ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო
უნივერსიტეტი

დავით ბერაძე

ანგარიშის ავტომატური შევსების სისტემა

ინფორმაციული სისტემები

სამაგისტრო ნაშრომი შესრულებულია ინფორმაციულ სისტემებში
მეცნიერების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: ტარიელ ხვედელიძე
ფიზ.-მათ. მეცნიერებათა კანდიდატი
ასოცირებული პროფესორი

თბილისი
2015

ანოტაცია

დღევანდელ ადამიანს ყოველდღიურად უწევს მოახდინოს საკუთარი ფინანსების განკარგვა, გადაიხადოს გადასახადები, შეავსოს მობილური ტელეფონის ანგარიში, იყიდოს ნივთები ინტერნეტით, გადაიხადოს ინტერნეტის საფასური და ა.შ. ამის გამო წამყვანი კომპანიები აქტიურად ცდილობენ ყველა ეს მოქმედება გახადონ ავტომატური და დააზოგვინონ კლიენტებს დრო, ხარჯები, გადახდის გადაცილებისაგან გამოწვეული პრობლემები. ასეთი კომპანიები მოღვაწეობენ ისეთ სფეროებში როგორებიცაა მობილური კავშირგაბმულობის კომპანიები, ინტერნეტ პროვაიდერები, საკაბელო და სატელიტური ტელევიზიები, ინტერნეტ მაღაზიები, მიკრო ფინანსები, VoIP (Voice Over IP) კავშირგაბმულობის ოპერატორები, ბანკები, ინტერნეტ აუქციონები და ა.შ.

ყველა ის კომპანია სადაც ხდება მომხმარებლების ანგარიშების აკუმულირება, ფინანსური აღრიცხვა, მონიტორინგი, ხარჯვა, შევსება იყენებს ეგრეთწოდებულ საბილინგო სისტემებს. ეს სისტემები გამოიყენება სწორედ ფინანსური ტრანზაქციებისთვის.

საბილინგო სისტემები ხშირ შემთხვევაში არიან დიდი სისტემები რომელთა განვითარება, მოდიფიკაცია, განახლება არის ისეთ პრობლემებთან დაკავშირებული როგორებიცაა დიდი ხარჯები, დრო რაც საჭიროა მის განსაახლებლად, დიდი მოცულობის ინფორმაცია რომლის მიგრაციაც არის საჭირო განახლებისას. ამის გამო კომპანიები ცდილობენ მოახდინონ ეგრეთწოდებული ინტეგრირებული სისტემების შექმნა რომლებიც დამოუკიდებლად მოახდენენ ფუნქციონირებას. ასეთი სისტემები როგორც წესი თავისთავზე იღებენ გარე სამყაროსთან კომუნიკაციის ფუნქციას.

წინამდებარე ნაშრომი საბილინგო სისტემის ერთ-ერთი ინტეგრირებული ტიპის სისტემას ეფუძნება, რომელიც ახდენს ანგარიშების მონიტორინგს და საჭიროების შემთხვევაში მის ავტომატურ შევსებას. მისი გამოყენებით მომხმარებლებს აღარ დასჭირდებათ ყოველდღიურად შევიდნენ ინტერნეტში, შეავსონ თანხა ხელით ან გაიარონ დიდი მანძილი ანგარიშის შევსების ტერმინალის მოსაძებნად.

„ანგარიშის შევსების ავტომატური სისტემა“ არის მზა პროდუქტი და შესაძლებელია გამოყენებული იქნას ყველა ისეთ კომპანიაში სადაც ხდება საბილინგო სისტემებით ანგარიშების მართვა, მაგალითად: მობილური კავშირგაბმულობის კომპანიები, ინტერნეტ პროვაიდერები, საკაბელო და სატელიტური ტელევიზიები, ინტერნეტ მაღაზიები, მიკრო

ფინანსები, VoIP (Voice Over IP) კავშირგაბმულობის ოპერატორები, ბანკები, ინტერნეტ აუქციონები და ა.შ.

Annotation

Modern people on a daily basis have to manage their own finances, pay different bills, set amount down on mobile phone's balance, buy things via internet, pay for the internet usage, etc. That's why all the leading companies are trying to make all these actions automatic because the target is to avoid subscribers wasting time, extra charge and different corresponding troubles and problems. The sphere of activity of mentioned companies is quite wide: mobile communication companies, Internet providers, cable and satellite TVs, internet-shops, banks and micro-finance companies, VoIP (Voice Over IP) communication operators, online auctions, etc.

All those companies which practice area is accumulation subscribers' accounts, financial accounting, monitoring, charging and filling balance use Billing Systems. These systems are used exactly for different financial transactions.

In most cases the Billing Systems are large systems and their development, modification and upgrade is linked to a huge expenses, amount of time that is required for its upgrading, a huge amount of information that will be migrated while upgrade, etc. Therefore these companies are trying to create kind of the Integrated Systems which will operate independently. As a rule such systems communicate with the outer space by themselves.

The thesis below is based on one of the system of the integrated type of the Billing system that monitors subscribers' accounts (balance) and automatic refilling in case of need. Using the system avoid the users from connect the internet on a daily basis, to fill balance manually or walk a long way to find any balance-filling terminal.

'Direct Debit Management System' is already finished product and can be used for all those companies which use the billing systems in Account Management. For example: Mobile communication companies, Internet providers, Cable and Satellite TVs, Internet shops, Micro-

finance companies, VoIP (Voice Over IP) communication operators, Banks, Internet Auctions, etc.

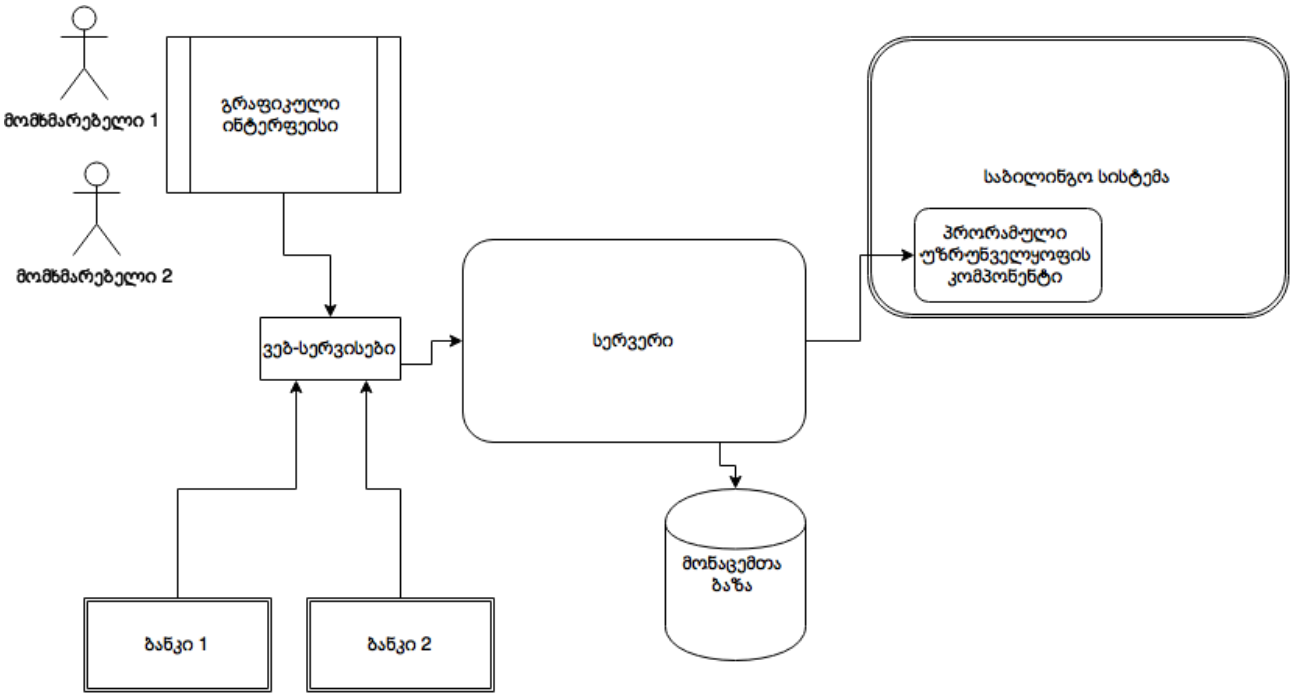
შინაარსი

შესავალი

თემის აქტუალურობა: დღევანდელი ადამიანისთვის დრო წარმოადგენს სიცოცხლის ერთ-ერთ ყველაზე მნიშნველოვან კომპონენტს. იმის გამო რომ ადამიანმა დაზოგოს დრო ყველა წამყვანი კომპანია თუ კორპორაცია ცდილობს არ დაიშუროს ფინანსები და რესურსები. ამისათვის იქმნება სისტემები რომლებიც მომხმარებლების მაგივრად ასრულებენ ამა თუ იმ ქმედებას.

ანგარიშის შევსების ავტომატურ სისტემის მოხმარებისას მომხმარებლები ზოგავენ დროს რაც უნდა დახარჯონ ანგარიშის შესავსებად.

„ანგარიშის შევსების ავტომატური სისტემა“ შედგება კომპონენტებისგან:



1) ბაზისგან - სადაც ინახება მომხმარებლების საბილინგო ანგარიშები, ლიმიტები, ანგარიშის შევსების გრაფიკი და ა.შ.
2) კონფიგურაციის ნაწილისგან - რომლის საშუალებით ხდება სხვადასხვა კომპონენტების ჩართვა, გამორთვა, პარამეტრების ცვლილება და ა.შ.
3) გრაფიკულ ინტერფეისისგან - სადაც ხდება მომხმარებლების დარეგისტრირება, წაშლა, სისტემის მონიტორინგი, შევსებების ისტორიის ნახვა და ა.შ.
4) სერვერი - რომელიც ყველა კომპონენტს აკავშირებს ერთმანეთთან.
5) მონიტორინგის ნაწილი - პროგრამული ნაწილი რომელიც ახდენს საბილინგო სისტემაში ანგარიშების მონიტორინგს.

6) ვებ-სერვისების ნაწილი - რომელიც უზრუნველყოფს საბანკო სისტემასთან კომუნიკაციას ვებ-სერვისების გამოყენებით და საჭიროების შემთხვევაში თანხის შევსების მოთხოვნის გაგზავნას.

სისტემაში არსებული ანგარიშები:

1) საბილინგო ანგარიში - ანგარიში რომელიც კომპანიის სპეციფიკიდან გამომდინარე მინიჭებული აქვს მომხმარებელს. ის თავისთავად მოიცავს თანხას რომელიც ეკუთვნის მომხმარებელს.
2) „ანგარიშის შევსების ავტომატური სისტემის“ ანგარიში - ის განსაზღვრავს სისტემაში მომხმარებლის რეგისტრაციის ინფორმაციას.
3) საბანკო ანგარიში - მომხმარებლის ბანკში არსებული თანხობრივი ან საბარათე (Visa, MasterCard) ანგარიში.

„ანგარიშის შევსების ავტომატური სისტემის“ მუშაობის ზოგადი პრინციპი შემდეგია;

1) მომხმარებელი რეგისტრირდება სისტემაში გრაფიკული ინტერფეისის გამოყენებით და უთითებს:
   სახელს, გვარს, პირად ნომერს, ელექტრონულ ფოსტას, ბანკს სადაც აქვს საფინანსო ანგარიში, საბილინგო ანგარიშებს რომლის მონიტორინგიც უნდა განახორციელოს სისტემამ, ლიმიტებს, ანგარიშის შევსების პრინციპს.
2) მომხმარებელი ააქტიურებს თავის ანგარიშს.
3) სისტემა ამოწმებს ყველა შევსებული პარამეტრის სისწორეს.
4) სისტემა ამოწმებს პარამეტრებს საბილინგო სისტემაში.
5) სისტემა არეგისტრირებს მომხმარებლის ანგარიშს ბაზაში და ანიჭებს მას უნიკალურ იდენტიფიკატორს.
6) სისტემას მომხმარებელი გადაყავს არჩეული ბანკის ელექტრონული სისტემის ვებ-გვერდზე სადაც ხდება მომხმარებლის საბანკო რეკვიზიტების შევსება (საბანკო ანგარიშის ან საბანკო ბარათის)

7) ბანკი ვებ-სერვისების გამოყენებით „ანგარიშის შევსების ავტომატურ სისტემას“ ატყობინებს რომ მომხმარებლის საბანკო ანგარიში დარეგისტრირებულია და სისტემას შეუძლია დაიწყოს მომხმარებლის ანგარიშების მონიტორინგი.
8) სისტემა ფონურ რეჟიმში ახდენს მომხმარებლის ყველა ანგარიშის ბალანსის გადამოწმებას საბილინგო სისტემაში.
9) თუ დაფიქსირდა რომ საბილინგო ანგარიშზე მომხმარებელს გაუთავდა თანხა ან მისი ბალანსი მითითებულ ლიმიტს ჩამოცდა სისტემა შექმნის ანგარიშის შევსების მოთხოვნას და გადასცემს ბანკს ვებ-სერვისების გამოყენებით.
10) სისტემა გაუგზავნის მომხმარებელს მოკლე ტექსტურ შეტყობინებას მითითებულ GSM ნომერზე და შეატყობინებს რომ მისი საბილინგო ანგარიში საჭიროებს თანხის შევსებას.
11) ბანკი შეამოწმებს აქვს თუ არა საკმარისი თანხა საბანკო ანგარიშზე.
12) თუ აღმოჩნდა რომ საბანკო ანგარიშზე არ არის საკმარისი თანხა ბანკი ვებ-სერვისის გამოყენებით შეატყობინებს „ანგარიშის შევსების ავტომატურ სისტემას“ წარუმატებელ ტრანზაქციის შესახებ.

    „ანგარიშის შევსების ავტომატური სისტემა“ შეატყობინებს მომხმარებელს რომ მას გაუთავდა თანხა საბანკო ანგარიშზე.
13) თუ აღმოჩნდა რომ მომხმარებელს საკმარისი თანხა გააჩნია საბანკო ანგარიშზე, ბანკი ჩამოაჭრის შესაბამის თანხას საბანკო ანგარიშიდან და გადარიცხავს მომხარებლის საბილინგო ანგარიშზე.

    „ანგარიშის შევსების ავტოამტური სისტემა“ მომხმარებელს მოკლე ტექსტური შეტყობინებით შეატყობინებს რომ მისი ანგარიში შევსებულია შესაბამისი თანხით.
14) სისტემა კვლავ გააგრძელებს მომხმარებლის საბილინგო ანგარიშების მონიტორინგს.

მომხმარებელს ნებისმიერ დროს შეუძლია შეაჩეროს „ანგარიშის ავტომატური შევსების სისტემაში“ მისი ანგარიშების მონიტორინგი ან სულაც გააუქმოს მისი ანგარიში. ასევე გადაამოწმოს ანგარიშის შევსების დეტალური ისტორია.

სამაგისტრო ნაშრომი შეიცავს შემდეგ ნაწილებს:

- შესავალი;
- ორი თავი;
- დასკვნა;
- დანართი.

პირველ თავში აღწერილია ანგარიშის შევსების ავტომატური სისტემის ყველა ის კომპონენტი რომლისგანაც შედგება ან ინტეგრირებულია მასთან.

მეორე თავში სისტემის პროგრამული რეალიზაციაა აღწერილი.

დასკვნაში აღწერილია თეორიული და პრაქტიკული შედეგები.

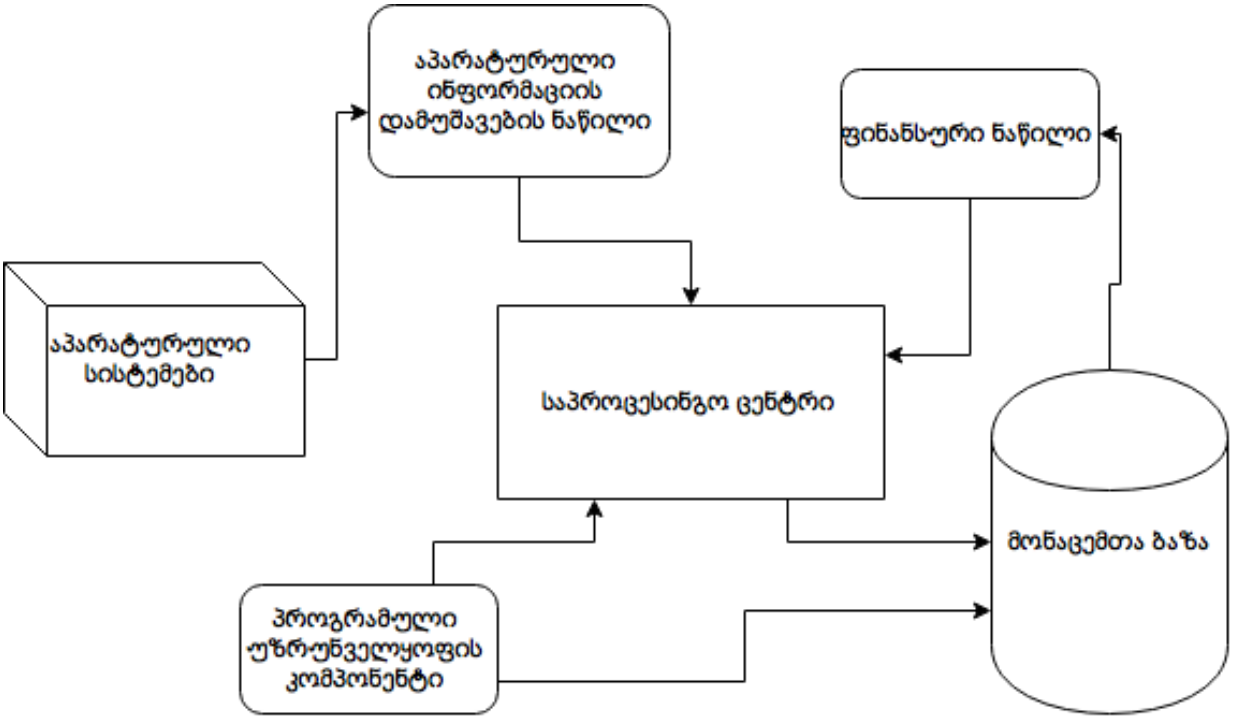
დანართში მოცემულია პროგრამული რეალიზაცია: ანგარიშის შევსების ავტომატური სისტემა.

თავი I.

სისტემის მუშაობის აღწერა

1.1 საბილინგო სისტემა

საბილინგო სისტემა წარმოადგენს პროგრამული და აპარატურული სისტემების ერთობლიობას. მისი საშუალებით ხდება მომხმარებლებზე ინფორმაციის შეგროვება, დამუშავება, ანალიზი, მონიტორინგი. საბილინგო სისტემა შედგება პროგრამული, აპარატურული ნაწილისგან, ასევე ინფორმაციული ბაზისგან სადაც ხდება ინფორმაციის შეგროვება. პროგრამული ნაწილი ახდენს აპარატურიდან ინფორმაციის წაკითხვას, ანალიზს და ახდენს მის დამუშავებას. ფინანსური ნაწილი უზრუნველყოფს ანგარიშებიდან თანხების ჩამოჭრას, შევსებას, ახდენს ინვოისების გამოწერას და ა.შ. საბილინგო სისტემების მნიშვნელოვანი ნაწილია აგრეთვე უსაფრთხოების უზრუნველყოფის და დემოგრაფიული მონაცემების აღრიცხვის სისტემები. მისი შემადგენელი კომპონენტებია:

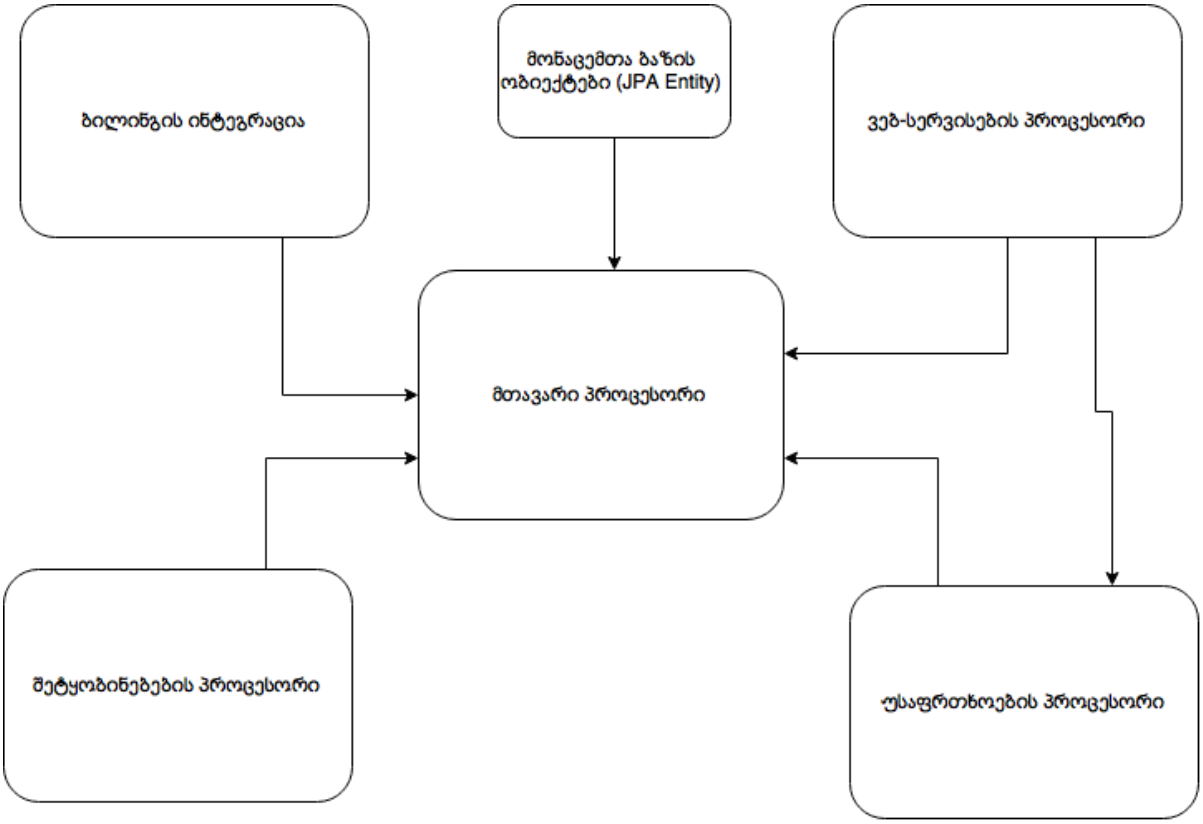


1) საპროცესინგო ცენტრი - მთავარი კომპონენტი რომელიც ახდენს დანარჩენ კომპონენტებს შორის კავშირის დამყარებას. მისი საშუალებით ხდება კომპონენტებს შორის ინფორმაციის გაცვლა.

2) ფინანსური ნაწილი - კომპონენტი რომლის საშუალებით ხდება მომხმარებლების ანგარიშებზე თანხების კონტროლი, ხარჯების დამუშავება, ფინანსური ანალიზი.
3) მონაცემთა ბაზა - მასში ინახება ყველა ის ინფორმაცია რასაც საბილინგო სისტემა ამუშავებს. ასევე მომხმარებლების ანგარიშები.
4) აპარატურული ინფორმაციის დამუშავების კომპონენტი - საბილინგო სისტემის ნაწილი რომელიც აანალიზებს აპარატურიდან მიღებულ ინფორმაციას და გადასცემს საპროცესინგო ცენტრს.
5) პროგრამული უზრუნველყოფის ნაწილი - მისი საშუალებით ხდება სხვადასხვა გარე სისტემების ინტეგრაცია.

1.2 „ანგარიშის შევსების ავტომატური სისტემის“ სერვერი

სერვერი უზრუნველყოფს სისტემის ყველა კომპონენტის კავშირს. მისი მთავარი ფუნქციაა მოახდინოს მონიტორინგი, მონაცემების დამუშავება, შენახვა. სერვერის შემადგენელი კომპონენტებია:



- მთავარი პროცესორი - მისი დანიშნულებაა მოახდინოს მონაცემების აღება ვებ-სერვისებიდან და შეინახოს მონაცემთა ბაზაში, მონიტორინგი მოახდინოს მომხმარებლების ანგარიშების, გამოთვალოს შესავსები თანხა და შექმნას მოთხოვნა რომელსაც ვებ-სერვისის პროცესორს გადასცემს.

- უსაფრთხოების პროცესორი - მისი საშუალებით ხდება ვებ-სერვისის პარამეტრების შემოწმება, ასევე პარამეტრების დაშიფვრა.
- ვებ-სერვისების პროცესორი - მისი დანიშნულებაა მოახდინოს ბანკებთან, გარე კლიენტებთან დაკავშირება SOAP ვებ-სერვისების გამოყენებით.
- ბილინგის ინტეგრაცია - მისი დანიშნულებაა მოახდინოს საბილინგო სისტემასთან კავშირი
- მონაცემთა ბაზის ობიექტები - მონაცემთა ბაზის ცხრილების ობიექტები. JPA ფრეიმვორკის გამოყენებით ახდენს ცხრილების ყველა ველების შევსებას, განახლებას, შენახვას, ამოღებას.
- შეტყობინებების პროცესორი - მისი საშუალებით ხდება მოკლე ტექსტური შეტყობინებების გაგზავნა.

  გამოყენებული ტექნოლოგიები:
- Java - ობიექტზე ორიენტირებული პროგრამირების ენა რომელიც სხვადასხვა პლათფორმაზე მუშაობს Java Virtual Machine (JVM) ის გამოყენებით. JVM უზრუნველყოფს მეხსიერების ავტომატურ მართვას.
- JBOSS application server - აპლიკაციის სერვერული ფრეიმვორკი რომლის საშუალებით ხდება აპლიკაციების მუშაობა JAVA ვირტუალურ მანქანაზე. დაწერილია JAVA ენაში და ვრცელდება როგორც "ღია კოდის" პრინციპით.
- Java Persistence API (JPA) – JAVA აპლიკაციის პროგრამული ინტერფეისი რომელიც აღწერს მონაცემთა ბაზის რელაციურ კავშირს. მისი საშუალებით შესაძლებელია მონაცემთა ბაზაში ინფორმაციის ჩაწერა, მოძებნა, განახლება, წაშლა. დაწერილია JAVA ენაში და ვრცელდება როგორც "ღია კოდის" პრინციპით.
- Enterprise JavaBeans (EJB) - სერვერის-მხარის პროგრამული კომპონენტი რომელიც უზრუნველყოფს აპლიკაციის ბიზნეს ლოგიკის შესრულებას. EJB არის JAVA EE სპეციფიკის ნაწილი რომლის საშუალებით ხდება ტრანზაქციების შესრულება, JAVA servlet ების მართვა, ვებ-სერვისების მართვა. დაწერილია JAVA ენაში და ვრცელდება როგორც "ღია კოდის" პრინციპით.
- Simple Object Access protocol (SOAP) - მონაცემთა გაცვლის პროტოკოლი რომელის საშუალებით ხდება ვებ-სერვის კომუნიკაცია. იყენებს HTTP პროტოკოლს.
- Java Servlet – Java პროგრამულ ენაში შესრულებული სერვერული ტექნოლოგია რომელსაც შეუძლია მიიღოს და დაამუშაოს HTTP მოთხოვნები.

1.3 „ანგარიშის შევსების ავტომატური სისტემის" მონაცემთა ბაზა

მონაცემთა ბაზა რომელიც ინახავს მომხმარებლების ანგარიშებს, ლიმიტებს, შესრულების გრაფიკს, გადახდების ისტორიებს. მონაცემთა ბაზა მაგალითისთვის შესრულებულია Oracle მონაცემთა ბაზის გამოყენებით. მონაცემთა ცხრილები:

DD_ACCOUNT

Column	Type	Nullable	Default	Comments
ID	**NUMBER(19)**			
ACCOUNT_ID	NUMBER(19)	Y		
ACTIVE	NUMBER(1)	Y		
CMT	VARCHAR2(4000 CHAR)	Y		
DSC	VARCHAR2(4000 CHAR)	Y		
EMAIL	VARCHAR2(255 CHAR)	Y		
FD	TIMESTAMP(6)	Y		
FIRST_NAME	VARCHAR2(255 CHAR)	Y		
LAST_NAME	VARCHAR2(255 CHAR)	Y		
PERSONAL_ID	VARCHAR2(255 CHAR)	Y		
TD	TIMESTAMP(6)	Y		
REQUESTOR_VENDOR_ID	NUMBER(19)	Y		
VENDOR_ID	NUMBER(19)	Y		
USER_ID	NUMBER(19)	Y		
USERNAME	VARCHAR2(255 CHAR)	Y		

Key	Column(s)	Type
SYS_C0054605	ID	P
FKB6DE454E9B41F612	VENDOR_ID	R
FKB6DE454EBA05D2C5	REQUESTOR_VENDOR_ID	R

Index	Column(s)	Type
SYS_C0054605	ID	unique

DD_SUBSCRIBER

Column	Type	Nullable	Default	Comments
ID	**NUMBER(19)**			
ACCOUNT_ID	NUMBER(19)	Y		
CMT	VARCHAR2(4000 CHAR)	Y		
DEBIT_TYPE	NUMBER(10)	Y		
DSC	VARCHAR2(4000 CHAR)	Y		
FD	TIMESTAMP(6)	Y		
GSM	VARCHAR2(255 CHAR)	Y		
MINIMAL_BALANCE	NUMBER(10)	Y		
REQUEST_ID	NUMBER(19)	Y		
RESULT	VARCHAR2(255 CHAR)	Y		
STATUS	NUMBER(10)	Y		
TARGET_BALANCE	NUMBER(10)	Y		
TD	TIMESTAMP(6)	Y		
IS_MASTER	NUMBER(1)	Y		

Key	Column(s)	Type
SYS_C0054609	ID	P

Index	Column(s)	Type
SYS_C0054609	ID	unique

- DD_ACCOUNT - მომხმარებლის ანგარიში რომელიც ინახავს: დემოგრაფიულ მონაცემებს, რომელი ბანკის ანგარიშით დარეგისტრირდა სისტემაში, რომელმა ოპერატორმა დაარეგისტრირა და ა.შ.
- DD_SUBSCRIBER - მომხმარებლის ნომრები რომელიც ინახავს: სტატუსებს, შევსების ლიმიტებს და ა.შ.

DD_VENDOR

Column	Type	Nullable	Default	Comn
ID	**NUMBER(19)**			
ACTIVE	NUMBER(1)	Y		
VENDOR_IDENT	VARCHAR2(255 CHAR)	Y		
IS_BANK	NUMBER(1)	Y		
VENDOR_NAME	VARCHAR2(255 CHAR)	Y		
SECURITY_LEVEL	NUMBER(10)	Y		
SIGNATURE_PASSWORD	VARCHAR2(255 CHAR)	Y		
WS_NAMESPACE	VARCHAR2(255 CHAR)	Y		
WS_SERVICENAME	VARCHAR2(255 CHAR)	Y		
WS_URL	VARCHAR2(255 CHAR)	Y		
REMOTE_IDENT	VARCHAR2(255 CHAR)	Y		
REMOTE_SIGNATURE_PASSWORD	VARCHAR2(255 CHAR)	Y		
WS_ENDPOINT_URL	VARCHAR2(255 CHAR)	Y		

Key	Column(s)	Type
SYS_C0050701	ID	P

Index	Column(s)	Type
SYS_C0050701	ID	unique

DD_PERMISSION

Column	Type	Nullable	Default	Comments
ID	**NUMBER(19)**			
ALLOWED_PARAMS	VARCHAR2(255 CHAR)	Y		
METHOD	VARCHAR2(255 CHAR)	Y		
VENDOR_ID	NUMBER(19)	Y		

Key	Column(s)	Type
SYS_C0054615	ID	P
FKA4EBAEEE9B41F612	VENDOR_ID	R

Index	Column(s)	Type
SYS_C0054615	ID	unique

- DD_VENDOR - სისტემაში ჩართული ყველა კომპონენტის ანგარიში რომელიც მოიცავს: ვებ-სერვისის პარამეტრებს, პაროლებს და ა.შ.
- DD_PERMISSION - სისტემაში ჩართული კომპონენტების სისტემის ფუნქციებზე დაშვების ცხრილი.

DD_REQUEST

Column	Type	Nullable	Default	Comments
ID	**NUMBER(19)**			
ACCOUNT_ID	NUMBER(19)	Y		
BALANCE	NUMBER(10)	Y		
CMT	VARCHAR2(4000 CHAR)	Y		
CONFIRM	VARCHAR2(255 CHAR)	Y		
FD	TIMESTAMP(6)	Y		
GSM	VARCHAR2(255 CHAR)	Y		
REQUEST_AMOUNT	NUMBER(10)	Y		
REQUEST_ID	NUMBER(19)	Y		
REQUEST_TYPE	VARCHAR2(255 CHAR)	Y		
REQUESTOR	VARCHAR2(4000 CHAR)	Y		
RESULT_CODE	NUMBER(19)	Y		
RETRY_COUNT	NUMBER(10)	Y		
TD	TIMESTAMP(6)	Y		
VENDOR_ID	NUMBER(19)	Y		
SUBSCRIBER_ID	NUMBER(19)	Y		
TARGET_BALANCE	NUMBER(10)	Y		

Key	Column(s)	Type		
SYS_C0054607	ID	P		
FK3E59ADB043A6B612	SUBSCRIBER_ID	R		
FK3E59ADB09B41F612	VENDOR_ID	R		

Index	Column(s)	Type		
SYS_C0054607	ID	unique		

DD_EPAYMENT

Column	Type	Nullable	Default	Comments
ID	**NUMBER(19)**			
ACCOUNT_ID	NUMBER(19)	Y		
FD	TIMESTAMP(6)	Y		
RESULT	NUMBER(10)	Y		
TD	TIMESTAMP(6)	Y		
TANSACTION_ID	VARCHAR2(255 CHAR)	Y		
URL	VARCHAR2(4000 CHAR)	Y		
BANK_TRANSACTION_ID	VARCHAR2(255 CHAR)	Y		

Key	Column(s)	Type		
SYS_C0054649	ID	P		

Index	Column(s)	Type		
SYS_C0054649	ID	unique		

- DD_REQUEST - სისტემის მიერ ყველა დასამუშავებელი მოთხოვნის ცხრილი რომელიც მოიცავს: მოთხოვნის ტიპს, სტატუს, მოთხოვნის პარამეტრებს და ა.შ.
- DD_EPAYMENT - საბანკო ბარათის მიბმის დროს განხორციელებული ყველა ტრანზაქციის ისტორია

DD_SMS

Column	Type	Nullable	Default	Comments
ID	**NUMBER(19)**			
LANGUAGE	NUMBER(10)	Y		
TEXT	VARCHAR2(255 CHAR)	Y		
TEXTID	NUMBER(10)	Y		
TEXT_2	VARCHAR2(255 CHAR)	Y		
TEXT_3	VARCHAR2(255 CHAR)	Y		

Key	Column(s)	Type		
SYS_C0050848	ID	P		

Index	Column(s)	Type		
SYS_C0050848	ID	unique		

DD_WEBSERVICE_LOG

Column	Type	Nullable	Default	Comments
ID	**NUMBER(19)**			
EXCEPTION	VARCHAR2(255 CHAR)	Y		
METHOD	VARCHAR2(255 CHAR)	Y		
PARAMS	VARCHAR2(4000)	Y		
RESPONSE	NUMBER(10)	Y		
RESPONSE_TEXT	VARCHAR2(4000)	Y		
STORE_DATE	TIMESTAMP(6)	Y		
REMOTE_IP	VARCHAR2(255 CHAR)	Y		
IS_INCOME	NUMBER(1)	Y		

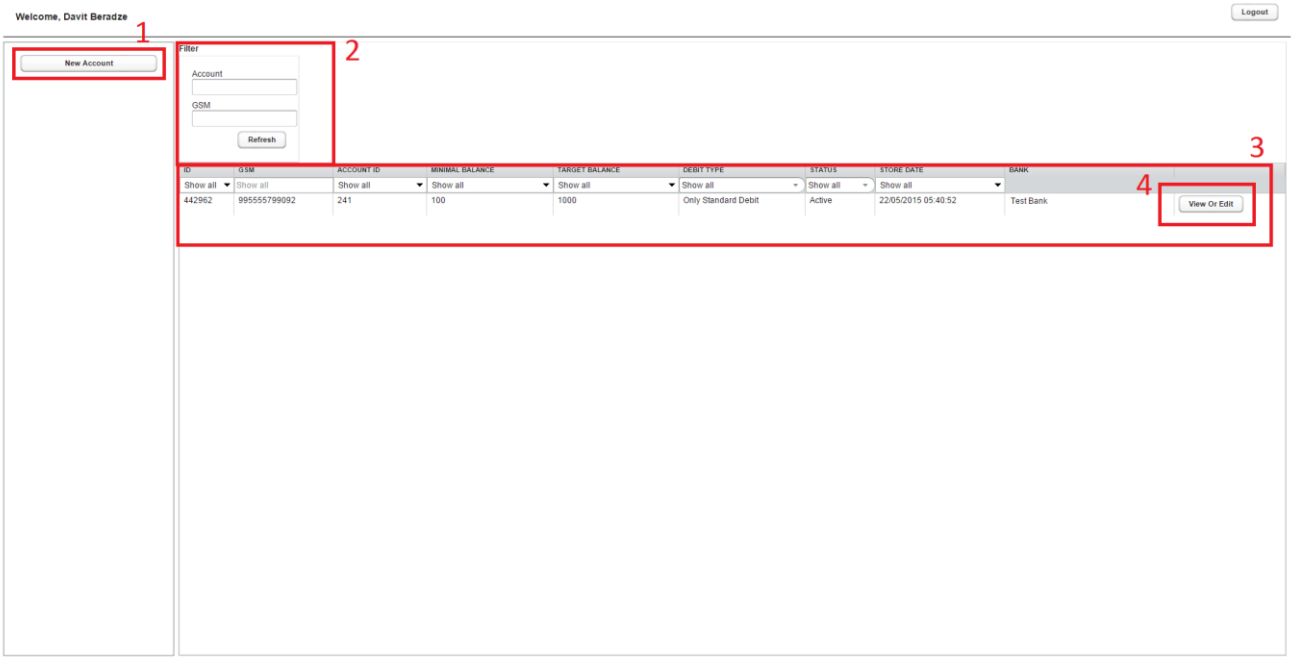
Key	Column(s)	Type		
SYS_C0050579	ID	P		

Index	Column(s)	Type		
SYS_C0050579	ID	unique		

- DD_SMS – SMS (მოკლე ტექსტური შეტყობინებები) ტექსტები სხვადასხვა ენაზე
- DD_WEBSERVICE_LOG - ვებ-სერვისის მიერ შესრულებული ყველა გარე/შიდა მოთხოვნის ისტორია.

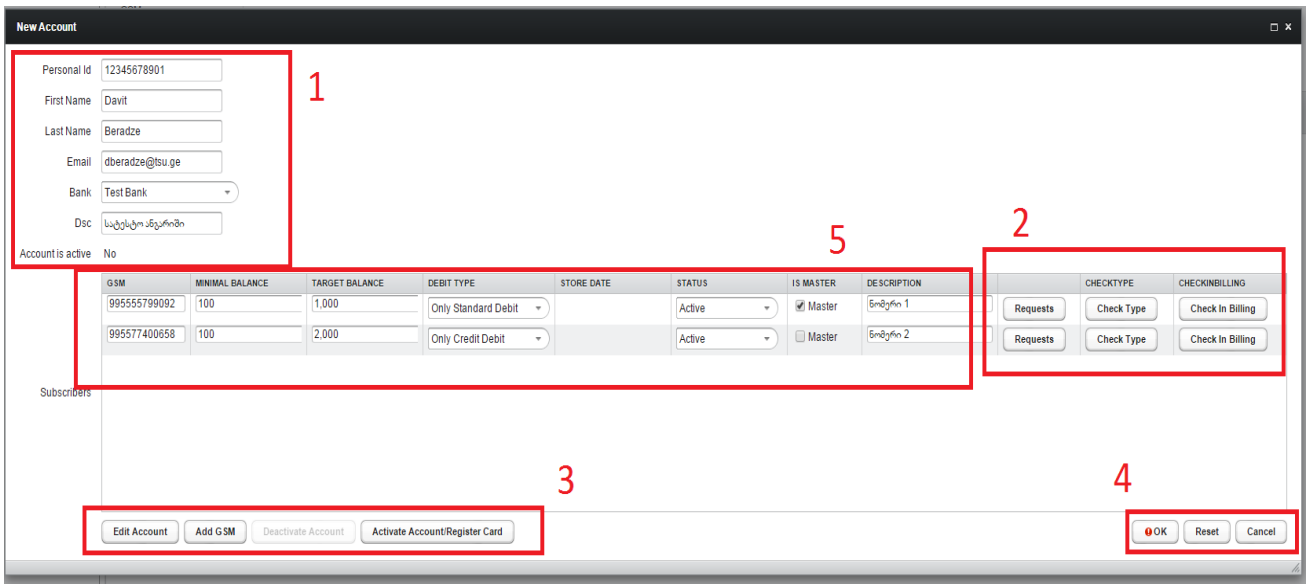
1.4 „ანგარიშის შევსების ავტომატური სისტემის“ გრაფიკული ინტერფეისი

გრაფიკული ინტერფეისი რომლის საშუალებით ხდება მომხმარებლების დარეგისტრირება, წაშლა, სისტემის მონიტორინგი, შევსებების ისტორიის ნახვა. მთავარი გვერდი.



1) მოქმედების ღილაკი რომლის საშუალებით ხდება ახალი მოთხოვნის ფანჯრის გამოძახება
2) ისტორიაში ფილტრით მოძიების ფანჯარა.
3) მოძიებული ინფორმაციის ზოგადი ველები.
4) მონაცემების დეტალური დათვალიერების ან ცვლილებების შეტანის ღილაკი.

ანგარიშის შექმნის/განახლების ფანჯარა.



1) ანგარიშის ზოგადი პარამეტრები სადაც მითითებულია მომხმარებლის სახელი, გვარი, პირადი ნომერი, იმეილი, ბანკი რომლის საშუალებით უნდა შეივსოს ანგარიში და ზოგადი აღწერა.
2) მოთხოვნების ისტორიის დათვალიერების ფანჯარა, საბილინგო სისტემაში ინფორმაციის გადამოწმების ღილაკი.
3) ანგარიშში ცვლილებების შეტანის მოთხოვნის ღილაკი, ახალი ნომრის დამატების ღილაკი, ანგარიშის გაუქმების ღილაკი, საბანკო ანგარიშის ან საბანკო ბარათის მიბმის ღილაკი
4) მონაცემების შენახვის, გაუქმების ღილაკები.
5) ანგარიშის დეტალური პარამეტრები: ნომრები, ლიმიტები, სტატუსები, აღწერა.

მოთხოვნების ისტორიის დათვალიერების ფანჯარა სადაც მითითებულია მოთხოვნის ნომერი, ტიპი, რაოდენობა, შედეგი, ხელახლა ცდის რაოდენობა, ბანკი რომლის საშუალებით მოხდა გადახდის მოთხოვნა, შემსრულებელი, შესრულების დრო, კომენტარი.

REQUEST ID	REQUEST TYPE	REQUEST AMOUNT	BALANCE	CONFIRM RESULT	RETRY COUNT	BANK	REQUESTOR	STORE DATE	COMMENT
Show all	Show all	Show all	Show all	Show all	Show all	Show all	Show all	Show all	Show all
1	Payment	190	23	Succesfully	0	Test Bank		22/05/2015 05:49:32	
1	Suspend Subscriber	190	42	Succesfully	0	Test Bank		22/05/2015 05:49:32	

გამოყენებული ტექნოლოგიები:

1) Java პროგრამული ენა
2) Vaadin - ვებ ფრეიმვორკი რომლის საშუალებით ხდება გრაფიკული გამოსახულების შექმნა. დაწერილია JAVA ენაში და ვრცელდება როგორც "ღია კოდის" პრინციპით.

თავი II.

პროგრამული კოდის აღწერა

2.1 ზოგადი სტრუქტურა

სისტემა მოიცავს შემდეგ პროექტებს:

1) DirectDebit – EJB პროექტი რომელიც მოიცავს კლასებს რომლის საშუალებით ხდება მონაცემების დამუშავება, ბაზაში შენახვა, ტრანზაქციების მართვა, ვებ-სერვისების მოთხოვნების დამუშავება.
2) DirectDebitClient – EJB კლიენტი რომელიც მოიცავს ინტერფეისებს და ბაზის ობიექტებს.
3) DirectDebitEar – EJB პროექტის გამაერთიანებელი არქივი.
4) DirectDebitGUI – სისტემის გრაფიკული ინტერფეისი რომლის საშუალებით ხდება მონაცემების რედაქტირება.
5) DirectDebitWeb – java servlet ებისგან შემდგარი პროექტი რომლის საშუალებით ხდება ბანკის ელექტრონულ გადახდების სისტემასთან კომუნიკაცია.

2.2 DirectDebit პროექტი

ge.tsu.directdebit პაკეტში შემავალი კლასები:

DirectDebitBean - კლასი რომლის საშუალებით ხდება ბაზის ტრანზაქციების მართვა.

DirectDebitTimer - კლასი რომლის საშუალებით ხდება პროცესორების ჩართვა, გათიშვა.

WebServiceBean - კლასი რომელიც ახდენს ვებ-სერვისების შექმნას და დამუშავებას.

ge.tsu.directdebit.billing პაკეტში შემავალი კლასები:

BillingInvoker - კლასი რომელიც ახდენს საბილინგო სისტემასთან კავშირს.

ge.tsu.directdebit.processor პაკეტში შემავალი კლასები:

BaseDistributor - ბაზისური (აბსტრაქტული) გამანაწილებელი რომელიც მრავალ ნაკადური პრინციპით იღებს მონაცემებს ბაზიდან და სხვადასხვა პროცესებში ანაწილებს.

BaseProcessor - ბაზისური (აბსტრაქტული) პროცესორი რომელიც მიღებულ ინფორმაციას ამუშავებს.

MainThread - მთავარი ნაკადი რომელიც ქმნის პროცესებს და დისტრიბუტორებს და მათ შორის კავშირს ამყარებს. ის აკონტროლებს რომ ყველა პროცესი მუშაობდეს გამართულად. პრობლემის აღმოჩენის შემთხვევაში ის გააჩერებს და თავიდან შექმნის პროცესს.

RequestDistributor - მოთხოვნების გამანაწილებელი კლასი რომელიც იღებს მოთხოვნებს მონაცემთა ბაზიდან და ანაწილებს მოთხოვნების პროცესორებში.

RequestProcessor - მოთხოვნების პროცესორი რომელიც იღებს და ამუშავებს მოთხოვნებს.

SubscriberDistributor - საბილინგო კონკრეტული ინფორმაციის კონკრეტული გამანაწილებელი კლასი.

SubscriberProcessor - საბილინგო კონკრეტული ინფორმაციის პროცესორი.

ge.tsu.directdebit.security პაკეტში შემავალი კლასები:

Validator - კლასი რომელიც ახდენს ვებ-სერვისის პარამეტრების შემოწმებას და დაშიფრვას (ჰეშირება) და ამოწმებს აქვს თუ არა კლიენტს ფუქნციის გამოძახების უფლება.

ge.tsu.directdebit.sms პაკეტში შემავალი კლასები:

SMSNotifier - კლასი რომელიც აგზავნის მოკლე ტექსტურ შეტყობინებას.

SMSSender - კლასი რომელიც აფორმებს მოკლე ტექსტურ შეტყობინებას და უგზავნის SMSNotifier კლასს.

ge.tsu.directdebit.ws პაკეტში შემავალი კლასები:

WsSender - კლასი რომელიც აგზავნის ვებ-სერვისის მოთხოვნებს და მონაცემთა ბაზაში ინახავს ისტორიას.

2.3 DirectDebitClient პროექტი

ge.tsu.directdebit პაკეტში შემავალი კლასები:

Configuration - კლასი რომელიც კონფიგურაციის სისტემიდან იღებს პარამეტრებს.

DirectDebitBeanRemote - ინტერფეისი რომელშიც აღწერილია ფუნქციები.

VendorWebService - ვებ-სერვისის ფუნქციების აღწერა.

WebserviceBeanRemote - ვებ-სერვისის ფუნქციების აღწერა.

ge.tsu.directdebit.entity პაკეტში შემავალი კლასები:

Account- DD_ACCOUNT მონაცემთა ბაზის ცხრილის ობიექტი.

EPayment – DD_EPAYMENT მონაცემთა ბაზის ცხრილის ობიექტი.

Permission – DD_PERMISSION მონაცემთა ბაზის ცხრილის ობიექტი.

Request – DD_REQUEST მონაცემთა ბაზის ცხრილის ობიექტი.

SMSEntity – DD_SMS მონაცემთა ბაზის ცხრილის ობიექტი.

Subscriber – DD_SUBSCRIBER მონაცემთა ბაზის ცხრილის ობიექტი.

Vendor – DD_VENDOR მონაცემთა ბაზის ცხრილის ობიექტი.

WebServiceLog – DD_WEBSERVICE_LOG მონაცემთა ბაზის ცხრილის ობიექტი.

ge.tsu.directdebit.enveloped პაკეტში შემავალი კლასები:

SubscriberObject - კლასი რომელიც გამოიყენება ვებ-სერვისებში მომხმარებლის ინფორმაციის მიმოცვლისთვის.

ge.tsu.directdebit.exception პაკეტში შემავალი კლასები:

PaymentNotProcessedException - მოთხოვნის შესრულების გამონაკლისი სიტუაციის კლასი.

ge.tsu.directdebit.types პაკეტში შემავალი ტიპების ჩამომთვლელი კლასები (ENUM):

Confirm, DebitType, EPaymentResult, Requestor, RequestType, Result, SMS, Status

2.4 DirectDebitGUI პროექტი

ge.tsu.directdebit.gui პაკეტში შემავალი კლასები:

DirectDebitGUI - მთავარი ინტერფეისი რომელიც მართავს ავტორიზაციას და პროგრამაზე დაშვებას.

Alert - კლასი რომელიც ეკრანზე გამოსატან შეტყობინებებს მართავს.

ResultHandler - მოთხოვნების შედეგების დამუშავების კლასი.

ge.tsu.directdebit.gui.server პაკეტში შემავალი კლასები:

ServerConnector - კლასი რომელიც კავშირს ამყარებს სერვერთან.

ge.tsu.directdebit.gui.ui პაკეტში შემავალი კლასები:

AccountWindow - ანგარიშის მართვის ფანჯრის კლასი.

FilterLayout - ფილტრის ფანჯრის კლასი.

MainLayout - მთავარი პანელი რომელიც აერთიანებს ყველა სხვა პანელს.

RequestsWindow - მოთხოვნების ისტორიის ფანჯრის კლასი.

SubscriberTable - ფილტრით ამოღებული ინფორმაციის კლასი.

2.5 DirectDebitWeb პროექტი

ge.tsu.directdebit.web პაკეტში შემავალი კლასები:

CheckPaymentAvail – servlet ის კლასი რომელიც ამოწმებს თუ არის გადახდის მოთხოვნა ვალიდური.

RegisterPayment – servlet ის კლასი რომელიც არეგისტრირებს მოთხოვნას სისტემაში.

ge.tsu.directdebit.web.filter პაკეტში შემავალი კლასები:

HttpsFilter – servlet ის ფილტრი რომელიც ამოწმებს მოხდა თუ არა servlet ის გამოძახება HTTPS პროტოკოლით და ინახავს გამოძახების ისტორიას.

MyHttpServletResponseWrapper - კლასი რომელიც servlet ში გადმოცემულ ინფორმაციას ახვევს ობიექტში.

ge.tsu.directdebit.web.xml პაკეტში შემავალი კლასები:

XMLBuilder - კლასი რომელიც ქმნის XML ტექსტს და გადაცემს servlet ს.

დასკვნა

„ანგარიშის შევსების ავტომატური სისტემა“ არის მზა პროდუქტი და შესაძლებელია გამოყენებული იქნას ყველა ისეთ კომპანიაში სადაც ხდება საბილინგო სისტემებით ანგარიშების მართვა, მაგალითად: მობილური კავშირგაბმულობის კომპანიები, ინტერნეტ პროვაიდერები, საკაბელო და სატელიტური ტელევიზიები, ინტერნეტ მაღაზიები, მიკრო ფინანსები, VoIP (Voice Over IP) კავშირგაბმულობის ოპერატორები, ბანკები, ინტერნეტ აუქციონები და ა.შ.

„ანგარიშის შევსების ავტომატური სისტემა“ შექმნილია თანამედროვე ტექნოლოგიების გამოყენებით რომლებიც უზრუნველყოფენ მის მოქნილობას, სისწრაფეს, ინტეგრაციის სიმარტივეს, მუშაობის პარამეტრების შეცვლის სიმარტივეს. მას შეუძლია იმუშაოს ეგრეთ წოდებული „გადატვირთვების“ გარეშე, რადგან შესრულებულია ისეთი ტექნოლოგიებს გამოყენებით რომლებიც ახდენენ ოპერატიული მეხსიერების მუდმივ კონტროლს.

სხვა ანალოგიური სისტემებისგან განსხვავებით „ანგარიშის შევსების ავტომატური სისტემა“ გააჩნია შემდეგი უპირატესობები:

1) სისტემა მარტივად შეიძლება დაინერგოს სხვადასხვა დარგის ან სპეციფიკის კომპანიაში სადაც გამოყენებულია საბილინგო სისტემები.
2) მომხმარებლებს შეუძლიათ დაარეგისტრირონ ერთზე მეტი საბილინგო ანგარიში სისტემაში და ერთი საბანკო ანგარიშით მოახდინონ მისი მართვა.
3) საბილინგო ანგარიშის მონიტორინგის ნაწილი არის მრავალ ნაკადური რაც უზრუნველყოფს რომ სისტემამ მყისიერად დააფიქსიროს თანხის შევსების აუცილებლობა.
4) ბანკის მხარე შეიძლება დარეგისტრირდეს სისტემაში ბაზაში მისი კონფიგურაციით მხოლოდ და არ საჭიროებს დამატებით სისტემის გადაკეთებას.
5) სისტემას მარტივად შეუძლია ინტეგრირდეს თვით-მომსახურებად სისტემებთან რათა მომხმარებლებმა თვითონ განახორციელონ ამ სისტემაში მათი ანგარიშების მართვა.
6) სისტემა უსაფრთხოებიდან გამომდინარე არ ინახავს არცერთ საბანკო ანგარიშის ნომერს ან ელექტრონული ბარათის რეკვიზიტებს, იგი შენახულია მხოლოდ ბანკის მხარეს.
7) სისტემის მუშაობა შესაძლებელია როგორც შიდა ისევე გარე ქსელში რადგან ის ბანკებთან ან სხვა ინტეგრირებულ სისტემებთან კომუნიკაციას ახდენს დაშიფრული კავშირით.
8) სისტემას შეუძლია იმუშაოს ნებისმიერ რელაციურ მონაცემთა ბაზასთან და არ არის შეზღუდული რომელიმე კონკრეტულ მონაცემთა ბაზაზე.

გამოყენებული ლიტერატურა

1. Mike Keith, Merrick Schincariol, Pro JPA 2, 2013
2. Andrew Lee Rubinger, Bill Burke, Enterprise JavaBeans 3.1, 2010
3. Norman Richards, Sam Griffith, Jboss: A Developers' Notebook, 2005
4. Alejandro Duarte, Vaadin 7 UI Design By Example, 2013
5. Jason Hunter, William Crawford, Java Servlet Programming, 2001

დანართი (პროგრამული რეალიზაცია)

DirectDebit

BaseDistributor.java

```
package ge.tsu.directdebit.processor;
import java.lang.Thread.State;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import org.apache.log4j.Logger;
import ge.tsu.directdebit.DirectDebitBeanLocal;
import ge.tsu.directdebit.ThreadLifetime;
/**
 * @author Davit Beradze
 *
 */
public abstract class BaseDistributor<T> implements Runnable,
ThreadLifetime {
	private Logger log = Logger.getLogger(BaseDistributor.class);
	private Thread thread;
	private boolean running;
	private BlockingQueue<T> list;
	protected DirectDebitBeanLocal ddb;

	private int[] indexer;

	int processAmount, distributorSleepCount;
	public BaseDistributor(DirectDebitBeanLocal ddb, int processAmount,
int distributorSleepCount) {
		this.processAmount = processAmount;
		this.distributorSleepCount = distributorSleepCount;
		this.ddb = ddb;
		list = new LinkedBlockingQueue<T>();
		indexer = new int[processAmount];

		cleanIndexer();
	}
	private long time = System.currentTimeMillis();
	private long all = 0;
	@Override
	public void run() {
		running = true;
		while (running) {
			try {
```

```
                    List<T> subscribers = getList();
                    if (subscribers != null && subscribers.size() > 0)
                         log.info(this.getClass().getSimpleName() + "
TIME: " + (System.currentTimeMillis() - time)
                                        + ", Overall: " + all + "  SIZE:" +
subscribers.size());
                    all += (System.currentTimeMillis() - time);
                    time = System.currentTimeMillis();
                    // ensure clean
                    list.clear();
                    list.addAll(subscribers);
                    while (true) {
                         if (isEmpty() && isEverythingReady()) {
                              break;
                         }
                         try {
                              Thread.sleep(distributorSleepCount);
                         } catch (InterruptedException e) {
                              log.error(e);
                         }
                    }
               } catch (Exception e) {
                    log.error(e);
               }
               try {
                    Thread.sleep(distributorSleepCount);
               } catch (InterruptedException e) {
                    log.error(e);
               }
          }
     }
     protected abstract List<T> getList();
     private boolean isEverythingReady() {
          for (int i = 0; i < indexer.length; i++) {
               if (indexer[i] > 0)
                    return false;
          }
          return true;
     }
     @Override
     public void start() {
          thread = new Thread(this, this.getClass().getSimpleName());
          log.info(this.getClass().getSimpleName() + " Started!");
          thread.start();
     }
     @Override
     public void stop() {
          running = false;
     }
     public boolean isLive() {
```

```
            if (thread == null)
                return false;
            if (thread.getState() == State.TERMINATED)
                return false;
            return running;
        }

        private void cleanIndexer() {
            for (int i = 0; i < indexer.length; i++) {
                indexer[i] = 0;
            }
        }
        private boolean isEmpty() {
            return list.isEmpty();
        }
        public T getNextElement(int index) {
            try {
                indexer[index] = 1;
                return list.remove();
            } catch (NoSuchElementException e) {
                indexer[index] = 0;
            }
            return null;
        }
    }
```

BaseProcessor.java

```
package ge.tsu.directdebit.processor;
import java.util.Date;
import java.util.concurrent.TimeUnit;
import ge.tsu.directdebit.billing.BillingInvoker;
import ge.tsu.directdebit.entity.Subscriber;
import ge.tsu.directdebit.types.DebitType;
import ge.tsu.directdebit.types.Status;
public abstract class BaseProcessor {
        /**
         * Get a diff between two dates
         *
         * @param date1
         *            the oldest date
         * @param date2
         *            the newest date
         * @param timeUnit
         *            the unit in which you want the diff
         * @return the diff value, in the provided unit
         */
        public long getDateDiff(Date newDate, Date oldDate, TimeUnit
timeUnit) {
            long diffInMillies = newDate.getTime() - oldDate.getTime();
            return timeUnit.convert(diffInMillies, TimeUnit.MILLISECONDS);
        }
```

```
	public RequestAmont calculateRequestAmount(Subscriber subscriber)
throws Exception {
		int balance;
		int previousMonthDebt;
		int requestAmount;
		if (subscriber.getStatus() != Status.ACTIVE)
			throw new IllegalArgumentException("Subscriber( " +
subscriber.getGsm()
				+ ") is not active and can't calculate
amount!");
		DebitType type = subscriber.getDebitType();
		if (type == null)
			throw new Exception("DebitType required");
		balance =
BillingInvoker.get().getBalance(subscriber.getGsm());
		if (type == DebitType.STANDARD || type ==
DebitType.STANDARD_AND_CREDIT) {
			// check if subscriber need to fill balance
			if (balance < subscriber.getMinimalBalance()) {
				requestAmount = subscriber.getTargetBalance() -
balance;
				subscriber.setComment("Payment request, balance = "
+ balance + " , requestAmount = " + requestAmount);
				return new RequestAmont(requestAmount, balance);
			}
		}
		if (type == DebitType.CREDIT_ONLY || type ==
DebitType.STANDARD_AND_CREDIT) {
			previousMonthDebt = new
Double(BillingInvoker.get().getPreviousMonthDebt(subscriber.getGsm()) *
100)
				.intValue();
			if (previousMonthDebt > 0) {
				subscriber.setComment("Previous month Debt fill
request, Debt = " + previousMonthDebt + " Balance = "
					+ balance + "  , requestAmount = " +
previousMonthDebt);
				return new RequestAmont(previousMonthDebt,
balance);
			}
		}
		return new RequestAmont(0, balance);
	}
	class RequestAmont {
		public RequestAmont(int requestAmount, int balance) {
			this.requestAmount = requestAmount;
			this.balance = balance;
		}
		int requestAmount;
		int balance;
```

```
		@Override
		public String toString() {
			return "REQUEST_AMOUNT = " + requestAmount + " BALANCE =
" + balance;
		}
	}
}
```

MainThread.java

```
package ge.tsu.directdebit.processor;
import org.apache.log4j.Logger;
import ge.tsu.directdebit.Configuration;
import ge.tsu.directdebit.DirectDebitBeanLocal;
import ge.tsu.directdebit.ThreadLifetime;
/**
 * @author Davit Beradze
 *
 */
public class MainThread implements Runnable, ThreadLifetime {
	private Logger log = Logger.getLogger(MainThread.class);
	private Thread thread;
	private SubscriberProcessor[] subscriberProcessors;
	private RequestProcessor[] requestProcessors;
	private SubscriberDistributor subscriberDistributor;
	private RequestDistributor requestDistributor;
	private DirectDebitBeanLocal ddb;
	private boolean running;
	public MainThread(DirectDebitBeanLocal ddb) {
		this.ddb = ddb;
	}
	@Override
	public void run() {
		subscriberProcessors = new
SubscriberProcessor[Configuration.get().getSubscriberProcessAmount()];
		requestProcessors = new
RequestProcessor[Configuration.get().getRequestProcessAmount()];
		running = true;
		while (running) {
			try {
				if (subscriberDistributor == null)
					subscriberDistributor = new
SubscriberDistributor(ddb, Configuration.get()
						.getSubscriberProcessAmount(),
Configuration.get().getSubscriberDistributorSleepCount());
				if (!subscriberDistributor.isLive())
					subscriberDistributor.start();
				if (requestDistributor == null)
					requestDistributor = new
RequestDistributor(ddb, Configuration.get().getRequestProcessAmount(),

	Configuration.get().getRequestDistrubutorSleepCount());
```

```
                    if (!requestDistributor.isLive())
                         requestDistributor.start();
                    for (int i = 0; i < subscriberProcessors.length;
i++) {
                         SubscriberProcessor processor =
subscriberProcessors[i];
                         if (processor == null) {
                              try {
                                   log.info("TRY START PROCESSOR ID="
+ i);
                                   subscriberProcessors[i] = new
SubscriberProcessor(ddb, subscriberDistributor, i);
                                   processor =
subscriberProcessors[i];
                              } catch (Exception e) {
                                   log.error(e);
                              }
                         }
                         if (processor != null && !processor.isLive())
{
                              processor.start();
                         }
                    }
                    for (int i = 0; i < requestProcessors.length; i++)
{
                         RequestProcessor processor =
requestProcessors[i];
                         if (processor == null) {
                              try {
                                   log.info("TRY START PROCESSOR ID="
+ i);
                                   requestProcessors[i] = new
RequestProcessor(ddb, requestDistributor, i);
                                   processor = requestProcessors[i];
                              } catch (Exception e) {
                                   log.error(e);
                              }
                         }
                         if (processor != null && !processor.isLive())
{
                              processor.start();
                         }
                    }
                    try {

     Thread.sleep(Configuration.get().getMainThreadSleepCount());
                    } catch (InterruptedException e) {
                         e.printStackTrace();
                    }
               } catch (Exception e) {
```

```
                    log.error("MainThread", e);
                }
            }
        }
        @Override
        public void start() {
                thread = new Thread(this, "MainThread");
            thread.start();
        }
        @Override
        public void stop() {
            running = false;
            try {
                for (SubscriberProcessor proc : subscriberProcessors) {
                    proc.stop();
                }
                for (RequestProcessor proc : requestProcessors) {
                    proc.stop();
                }
                subscriberDistributor.stop();
                requestDistributor.stop();
            } catch (Exception e) {
                log.error(e);
            }
        }
        @Override
        public boolean isLive() {
            // ignore
            return false;
        }
    }
```

RequestDistributor.java

```
package ge.tsu.directdebit.processor;
import java.util.List;
import ge.tsu.directdebit.DirectDebitBeanLocal;
import ge.tsu.directdebit.entity.Request;
public class RequestDistributor extends BaseDistributor<Request> {

    public RequestDistributor(DirectDebitBeanLocal ddb, int
processAmount, int distributorSleepCount) {
        super(ddb, processAmount, distributorSleepCount);
    }
    @Override
    protected List<Request> getList() {
        return ddb.getRequests();
    }
}
```

RequestProcessor.java

```
package ge.tsu.directdebit.processor;
import java.lang.Thread.State;
```

```
import java.util.Date;
import java.util.concurrent.TimeUnit;
import org.apache.log4j.Logger;
import ge.tsu.directdebit.Configuration;
import ge.tsu.directdebit.DirectDebitBeanLocal;
import ge.tsu.directdebit.ThreadLifetime;
import ge.tsu.directdebit.billing.BillingInvoker;
import ge.tsu.directdebit.entity.Request;
import ge.tsu.directdebit.entity.Subscriber;
import ge.tsu.directdebit.exception.PaymentNotProcessedException;
import ge.tsu.directdebit.sms.SMSSender;
import ge.tsu.directdebit.types.Confirm;
import ge.tsu.directdebit.types.RequestType;
import ge.tsu.directdebit.types.Requestor;
import ge.tsu.directdebit.types.SMS;
import ge.tsu.directdebit.types.Status;
import ge.tsu.directdebit.ws.WsSender;
/**
 * @author Davit Beradze
 *
 */
public class RequestProcessor extends BaseProcessor implements Runnable,
ThreadLifetime {
	private Logger log = Logger.getLogger(RequestProcessor.class);
	private Thread thread;
	private boolean running;
	private DirectDebitBeanLocal ddb;
	private WsSender ws;
	private RequestDistributor distributor;
	private int processIndex;
	public RequestProcessor(DirectDebitBeanLocal ddb,
RequestDistributor distributor, int processIndex) {
		this.ddb = ddb;
		this.processIndex = processIndex;
		this.distributor = distributor;
		ws = new WsSender(ddb);
	}
	public RequestProcessor(DirectDebitBeanLocal ddb) {
		this.ddb = ddb;
		processIndex = -1;
		ws = new WsSender(ddb);
	}
	@Override
	public void run() {
		running = true;
		Request request = null;
		while (running) {
			try {
				request = distributor.getNextElement(processIndex);
				// wait for distributor
```

```
                    if (request == null) {
                        try {

    Thread.sleep(Configuration.get().getRequestProcessorSleepCount());
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                        continue;
                    }
                    processRequest(request, true);
                } catch (Throwable t) {
                    log.error("RequestProcessor", t);
                    try {
                        if (request != null) {
                            request.addComment("Error:" +
t.getMessage());
                            request.setConfirm(Confirm.N);
                            ddb.updateRequest(request);
                        }
                    } catch (Exception e) {
                        log.error("RequestProcessor error to db", e);
                    }
                }
                try {

    Thread.sleep(Configuration.get().getRequestProcessorSleepCount());
                } catch (InterruptedException e) {
                    log.error(e);
                }
            }
        }
        public void processSynchRequest(Request request) throws Throwable {
            try {
                processRequest(request, false);
            } catch (Throwable t) {
                log.error("processSynchRequest", t);
                try {
                    if (request != null) {
                        request.addComment("Error:" + t.getMessage());
                        request.setConfirm(Confirm.N);
                        ddb.newRequest(request);
                    }
                } catch (Exception e) {
                    log.error("RequestProcessor error to db", e);
                }
                throw t;
            }
        }
        private void processRequest(Request request, boolean isAsync)
throws Exception {
```

```
            Subscriber subscriber = null;
            switch (request.getRequestType()) {
            case PAYMENT:
            case MANUAL_PAYMENT:
                  if (request.getRetryCount() >
Configuration.get().getMaxRetryAmount()) {
                        request.addComment("retry count (" +
request.getRetryCount() + " ) > max retry amount ("
                                    +
Configuration.get().getMaxRetryAmount() + ")");
                        request.setConfirm(Confirm.N);
                        if (isAsync)
                              ddb.updateRequest(request);
                        else
                              ddb.newRequest(request);
                        Request newRequest = new Request();

      newRequest.setRequestType(RequestType.SUSPEND_SUBSCRIBER);
                        newRequest.setComment("retry limit");
                        newRequest.setAccountId(request.getAccountId());
                        newRequest.setVendor(request.getVendor());
                        newRequest.setGsm(request.getGsm());
                        newRequest.setSubscriber(request.getSubscriber());
                        newRequest.setConfirm(Confirm.U);

      newRequest.setRequestor(Requestor.DDMS_REQUEST.name());
                        ddb.newRequest(newRequest);
                        return;
                  }
                  if (request.getRetryCount() > 0) {
                        long diffMinutes = getDateDiff(new Date(),
request.getFd(), TimeUnit.MINUTES);
                        log.debug("DiffMinutes = " + diffMinutes + " after
retry");
                        // ignore this request
                        if (diffMinutes <
Configuration.get().getNextRetryInMinutes()) {
                              return;
                        }
                  }
                  long resultCode = 0;
                  try {
                        // recalculate request amount
                        RequestAmont requestAmount =
calculateRequestAmount(request.getSubscriber());

      request.setRequestAmount(requestAmount.requestAmount);
                        request.setBalance(requestAmount.balance);
                        if (request.getRequestAmount() <
Configuration.get().getMinimalRequestAmount()) {
```

```
                        // request amount is too low
                        request.setConfirm(Confirm.I);
                        request.setComment("request amount is too low
(" + request.getRequestAmount() + " < "
                                    +
Configuration.get().getMinimalRequestAmount() + ") or balance changed = "
                                    + requestAmount.balance);
                        if (isAsync)
                              ddb.updateRequest(request);
                        else
                              ddb.newRequest(request);
                        return;
                  }
                  try {
                        if
(!BillingInvoker.get().fixServiceInBilling(request.getGsm())) {
                              throw new Exception("Subscriber = " +
request.getGsm() + " has not active service in billing");
                        }
                  } catch (Exception e) {


                        throw e;
                  }

                  log.info("Trying send payment request " + request);
                  resultCode = ws.paymentRequest(request);
                  log.info("Payent request result =  " + resultCode);
                  if (resultCode >= 0) {
                        request.setResultCode(resultCode);
                        request.setConfirm(Confirm.Y);

                        if (isAsync)
                              ddb.updateRequest(request);
                        else
                              ddb.newRequest(request);
                        // END
                        return;
                  }
                  else
                  if (resultCode == -7) {
                      //SMS not have enough balance on VISA
                        SMSSender.get().send(request.getSubscriber(),
SMS.ERROR_NO_ENOUGH_BALANCE, ddb);
                  } else if (resultCode == -5) {
                        //SMS you have problem on VISA
                        SMSSender.get().send(request.getSubscriber(),
SMS.ERORR_IN_PAYMENT, ddb);
                  } else {
                        //SMS other error
```

```
                        SMSSender.get().send(request.getSubscriber(),
SMS.ERROR_GENERAL, ddb);
                    }

                        throw new
PaymentNotProcessedException("ResultCode = " + resultCode);
                } catch (Exception e) {
                    log.error("SendPayment GSM = " + request.getGsm(),
e);
                    request.addComment(e.getMessage());
                    request.setConfirm(Confirm.N);
                    request.setResultCode(resultCode);
                    if (isAsync)
                        ddb.updateRequest(request);
                    else
                        ddb.newRequest(request);

                    // Request newRequest = new Request();
                    // request.setRequestType(RequestType.PAYMENT);
                    request.setComment("retry");
                    // request.setBalance(request.getBalance());
                    //
request.setRequestAmount(request.getRequestAmount());
                    // request.setAccountId(request.getAccountId());
                    // request.setVendor(request.getVendor());
                    // request.setGsm(request.getGsm());
                    // request.setSubscriber(request.getSubscriber());
                    request.setConfirm(Confirm.U);

     request.setRequestor(Requestor.DDMS_REQUEST.name());
                    request.setRetryCount(request.getRetryCount());
                    request.addRetry();
                    if (isAsync)
                        ddb.updateRequest(request);
                    else
                        ddb.newRequest(request);
                }
                break;
            case DEACTIVATE_SUBSCRIBER:
                subscriber = ddb.getSubscriberByGSM(request.getGsm());
                subscriber.setStatus(Status.DEACTIVATED);
                subscriber.setRequestId(request.getRequestId());
                subscriber.setComment("DEACTIVATE by " +
Requestor.DDMS_REQUEST.name());
                if (request.getSubscriber() != null)

     subscriber.setResult(request.getSubscriber().getResult());
                ddb.updateSubscriber(subscriber);
```

```
BillingInvoker.get().deactivateService(request.getGsm());

                request.setConfirm(Confirm.Y);
                if (isAsync)
                     ddb.updateRequest(request);
                else
                     ddb.newRequest(request);
                break;
           case ACTIVATE_SUBSCRIBER:
                subscriber = ddb.getSubscriberByGSM(request.getGsm());
                subscriber.setStatus(Status.ACTIVE);
                subscriber.setRequestId(request.getRequestId());
                subscriber.setComment("ACTIVATE by " +
Requestor.DDMS_REQUEST.name());
                ddb.updateSubscriber(subscriber);
                BillingInvoker.get().activateService(request.getGsm());

                request.setConfirm(Confirm.Y);
                if (isAsync)
                     ddb.updateRequest(request);
                else
                     ddb.newRequest(request);
                break;
           case SUSPEND_SUBSCRIBER:
                subscriber = ddb.getSubscriberByGSM(request.getGsm());
                subscriber.setStatus(Status.SUSPENDED);
                subscriber.setRequestId(request.getRequestId());
                subscriber.setComment("SUSPEND by " +
Requestor.DDMS_REQUEST.name());
                ddb.updateSubscriber(subscriber);

                request.setConfirm(Confirm.Y);
                if (isAsync)
                     ddb.updateRequest(request);
                else
                     ddb.newRequest(request);
                break;
           default:
                throw new Exception("UNKNOWN request type");
           }
      }
      @Override
      public void start() {
      //   if (thread == null)
                thread = new Thread(this, "RequestProcessor(" +
processIndex + ")");
           log.info("Started Process(" + processIndex + ")");
           thread.start();
      }
```

```
	@Override
	public void stop() {
		running = false;
	}
	public boolean isLive() {
		if (thread == null)
			return false;
		if (thread.getState() == State.TERMINATED)
			return false;
		return running;
	}
}
```

SubscriberDistributor.java

```
package ge.tsu.directdebit.processor;
import java.util.List;
import ge.tsu.directdebit.DirectDebitBeanLocal;
import ge.tsu.directdebit.entity.Subscriber;
public class SubscriberDistributor extends BaseDistributor<Subscriber> {

	public SubscriberDistributor(DirectDebitBeanLocal ddb, int
processAmount, int distributorSleepCount) {
		super(ddb, processAmount, distributorSleepCount);

	}
	@Override
	protected List<Subscriber> getList() {
		return ddb.getActiveSubscribers();
	}
}
```

SubscriberProcessor.java

```
package ge.tsu.directdebit.processor;
import java.lang.Thread.State;
import java.util.Date;
import java.util.concurrent.TimeUnit;
import org.apache.log4j.Logger;
import ge.tsu.billing.facade.entity.PaymentMethod;
import ge.tsu.directdebit.Configuration;
import ge.tsu.directdebit.DirectDebitBeanLocal;
import ge.tsu.directdebit.ThreadLifetime;
import ge.tsu.directdebit.billing.BillingInvoker;
import ge.tsu.directdebit.entity.Request;
import ge.tsu.directdebit.entity.Subscriber;
import ge.tsu.directdebit.types.Confirm;
import ge.tsu.directdebit.types.RequestType;
import ge.tsu.directdebit.types.Requestor;
import ge.tsu.directdebit.types.Result;
import ge.tsu.directdebit.types.Status;
/**
 * @author Davit Beradze
 *
```

```
 */
public class SubscriberProcessor extends BaseProcessor implements
Runnable, ThreadLifetime {
	private Logger log = Logger.getLogger(SubscriberProcessor.class);
	private DirectDebitBeanLocal ddb;
	private SubscriberDistributor distributor;
	private Thread thread;
	private boolean running;
	private int processIndex;
	private RequestProcessor synchRequestProcessor;
	public SubscriberProcessor(DirectDebitBeanLocal ddb,
SubscriberDistributor distributor, int processIndex) {
		this.ddb = ddb;
		this.distributor = distributor;
		this.processIndex = processIndex;
		synchRequestProcessor = new RequestProcessor(ddb);
	}
	@Override
	public void run() {
		running = true;
		Subscriber subscriber = null;
		while (running) {
			try {
				subscriber =
distributor.getNextElement(processIndex);
				// wait for distributor
				if (subscriber == null) {
					try {

	Thread.sleep(Configuration.get().getSubscriberProcessorSleepCount()
);
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
					continue;
				}
				// check if account is available
				if (subscriber.getAccount() == null) {
					throw new Exception("Account is null");
				}
				// if account is not active ignore
				if (!subscriber.getAccount().getActive()) {
					if (subscriber.getResult() !=
Result.AccountIsNotActive) {

	subscriber.setResult(Result.AccountIsNotActive);
						ddb.updateSubscriber(subscriber);
					}
					continue;
				}
```

```
                    // check if bank is suspended
                    if
(!subscriber.getAccount().getVendor().getActive()) {
                         if (subscriber.getResult() !=
Result.BankSuspended) {

     subscriber.setResult(Result.BankSuspended);

                              ddb.updateSubscriber(subscriber);
                         }
                         continue;
                    }
                    try {
                         if
(!BillingInvoker.get().fixServiceInBilling(subscriber.getGsm())) {
                              throw new Exception("Subscriber = " +
subscriber.getGsm()
                                        + " has not active service in
billing");
                         }
                    } catch (Exception e) {
                         if (subscriber.getResult() !=
Result.BillingServiceNotActive) {

     subscriber.setResult(Result.BillingServiceNotActive);

                              // unsufficientbalance in message

                              ddb.updateSubscriber(subscriber);
                         }
                         continue;
                    }
                    PaymentMethod method =
BillingInvoker.get().getPaymentMethod(subscriber.getGsm());
                    if (method == PaymentMethod.UNKNOWN) {
                         if (subscriber.getResult() !=
Result.PortedOut) {
                              subscriber.setResult(Result.PortedOut);
                              Request request = new Request();

     request.setAccountId(subscriber.getAccountId());

     request.setVendor(subscriber.getAccount().getVendor());
                              request.setGsm(subscriber.getGsm());
                              request.setConfirm(Confirm.U);

     request.setRequestor(Requestor.DDMS.name());
                              request.setRetryCount(0);
                              request.setSubscriber(subscriber);
```

```
	request.setRequestType(RequestType.DEACTIVATE_SUBSCRIBER);

	synchRequestProcessor.processSynchRequest(request);
				}
				continue;
			}
			if
(!BillingInvoker.get().isPaymentPossible(subscriber.getGsm())) {
				if (subscriber.getResult() !=
Result.NotActiveInBilling) {

	subscriber.setResult(Result.NotActiveInBilling);

					ddb.updateSubscriber(subscriber);
				}

				continue;
			}
			if (subscriber.getResult() == Result.Paid) {
				long diffMinutes = getDateDiff(new Date(),
subscriber.getFd(), TimeUnit.MINUTES);
				log.debug("DiffMinutes = " + diffMinutes + "
after Paid");
				// ignore this request
				if (diffMinutes <
Configuration.get().getNextPaymentInMinutes()) {
					continue;
				}
			}
			// check if already requested payment
			if (subscriber.getResult() ==
Result.PeymentRequested) {
				Request request =
ddb.getRequestByRequestId(subscriber.getRequestId());
				// resume listening
				if (request == null || request.getConfirm() ==
Confirm.Y || request.getConfirm() == Confirm.I) {
					// TODO: continue?
					subscriber.setStatus(Status.ACTIVE);
					subscriber.setResult(Result.Paid);
					subscriber.setRequestId(null);
					subscriber.setComment("Request processed
or not found in db.  Resume listening");
					ddb.updateSubscriber(subscriber);
					continue;
				} else {
					log.debug("Subscriber(" +
request.getGsm() + ") ignored, cause Request("
```

```
                                        + (request != null ?
request.getId() + "" : "NULL") + ").Confirm = "
                                        + request.getConfirm());
                                continue;
                            }
                        }
                        RequestAmont requestAmount =
calculateRequestAmount(subscriber);
                        if (requestAmount.requestAmount >
Configuration.get().getMinimalRequestAmount()) {
                            if (checkOverLimit(subscriber))
                                continue;
                            Request request = new Request();
                            request.setBalance(requestAmount.balance);

      request.setAccountId(subscriber.getAccountId());

      request.setVendor(subscriber.getAccount().getVendor());
                            request.setGsm(subscriber.getGsm());
                            request.setConfirm(Confirm.U);

      request.setRequestAmount(requestAmount.requestAmount);
                            request.setRequestor(Requestor.DDMS.name());
                            request.setRetryCount(0);
                            request.setSubscriber(subscriber);
                            request.setRequestType(RequestType.PAYMENT);
                            Request resultRequest =
ddb.newRequest(request);


      //subscriber.setStatus(Status.PAYMENT_REQUESTED);

      subscriber.setRequestId(resultRequest.getRequestId());
                            subscriber.setResult(Result.PeymentRequested);
                            ddb.updateSubscriber(subscriber);
                        } else {
                            if (requestAmount.requestAmount > 0)
                                log.info("Subscriber(" +
subscriber.getGsm() + ") ignored because requestAmount("
                                        + requestAmount.requestAmount
+ " is too low");
                        }
                    } catch (Throwable t) {
                        log.error("SubscriberProcessor", t);
                        try {
                            if (subscriber != null) {
                                subscriber.addComment(t.getMessage());
                                ddb.updateSubscriber(subscriber);
                            }
                        } catch (Exception e) {
```

```
                                log.error("SubscriberProcessor error to db",
e);
                            }
                        }
                        try {

            Thread.sleep(Configuration.get().getSubscriberProcessLoopSleepCount
());
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
            }
            /**
             * check subscriber if has already big number of payments
             */
            private boolean checkOverLimit(Subscriber subscriber) {
                    int oldRequestSize =
ddb.getSuccesfullRequestsSize(subscriber.getGsm(),
subscriber.getAccountId(),
                            Configuration.get().getLimitRequestDays());
                    if (oldRequestSize >=
Configuration.get().getMaximumPaymentRequest()) {
                        if (subscriber.getResult() == Result.OverLimit)
                            return true;

                        subscriber.setResult(Result.OverLimit);
                        subscriber.setRequestId(null);
                        subscriber.setComment("payment request number (" +
oldRequestSize + ") is more than limit ("
                                    +
Configuration.get().getMaximumPaymentRequest() + ")");
                        ddb.updateSubscriber(subscriber);
                        return true;
                    }
                    return false;
            }
            @Override
            public void start() {
            //    if (thread == null)
                        thread = new Thread(this, "Process(" + processIndex +
")");
                    log.info("Started Process(" + processIndex + ")");
                    thread.start();
            }
            @Override
            public void stop() {
                    running = false;
            }
            public boolean isLive() {
```

```
            if (thread == null)
                  return false;
            if (thread.getState() == State.TERMINATED)
                  return false;
            return running;
      }
}
```

Validator.java

```
package ge.tsu.directdebit.security;

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import ge.tsu.directdebit.entity.Permission;

import ge.tsu.directdebit.entity.Vendor;

/**

 * @author Davit Beradze

 *

 */

public class Validator {

        public String md5Generator(String... params) {

                StringBuilder ourSignature = new StringBuilder();

                for (int i = 0; i < params.length; i++) {

                        String string = params[i];

                        if (i != 0)

                                ourSignature.append(",");
```

```
            if (string != null)
                ourSignature.append(string);
        }
        return stringToMd5(ourSignature.toString());
    }
    private String stringToMd5(String text) {
        try {
            MessageDigest m = MessageDigest.getInstance("MD5");
            m.reset();
            m.update(text.getBytes());
            byte[] digest = m.digest();
            BigInteger bigInt = new BigInteger(1, digest);
            String hashtext = bigInt.toString(16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            return hashtext;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return text;
```

```
}
public boolean validate(String signature, String signaturePassword, String... params) {
	StringBuilder ourSignature = new StringBuilder();
	for (int i = 0; i < params.length; i++) {
		String string = params[i];
		if (i != 0)
			ourSignature.append(",");
		if (string != null)
			ourSignature.append(string);
	}
	ourSignature.append("," + signaturePassword);
	return signature.equalsIgnoreCase(this.stringToMd5(ourSignature.toString()));
}
public boolean validatePermission(String method, Vendor vendor) {
	for (Permission item : vendor.getPermissions()) {
		if (item.getMethod().equalsIgnoreCase(method))
			return true;
	}
	return false;
}
```

```
	public boolean validateAllowedParams(String param, Vendor vendor) {
		for (Permission item : vendor.getPermissions()) {
			if (item.getAllowedParams().contains(param))
				return true;
		}
		return false;
	}
}
```

WsSender.java

```
package ge.tsu.directdebit.ws;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;
import org.apache.log4j.Logger;
import ge.tsu.directdebit.DirectDebitBeanLocal;
import ge.tsu.directdebit.VendorWebService;
import ge.tsu.directdebit.entity.Request;
import ge.tsu.directdebit.entity.Vendor;
import ge.tsu.directdebit.entity.WebServiceLog;
import ge.tsu.directdebit.security.Validator;
```

```
/**
 * @author Davit Beradze
 *
 */
public class WsSender {
	private Logger log = Logger.getLogger(WsSender.class);
	private Validator validator;
	private DirectDebitBeanLocal ddb;
	public WsSender(DirectDebitBeanLocal ddb) {
		validator = new Validator();
		this.ddb = ddb;
	}
	public long paymentRequest(Request request) throws Exception {
		Service service = null;
		WebServiceLog wsLog = new WebServiceLog();
		wsLog.setMethod("paymentRequest");
		wsLog.setIncome(false);
		try {
			Vendor bank = request.getVendor();


			String gsm = request.getGsm().substring(3);
```

```
                    String signature = validator.md5Generator(bank.getRemoteIdent(),
request.getAccountId() +"", gsm,
                                        request.getRequestAmount() + "", request.getId() + "",
bank.getRemoteSignaturePassword());
                    wsLog.setParams(bank.getRemoteIdent(), signature,
request.getAccountId() + "", gsm,
                                        request.getRequestAmount() + "", request.getId() + "");
                    service = Service.create(new URL(bank.getWsUrl()),
                                        new QName(bank.getWsNamespace(),
bank.getWsServicename()));
                    VendorWebService port = service.getPort(VendorWebService.class);
                    BindingProvider bp = (BindingProvider) port;

          bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
bank.getEndPointURL());
                    long response = port.paymentRequest(bank.getRemoteIdent(), signature,
request.getAccountId(),
                                        gsm, request.getRequestAmount(), request.getId() + "");
                    wsLog.setResponse(response);
                    ddb.storeWSLog(wsLog);
                    return response;
                } catch (Exception e) {
                    log.error("paymentRequest", e);
```

```
                wsLog.setResponse(-1);
                wsLog.setException(e.getMessage());
                ddb.storeWSLog(wsLog);
                throw e;
            }
        }
}
```

DirectDebitTimer.java

```
package ge.tsu.directdebit;
import javax.ejb.EJB;
import org.jboss.ejb3.annotation.Management;
import org.jboss.ejb3.annotation.Service;
import ge.tsu.directdebit.processor.MainThread;
/**
 * @author Davit Beradze
 *
 */
@Service(objectName = "DirectDebit:service=DirectDebitTimer")
@Management
public class DirectDebitTimer implements ThreadLifetime {
        private MainThread main;
```

```
        @EJB
        DirectDebitBeanLocal ddb;
        @Override
        public void start() throws Exception {
                if (main == null)
                        main = new MainThread(ddb);
                main.start();
        }
        @Override
        public void stop() {
                if (main != null)
                        main.stop();
        }
        @Override
        public boolean isLive() {
                // ignore
                return false;
        }
}
```

ThreadLifetime.java

```
package ge.tsu.directdebit;
/**
```

```
 * @author Davit Beradze
 *
 */
public interface ThreadLifetime {
     public void start() throws Exception;
     public void stop() throws Exception;
     boolean isLive();
}
```