის გამართულობა.

ობიექტზე-ორიენტირებული არქიტექტურის გამოყენება სასურველია აპლიკაციის მოდელირებისას რეალური ობიექტებისა და მოვლენების ბაზაზე ან თუ უკვე

არსებობს შესაფერისი ობიექტები და კლასები, რომლებიც შეესაბამებიან პროექრისა და ექსპლუატაციის მოთხოვნებს. ობიექტზე-ორიენტირებული სტილის გამოყენება ასევე რეკომენდირებულია, როდესაც საჭიროა ლოგიკისა და მონაცემებების ინკაპსულაცია კომპონენტებში, რომლებიც გამოსადეგია განმეორებითი გამოყენებისათვის.

4.6. სერვისზე ორიენტირებული არქიტექტურა

სერვისზე ორიენტირებული არქიტექტურა (Service-oriented architecture, SOA) იძლევა შესაძლებლობას უზრუნველყოს აპლიკაციის ფუნქციონალობა სერვისების კომპლექტის სახით და შექმნას აპლიკაციები, რომლებიც იყენებენ პროგრამულ სერვისებს. სერვისები სუსტად არის ერთმანეთთან დაკავშირებული, რადგანაც იყენებენ სტანდარტებზე დაფუძნებულ ინტერფეისებს, რომელთა გამოძახება, გამოქვეყნება და აღმოჩენა შესაძლებელია. SOA-ში სერვისების ძირითადი ამოცანაა უზრუნველყოს აპლიკაციასთან ურთიერთქმედება ინტერფეისით მიღებული შეტყობინებების მეშვეობით, რომელთა მოქმედების არეალს წარმოადგენს აპლიკაცია, და არა კომპონენტი ან ობიექტი. არ უნდა იქნას განხილული SOA-სერვისი როგორც სერვისების კომპონენტური მომწოდებელი.

SOA-არქიტექტურას შეუძლია უზრუნველყოს ბიზნეს-პროცესების შეფუთვა სერვისებში, რომლებიც უზრუნველყოფენ ურთიერთქმედების შესაძლებლობას და იყენებენ ინფორმაციის გადასაცემად პროტოკოლებისა და მონაცემთა ფორმატების ფართო დიაპაზონს. კლიენტებს და სხვა სერვისებს შეუძლიათ წვდომა ლოკალურ სერვისებზე, რომლებიც შესრულებულია იმავე დონეზე, ან ქსელით მოშორებულ სერვისებზე.

SOA არქიტექტურული სტილის ძირითად პრინციპებს წარმოადგენს:

- სერვისები ავტონომიურია. თითოეული სერვისის ვერსიის მომსახურება, შემუშავება, განლაგება და კონტროლი ხორციელდება ერთმანეთისაგან დამოუკიდებლად.
- შესაძლებელია სერვისების განაწილება. სერვისები შეიძლება განლაგდეს ნებისმიერ ადგილას, ლოკალურად ან დისტანციურად, თუ ქსელს გააჩნია აუცილებელი კავშირის პროტოკოლები.
- სერვისები სუსტად არის ერთმანეთთან დაკავშირებული. თითოეული სერვისი აბსოლუტურად დამოუკიდებელია და შეიძლება შეიცვალოს ან განახლდეს იმ აპლიკაციებზე ზეგავლენის გარეშე, რომლებიც მას იყენებენ, თავსებადი ინტერფეისით უზრუნველყოფის პირობებში.
- სერვისები ერთობლივად იყენებენ სქემას და კონტრაქტს, მაგრამ არა კლასს. მონაცემთა გაცვლისას სერვისები ერთობლივად იყენებენ კონტრაქტებსა და სქემებს, მაგრამ არა შიდა კლასებს.
- თავსებადობა დაფუძნებულია პოლიტიკაზე. პოლიტიკა, მოცემულ შემთხვევაში, ნიშნავს მახასიათებლების აღწერას, როგორიცაა ტრანსპორტი, პროტოკოლი და უსაფრთხოება.

ტიპიური სერვისზე ორიენტირებული აპლიკაციები უზრუნველყოფენ ინფომაციის ერთობლივ გამოყენებას, მრავალეტაპიანი პროცესების შესრულებას (განლაგების სისტემები და ონლაინ-მაღაზიები), სპეციალური დარგობრივი მონაცემების ან ორგანიზაციებს შორის სერვისების უზრუნველყოფას და კომპოზიტური აპლიკაციების შექმნას, რომლებიც აერთიანებენ მონაცემებს მრავალი წყაროებიდან.

SOA არქიტექტურის ძირითად უპირატესობებს წარმოადგენს:

- განმეორებითი გამოყენება საერთო სერვისებისა სტანდარტული ინტერფეისებით აფართოებენ ტექნოლოგიურ და ბიზნეს-შესაძლებლობებს, აგრეთვე ამცირებენ ღირებულებას.
- აბსტრაქცია. სერვისები ავტონომიურია, მათთან წვდომა ხორციელდება ფორმალური კონტრაქტით, რაც უზრუნველყოფს სუსტ კავშირს და აბსტრაქციას.

- აღმოჩენის შესაძლებლობა. სერვისებს შეუძლიათ უზრუნველყონ აღწერა, რაც საშუალებას აძლევს სხვა აპლიკაციებს და სერვისებს აღმოაჩინონ ისინი და ავტომატურად განსაზღვრონ ინტერფეისი.
- თავსებადობის შესაძლებლობა. რამდენადაც პროტოკოლები და მონაცემთა ფორმატები დაფუძნებულია დარგობრივ სტანდარტებზე, სერვისის მიმწოდებელი და მომხმარებელი შეიძლება შეიქმნას და განლაგდეს სხვადასხვა პლატფორმაზე.
- რაციონალიზაცია. სერვისები უზრუნველყოფენ განსაზღვრულ ფუნქციონალობას, რითაც ხდება მისი დუბლირების აუცილებლობის აღმოფხვრა აპლიკაციებში.

5. არქიტექტურის შერჩევა საბანკო ტიპის პროგრამებისათვის

დღეს საქართველოში ერთ-ერთი ყველაზე მეტად რეალიზებადი საბანკო პროგრამებია. თანამედროვე ბანკის ბიზნეს-სტრატეგიები მთლიანად დაფუძნებულია ინფომაციული ტექნოლოგიების გამოყენებაზე. ბანკის ეფექტურ მუშაობაში დიდ როლს თამაშობს პროგრამული უზრუნველყოფის მოხერხებულობა, საიმედოობა და უსაფრთხოება.

თანამედროვე ბანკში პროგრამული უზრუნველყოფის როლი ძალიან დიდია . მაგრამ ამავე დროს დიდია რისკი, რომელიც დაკავშირებულია უხარისხოდ შემუშავებული პროგრამული უზრუნველყოფის დანერგვასთან. ბევრი ბანკი ზარალდება ძირითადი პროგრამული სისტემების არასტაბილური მუშაობით.

პროგრამული უზრუნველყოფის კარგი არქიტექტურისა შექმნისათვის გათვალისწინებულ უნდა იქნას პროექტირების ძირითადი პრინციპები.

5.1. კლიენტის პროფაილი(საბანკო პროგრამა)

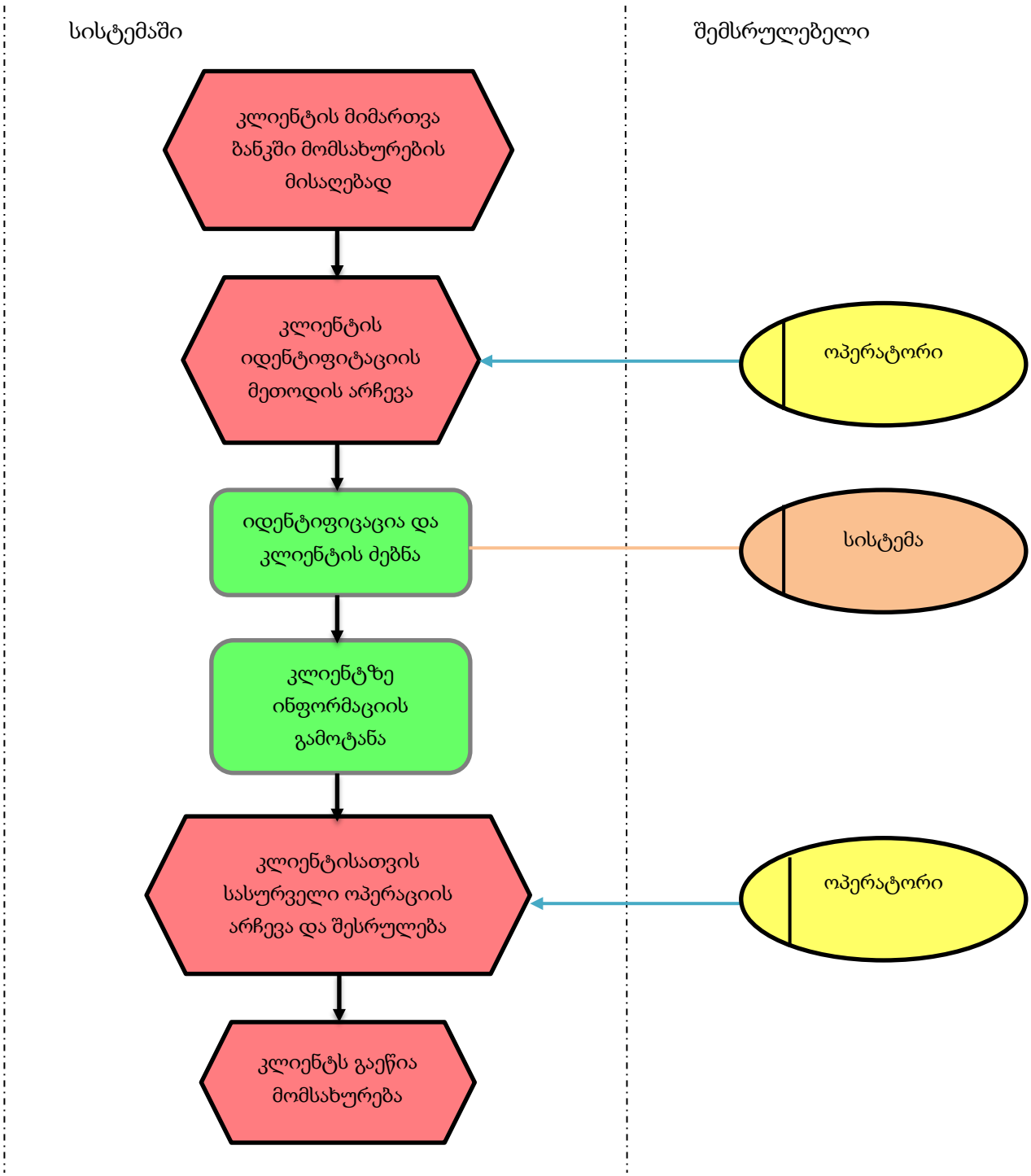
საქართველოს ბანკების უმეტესობაში კლიენტის შესახებ ინფორმაცია სხვადასხვა პროგრამებშია გადანაწილებული, მაგალითად კლიენტის პირადი ინფორმაცია და მის მიერ შესრულებული ოპერციები(გადარიცხვები, ჩარიცხვები, თანხის შეტანა , გატანა და ა.შ) ერთ პროგრამაშია თავმოყრილი, კლიენტის სესხების შესახებ ინფორმაცია - მეორეში, დანარჩენი პროდუქტების შესახებ - მესამეში. ეს ძალიან მოუხერხებელი ოპერატორისათვის ,რადგან მან სამივე პროგრამიდან უნდა მოახდინოს ინფორმაციის შეგროვება, რათა სრული სურათი შეექმნას კლიენტის შესახებ, რაც ბევრ დროს და ძალისხმევას მოითხოვს.

ჩემი მიზანია შევქმნა ისეთი ახალი სამომხმარებლო ინტერფეისი ანუ კლიენტის პროფაილი, რომელიც მნიშვნელოვნად გაამარტივებს და ააჩქარებს კლიენტის

მომსახურებას და ამავე დროს იქნება ხარისხიანი და უსაფრთხო. ბანკის თანამშრომლისთვის ხელმისაწვდომი იქნება კლიენტის შესახებ კონსოლიდირებული ინფორმაცია:

- კლიენტის პირადი ინფორმაცია;
- ბანკის იმ პროდუქტების ჩამონათვალი, რომლითაც კლიენტი სარგებლობს;
- კლიენტის ანგარიშები და ბალანსი;
- შესათავაზებელი პროდუქტების ჩამონათვალი
- პროდუქტები, რომლებიც დამზადების პროცესშია (მაგ: ბარათის დამზადება);
- კლიენტისათვის მნიშვნელოვან ინფორმაციის გამოტანა, რომელიც აუცილებლად უნდა შეატყობინოს კლიენტს ბანკის თანამშრომელმა(მაგ:სესხის დაფარვის ვადის მოახლოების შესახებ).

კლიენტის პროფაილი ბანკის თანამშრომელს საშუალებას მისცემს დაინახოს ბანკისა და კლიენტის „ურთიერთქმედების ისტორია“, დაეხმარება გაიგოს კლიენტის მოთხოვნები და მისცეს სრულყოფილი პერსონალური რჩევა კონკრეტული სიტუაციის მიხედვით, რაც ბანკის მომსახურებას გახდის ინდივიდუალურს (პერსონიფიცირებულს).



ნახ. 2

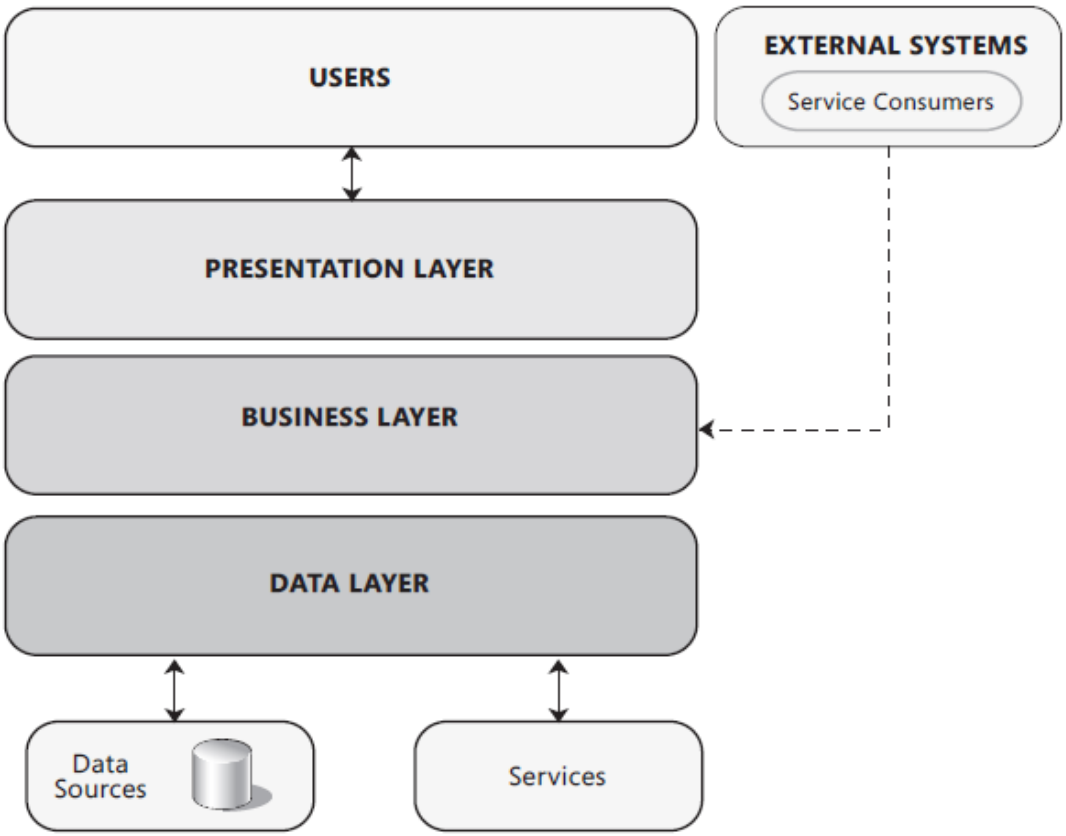
კლიენტის პროფაილის ბიზნეს პროცესის ბლოკ-სქემა

5.2. მრავალშრიანი არქიტექტურის პროექტირება

მნიშვნელოვანია გვახსოვდეს რა განსხვავებაა შრეებსა(Layers) და დონეებს(tiers) შორის. შრეები განსაზღვრავენ აპლიკაციის ფუნქციებისა და კომპონენეტების ლოგიკურ დაჯგუფებას, ხოლო დონეები განსაზღვრავენ ფუნქციებისა და კომპონენეტების ფიზიკურ განლაგებას სერვერებზე , კომპიტერებსა თუ ქსელებში. მიუხედავად იმისა, რომ შრეებისა და დონეებისთვის გამოიყენება ერთი და იგივე ტერმინოლოგია(წარმოდგენა,ბიზნესი, სერვისები და მონაცემები), საჭიროა გვახსოვდეს, რომ მხოლოდ დონებში იგულისხმება ფიზიკური დაყოფა. რამდენიმე შრის ერთ კომპიუტერზე (ერთ დონეზე) განლაგება საკმაოდ ჩვეულებრივი მოვლენაა.

5.3. წარმოდგენის , ბიზნესისა და მონაცემების შრეები

ნახ.3 ნაჩვენებია წარმოდგენის , ბიზნესისა და მონაცემების შრეები და მათი ურთიერთქმედება მომხმარებლებთან და სხვა სერვისებთან.



ნახ. 3

აპლიკაცია შეძლება შედგებოდეს მთელი რიგი შრეებისგან. სურათზე წარმოდგენილი ტიპიური სამშრიანი დიზაინი შედგება შემდეგი შრეებისაგან:

- **წარმოდგენის შრე.** მოცემული შრე შეიცავს მომხმარებელზე ორიენტირებულ ფუნქციონალურობას, რომელიც პასუხისმგებელია მომხმარებლის სისტემასთან ურიერთქმედებაზე,როგორც წესი შეიცავს კომპონენტებს, რომლებიც უზრუნველყოფენ კავშირს ძირითად ბიზნეს ლოგიკასთან. ეს შრე წარმოადგენს მომხმარებლის ინტერფეისს,რომელიც შეიცავს მონაცემების შეტანისა და გამოტანის ელემენტებს.

  მომხმარებლებთან ურთიერთქმედების კარგი სტილი და კარგი სამომხმარებლო ინტერფეისი - ეს განასხვავებს კარგ სამომხმარებლო ინტერფეისს ცუდისგან. საჭიროა მომხმარებელთა გამოკითხვა რათა დადგინდეს რა უნდათ მომხმარებლებს და რა რას ელიან იცინი ამ

აპლიკაციისგან. ამ მონაცემების შედეგად უნდა შეიქმნას სამომხმარებლო ინტერფეისი. არსებობს სამომხმარებლო ინტერფეისის პროექტირების შემდეგი რეკომენდაციები :

- ინტერფეისი არ უნდა გადატვირთული და რთული.
- შეცდომების შემთხვევაში უნდა გამოდიოდეს სასარგებლო და ინფორმატიული შეტყობინებები ამ შეცდომის შესახებ.
- მომხმარებლის მიერ შეყვანილი მონაცემების შემოწმების ეფექტური სტრატეგია ძალიან მნიშვნელოვანია აპლიკაციის უსაფრთხოებისა და კორექტული მუშაობისთვის.

- **ბიზნეს შრე.** ამ შრეში რელიზებულია სისტემის ძირითადი ფუნქციონალობა. ეს შრე პასუხისმგებელია მონაცემების მიღებაზე, დამუშავებასა და მართვაზე.
- **მონაცემებთან წვდომის შრე.** ეს შრე უზრუნველფოფს მონაცემებთან კავშირს ეს შრე წარმოადგენს უნივერსალურ ინტერფეისებს, რომლებიც შეიძლება გამოიყენოს ბიზნეს შრემ. ეს შრე უნა პასუხობდეს აპლიკაციის მოთხოვნებს, უნდა მუშაობდეს ეფექტურად და უსაფრთხოდ.

5.4. შრეებად დაყოფის სტრატეგიის შერჩევა

შრეებად დაყოფა წარმოადგენს აპლიკაციის კომპონენტების ლოგიკურ დანაწილებას ჯგუფებად, რომლებიც განსაზღვრულ როლსა და ფუნქციებს ასრულებენ. მწარმოებლურობის ამაღლების აუცილებლობის შემთხვევაში მრავალშრიანი მიდგომის გამოყენებით შესაძლებელია აპლიკაციის მომსახურების მოხერხებულობის ამაღლება და მისი მასშტაბირების გამარტივება. ურთიერთდაკავშირებული ფუნქციების შრეებად დაჯგუფების მრვალი მეთოდი არსებობს. თუმცა შრეებად არასწორმა დაყოფამ (მეტისმეტად მცირე ან მეტისმეტად ბევრი) შეიძლება მხოლოდ გაართულოს აპლიკაცია, რაც გამოიწვევს საერთო მწარმოებლურობის, მომსახურების მოხერხებულობის და მოქნილობის შემცირებას. დეტალიზაციის შესაბამისი დონის განსაზღვრა აპლიკაციის შრეებად დაყოფისას-კრიტიკულად აუცილებელი პირველი ნაბიჯია შრეებად დაყოფის სტრატეგიის განსაზღვრაში.

მრავალშრიანი მიდგომის გამოყენებამ შესაძლოა რამდენადმე გაართულოს დიზაინი და გაზარდოს შემუშავების მოსამზადებელი ეტაპის ხანგრძლივობა, მაგრამ სწორი რეალიზაციის შემთხვევაში მნიშვნელოვნად გააუმჯობესებს აპლიკაციის მომსახურებას, გაფართოებისა და მოქნილობის შესაძლებლობას. გულდასმით უნდა მოვიფიქროთ, როგორ დავყოთ აპლიკაცია შრეებად, და როგორი ურთიერთქმედება ექნებათ შრეებს ერთმანეთთან; ამით მიიღწევა მწარმოებლურობასა და მოქნილობას შორის კარგი ბალანი. როგორც წესი, მოგება მოქნილობასა და მომსახურების მოხერხებულობაში,რომელსაც უზრუნველყოფს მრავალშრიანი სქემა, მნიშვნელოვნად აჭარბებს მწარმოებლობის საეჭვო ამაღლებას, რომელსაც შეიძლება მიღწეულ იქნას მჭიდროდ დაკავშირებულ დიზაინში,შრეების გამოყენების გარეშე.

5.5. შრეებს შორის ურთიერთქმედების წესების განსაზღვრა

როდესაც საქმე ეხება შრებად დაყოფის სტრატეგიას, აუცილებელია განისაზღვროს შრეების ერთმანეთთან ურთიერთქმედების წესები. ურთიერთქმედების წესების განსაზღვრის ძირითადი მიზანია-დამოკიდებულებების მინიმიზაცია და ციკლური ბმულების გამორიცხვა. მაგალითად, თუ ორ შრეს აქვს დამოკიდებულება მესამე შრის კომპონენტებისაგან, ჩნდება ციკლური დამოკიდებულება. საერთო წესს, რომელსაც უნდა მივყვეთ ამ შემთხვევაში, წარმოადგენს შრეებს შორის მხოლოდ ცალმხრივი ურთიერთქმედების მოგვარება ერთ-ერთი ქვემოთ მოყვანილი მიდგომის გამოყენებით:

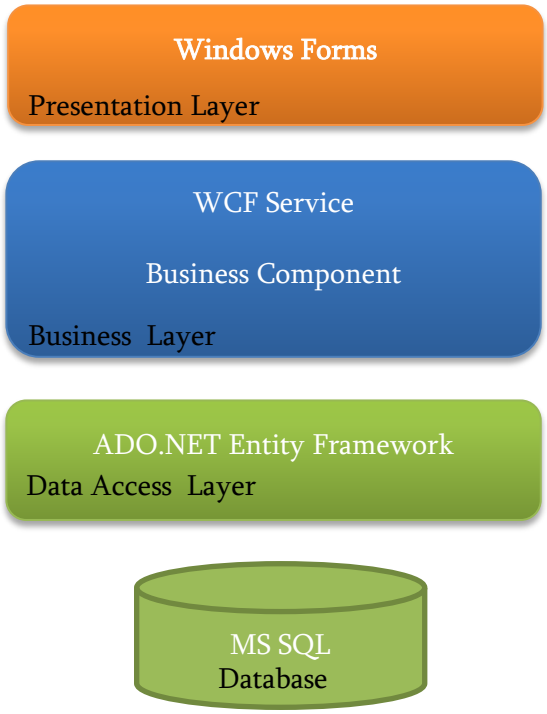
- **ურთიერთქმედება ზემოდან ქვემოთ.** შრეებს შეუძლიათ ურთიერთქმედება ქვემოთ განლაგებულ შრეებთან, მაგრამ ქვედა შრეები ვერასოდეს ვერ მოახდენენ ურთიერთქმედებას ზემოთ განლაგებულ შრეებთან. ეს წესი თავიდან აცილებს ციკლური დამოკიდებულება შრეებს შორის.
- **მკაცრი ურთიერთქმედება.** თითოეული შრე უთიერთიერთქმედებს მხოლოდ მის ქვეშ მდგომ შრესთან.ეს წესი უზრუნველყოფს მკაცრ დაყოფას , რომლიც

დროსაც თითოეულმა შრემ აქვს ინფორმაცია მხოლოდ მის ქვედა შრის შესახებ. ასეთი ურთიერთქმედების დადებითი ეფექტი ის არის, რომ რომელიმე შრეში შეტანილი ცვლილებები, მხოლოდ მის ქვედა შრეზე მოახდენს ზემოქმედებას.ასეთი მიდგომა გამოიყენება მაშინ, როდესაც მომავალში იგეგმება აპლიკაციის ახალი ფუნქციონალით გაფართოება.

- **თავისუფალი ურთიერთქმედება.** მაღალ შრეებს შეუძლიათ ურთიერთქმედება ქვედა შრეებთან პირდაპირ , სხვა შრეების შემოვლით. ეს ზრდის მწარმოებლურობას , მაგრამ ამავე დროს ზრდის დამოკიდებულებას. ასეთი მიდგომა გამოიყენება პატარა აპლიკაციების პროექტირებისას,როდესაც ბევრ შრეში ცვლილებების შეტანა არ მოითხოვს ბევრ ძალისხმევას.

6. კლიენტის პროფაილის არქიტექტურული სტილი

კლიენტის პროფაილის პროექტირებისთვის შევარჩიე მრავალშრიანი არქიტექტურის სტილი, რომელსაც აქვს შემდეგი სახე:



პირველი არის წარმოდგენის შრე , რომელშიც გამოვიყენე Windows Forms. Windows Forms წარმოადგენს .NET Framework -ის ნაწილს მისი შექმნის დღიდან და მათი გამოყენება იდეალურია ბიზნეს-აპლიკაციებისთვის. Windows Forms არის საუკეთესო გადაწყვეტა ისეთი პროგრამული უზრუნველყოფისათვის , რომელშიც მდიდარი გრაფიკული სამომხმარებლო ინტერფეისი არ არის პრიორიტეტული, არ არის წამოყენებული განსაკუთრებული მოთხოვნები მედიისა და ინტერაქტიული შესაძლებლობების მიმართ. ამიტომ Windows Forms კლიენტის პროფაილისთვის წარმოადგენს საუკეთესო გადაწყევტას.

მეორე არის ბიზნეს შრე, რომელშიც გამოვიყენე WCF Service , ამ სერვისში მოთავსებულია მთელი ბიზნეს ლოგიკა. რადგანაც საჭიროა, რომ ბიზნეს ლოგიკა იყოს უსაფთხო გარემოში, სერვისი ამ შემთხვევაში უზრუნველყოფს დაცული და უსაფრთხო პროგრამული უზრუნველყოფის შექმნას. ასევე მას შეუძლია მიმართო სხვა სერვისებს და წამოიღოს ის ინფორმაცია , რომელიც მისთვის არის საჭირო.

მესამე არის მონაცემებთან წვდომის შრე. რომელშიც გამოვიყენე ADO.NET Entity Framework. ADO.NET Entity Framework საშუალებას იძლევა მონაცემებთან აბსტრაქციის უფრო მაღალ დონეზე მუშაობის , მონაცემებზე ორიენტირებული პროგრამული უზრუნველოფის შექმნას ნაკლები კოდის გამოყენებით.

ხოლო MS SQL გამოვიყენე მხოლოდ მონაცემების შესანახად.

მრავალშრიანი არქიტექტურას უპირატესობები, რომელიც გამოვიყენე კლიენტის პროფაილის რეალიზებისთვის არის:

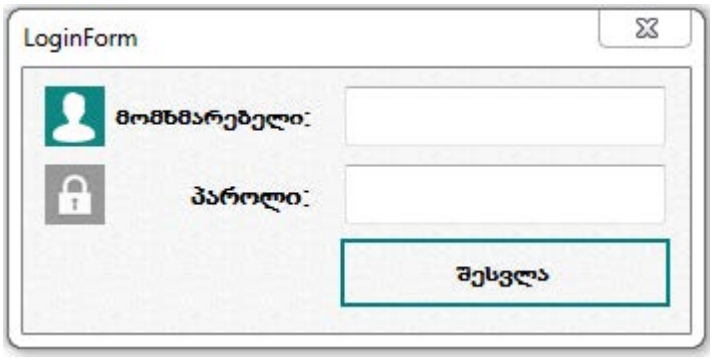
- აბსტრაქცია.ეს არქიტექტურა წარმოადგენს ერთ მთლიან სისტემას , რომელიც ამასთანავე უზრუნველყოფს საკმარის დეტალებს ცაკლეული შრეების როლისა და პასუხისმგებლობისა და მათ შორის დამოკიდებულების გასარკვევად.
- ინკაპსულაცია. პროექტირების პროცესში არ არის აუცილებელი რაიმე დაშვებების გაკეთება მონაცემთა ტიპების, მეთოდებისა და თვისებების ან რეალიზაციის შესახებ , რადგან ყველა ეს დეტალი დამალულია შრის ჩარჩოებში.
- მკაფიოდ განსაზღვრული ფუნქციონალური შრეები. ფუნქციონალობის განაწილება შრეებს შორის ძალიან მკაფიოა. ზედა შრეები , ისეთი როგორიცაა წარმოდგენის შრე , აგზავნიან ბრძანებას ქვედა შრეებში, როგორიცაა ბიზნეს შრე და მონაცემთა შრე. ზედა შრეებს შეუძლიათ ამ შრეებში წარმოქმნილ მოვლენებზე რეაგირება, ამასთან უზრუნველყოფენ მონაცემთა გადაცემის შესაძლებლობას შრეებს შორის ზემოთ და ქვემოთ.
- მაღალი გამართულობა. თითოეული შრისათვის პასუხისმგებლობის საზღვრების მკაფიოდ განსაზღვრა და შრეში მხოლოდ იმ ფუნქციონალობის

გარანტირებული ჩართვა , რომელიც პირდაპირ არის დაკავშირებული მის ამოცანებთან უზრუნველყოფს მაქსიმალურ გამართულობას შრის ჩარჩოებში.

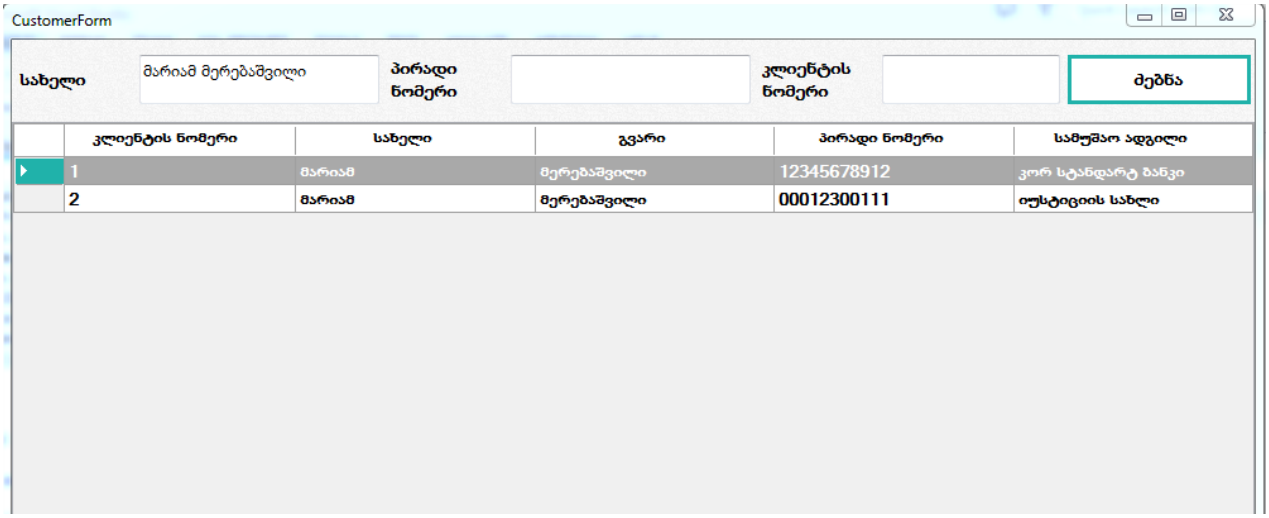
- განმეორებითი გამოყენების შესაძლებლობა. ქვედა და ზედა შრეებს შორის დამოკიდებულების არარსებობა უზრუნველყოფს მათი განმეორებითი გამოყენების შესაძლებლობას სხვა სცენარებში.
- სუსტი დამოკიდებულება. შრეებს შორის სუსტი დამოკიდებულების უზრუნველსაყოფად კავშირი მათ შორის დაფუძნებულია აბტრაქციასა და მოვლენებზე.

6.1. კლიენტის პროფაილის ინტერფეისი

სისტემაში შესვლა შესაძლებელია მომხმარებლის და პაროლის შეყვანის შემდეგ. არსებობს ორი სახის მომხმარებელი : ადმინისტრატორი და ოპერატორი ოპერატორს აქვს კლიენტისთვის სასურველი ოპერაციის ჩატარების უფლება. ადმინისტრატორი არეგისტრირებს ოპერატორებს სისტემაში.



კლიენტის ძებნა ხდება სამი ველის საშუალებით: სახელი, პირადი ნომრი და კლიენტის ნომერი. ძებნის შემდეგ გამოდის შესაბამის კლიენტზე ან კლიენტებზე ძირითადი ინფორმაცია. შესაბამისი კლიენტის სტრიქონზე ორჯერ დაწკაპუნებით გადავდივართ ძირითად ფანჯარაზე.

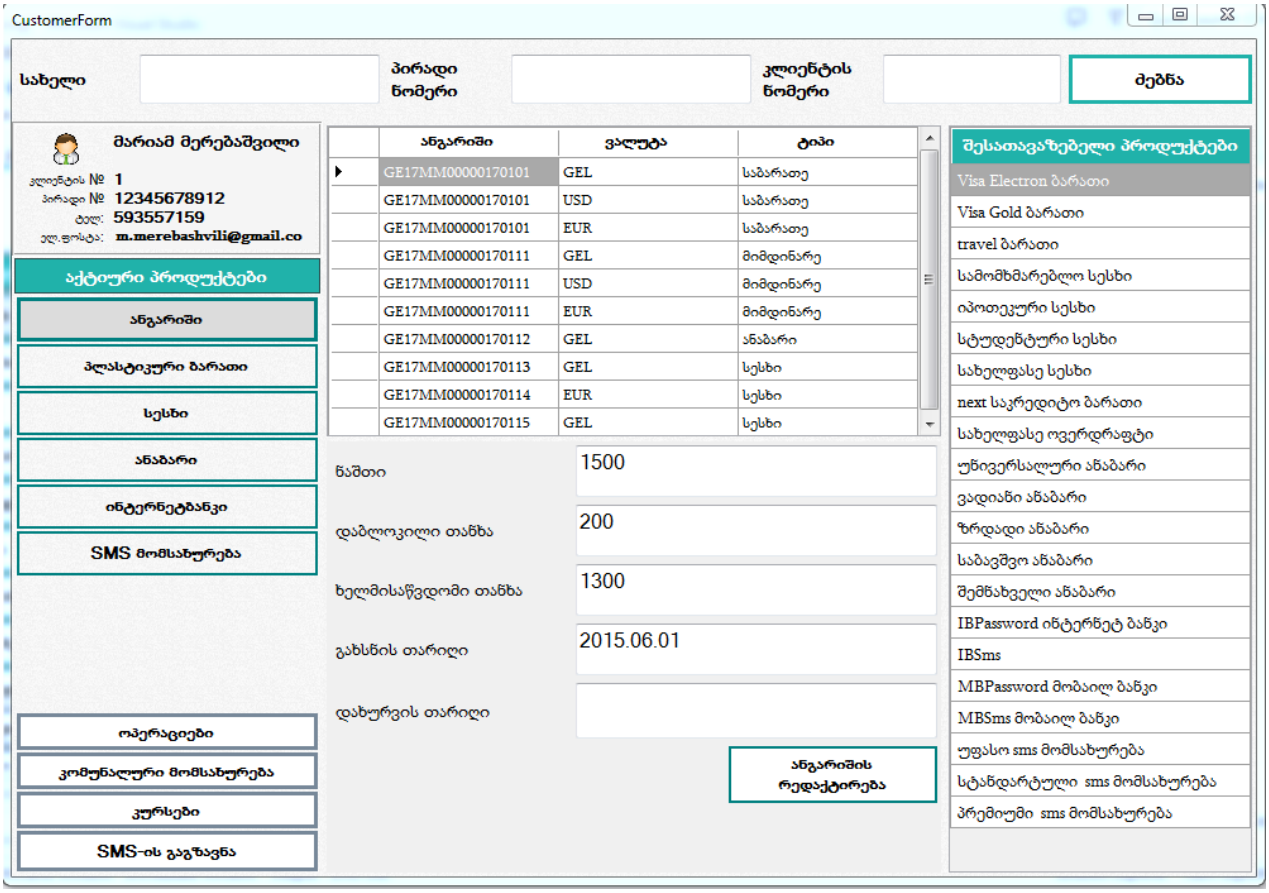


ძირითად ფანჯარაში , გამოდის კლიენტის შესახებ კონსოლიდირებული ინფორმაცია:

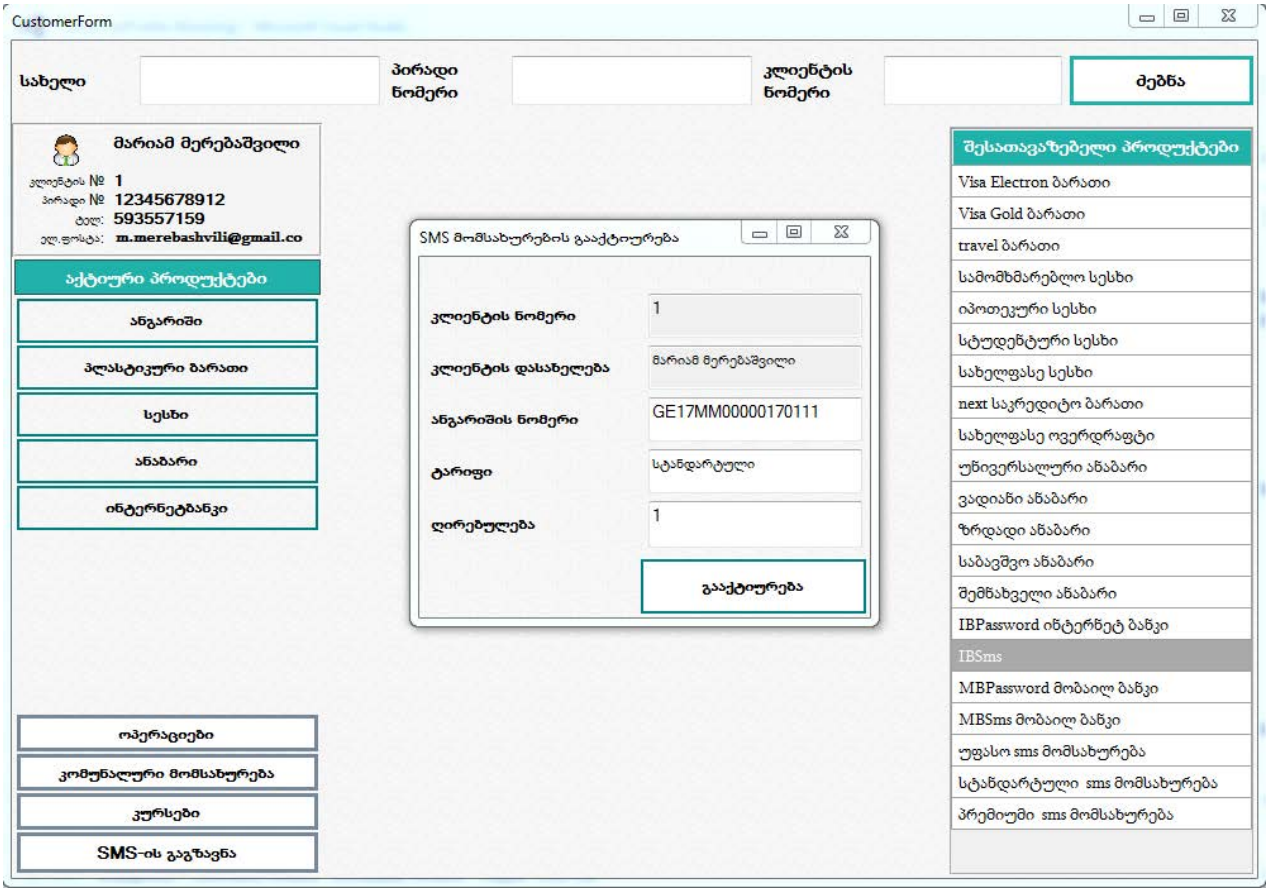
1. მარცხენა მხარეს ნაჩვენებია :
   - კლიენტის შესახებ ძირითადი ინფორმაცია;
   - რომელი პროდუქტები აქვს აქტიური;
   - ძირითადი სერვისები(ოპერაციები,კომუნალური მომსახურება და ა.შ)

   თითოეულ პროდუქტზე ერთხელ დაწკაპუნებით გამოდის დეტალური ინფორმაცია.

2. მარჯვენა მხარეს ნაჩვენებია შესათავაზებელი პროდუქტები;



შესათავაზებელ პროდუქტზე ორჯერ დაწკაპუნებით გამოდის პროდუქტის გააქტიურების ფანჯარა.



მივიღეთ ინტერფეისი, რომელიც ოპერატორისთვის წარმოადგენს მარტივ და მოხერხებულ სისტემას.

ამ მოდულის მთავარი თვისება არის კლიენტის ერთჯერადი იდენტიფიკაცია , ამის შემდეგ მომსახურების ერთ ციკლში მას შეეძლება ნებისმიერი სერვისით სარგებლობა:

- თანხის შეტანა/გამოტანა;
- კომუნალური ან სხვა მომსახურების გადახდა;
- ნებისმიერი სახის მომსახურების გააქტიურება (მაგ: sms მომსახურება, ინტერნეტ ბანკი და ა.შ)
- სესხების გაცემა ან დაფარვა;

დასკვნა

პროგრამული არქიტექტურის შემუშავებისას უნდა გვახსოვდეს პროექტირების ძირითადი პრინციპების შესახებ. ეს დაგვეხმარება ისეთი არქიტექტურის შექმნაში, რომელიც გაითვალისწინებს აპრობირებულ მიდგომებს, უზრუნველყოფს დანახარჯის მინიმიზაციას, მომსახურების სიმარტივეს, გამოყენების მოხერხებულობას და გაფართოებითობას. აპლიკაციის პროექტირებისას პროგრამული უზრუნველყოფის არქიტექტორის მთავარი მიზანი უნდა იყოს, დასმული ამოცანის მიხედვით განსაზღვროს აპლიკაციის ტიპი. იქნება ეს მობილური აპლიკაცია, სერვისი, web-აპლიკაცია, desktop-აპლიკაცია თუ სხვა რამე. განსაზღვროს რომელ არქიტექტურულ სტილს გამოიყენებს და ბოლოს ,შერჩეული აპლიკაციის ტიპისა და არქიტექტურული სტილის საფუძველზე განსაზღვროს რა ტექნლოგიებს გამოიყენებს.

გამოყენებული ლიტერატურა

- Microsoft Patterns , Practices Team, Microsoft patterns & practices Application Architecture Guide 2.0 ;
- http://www.microsoft.com/architectureguide;
- Alex Berson , McGraw-Hill CLIENT/SERVER ARCHITECTURE. 2nd edition;
- Hector Garcia-Molina , Jeffrey D. Ullman , Jennifer Widom, Database Systems: The Complete Book (2nd Edition) ;
- Leon Shklar ,Rich Rosen, Web Application Architecture: Principles, Protocols and Practices ;
- Len Bass, Paul C. Clements, Rick Kazman Software Architecture in Practice (3rd Edition);
- Nishith Pathak ,Pro WCF 4: Practical Microsoft SOA Implementation;
- Dino Esposito , Andrea Saltarello, Microsoft .NET - Architecting Applications for the Enterprise (2nd Edition);
- http://www.infoq.com/articles/ddd-in-practice;
- http://techie-tel.blogspot.com/2011/08/using-patterns-to-build-robust-3-tier.html;