Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабороторная работа №4

Выполнил студент:

Колпикова Ксения Денисовна
группа: M32071

Проверил:

Чикишев Константин Максимович

Санкт-Петербург,
2022 г.

1.1. Текст задания

4 лабораторная
Владельцы недовольны, что информацию о котиках может получить кто угодно. В этой лабораторной мы добавим авторизацию к сервису.
Добавляется роль администратора. Он имеет доступ ко всем методам и может создавать новых пользователей. Пользователь связан с владельцем в соотношении 1:1.
Методы по получению информации и котиках и владельцах должны быть защищены Spring Security. Доступ к соответствующим endpoint'ам имеют только владельцы котиков и администраторы. Доступ к методам для фильтрации имеют все авторизованные пользователи, но на выходе получают только данные о своих котиках.
Внимание: эндпоинты, созданные на предыдущем этапе, не должны быть удалены.

1.2. Решение

Листинг 1.1: Main.java

```
import models.Bank;
import models.CentralBank;
import models.Client;
import models.PercentOfDeposit;
import services.ClientBuilder;
import tools.BanksException;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws BanksException {
        CentralBank central = null;
        Scanner in = new Scanner(System.in);
        System.out.println("1 – РЎРѕР·РґР°РЅРёРµ РїРѕР»РЬР·РѕРІР°
    РєР»РёРµРЬС‚Р° ");
        System.out.println("2 – РЎРѕР·РґР°РЅРёРµ РєР»РёРµРЬС‚Р° СЃ
    РѕРїС‚Р°РЅРёС‡РµРЅРёРєР¬РјРё");
        System.out.println("Name, Surname, Address, Passport");
        String str = in.nextLine();
        ClientBuilder creation = null;
        Client client = null;
        Bank bank = null;
        switch (str)
        {
            case "1":
                client = creation.changeName(in.nextLine()).
                        changeSurname(in.nextLine()).changeAddress(in.
    nextLine()).
                        changePassport(in.nextLine()).create();
                break;
            case "2":
                client = creation.changeName(in.nextLine()).
    changeSurname(in.nextLine()).create();
                break;
        }

        while (str != "exit")
        {
            System.out.println("1 – РЎРѕР·РґР°РЅРёРµ Р±Р°РЅРєР°");
            System.out.println("2 – Р"РѕР±Р°РІР»РµРЬРёРµ РєР»РёРµРЬС‚Р°
    РІ Р±Р°РЬРєР");
            System.out.println("3 – РЎРѕР·РґР°РЅРёРµ СЃС‡РµС‚Р° РІ
    Р±Р°РЬРєРµ");
            System.out.println("4 – РЎРЬС‚РёРµ РґРµРЬРіРё СЃРѕ
    СЃС‡РµС‚Р°");
            System.out.println("5 – РџРѕРїРѕР»РЬРµРЬРёРµ СЃС‡РµС‚Р°");
            System.out.println("6 – РџРµСЂРµРІРѕРґ РґРµРЬРіРё");
            System.out.println("7 – РћС‚РјРЬРЬ РїРѕРїРѕР»РЬРµРЬРёС");
```

```
42              System.out.println("8 – РћС,РјРµРSР° СЃРSСЏС,РёСЏ");
43              System.out.println("9 – РћС,РјРµРSР° РїРµСЂРµРІРsРґР°");
44              System.out.println("10 – РџСЂРsРjРsС,РєР° РІСЂРµРjРµРSРё");
45              System.out.println("exit – program finish");
46              str = in.nextLine();
47              switch (str)
48              {
49                  case "1":
50                      PercentOfDeposit percent = null;
51                      for (int j = 0; j < 2; j++)
52                      {
53                          System.out.println("Р'РІРµРґРёС,Рµ СЃСґРjРjСґ РёРїСЂРsС†РµРSС,");
54                          percent.addPercentAndSum(in.nextInt(), in.nextDouble());
55                      }
56                      System.out.println("Р'РІРµРґРёС,Рµ РїР°СЂР°РjРµС,СЂС<Р±Р°РSРєР°");
57
58                      bank = new Bank(in.nextLine(), in.nextDouble(), in.nextDouble(), in.nextDouble(), in.nextDouble(), percent);
59                      central.registerBank(bank);
60                      break;
61                  case "2":
62                      bank.addClientInBank(client);
63                      break;
64                  case "3":
65                      switch (in.nextLine())
66                      {
67                          case "1":
68                              System.out.println("Р'РІРµРґРёС,Рµ СЃСґРjРjСґ РёСЃРsРє");
69                              central.creationAccount(bank, in.nextInt(), client, in.nextInt(), "Credit");
70                              break;
71                          case "2":
72                              System.out.println("Р'РІРµРґРёС,Рµ СЃСґРjРjСґ РёСЃРsРє");
73                              central.creationAccount(bank, in.nextInt(), client, in.nextInt(), "Debit");
74                              break;
75                          case "3":
76                              System.out.println("Р'РІРµРґРёС,Рµ СЃСґРjРjСґ РёСЃРsРє");
77                              central.creationAccount(bank, in.nextInt(), client, in.nextInt(), "Deposit");
78                              break;
79                          default:
80                              throw new BanksException("Invalid option");
```

```
81                      }
82
83                      break;
84                  case "4":
85                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P°");
86                      central.withdrawalMoney(bank, in.nextInt(), in.
    nextInt(), client);
87                      break;
88                  case "5":
89                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P°");
90                      central.refillMoneyOn(bank, in.nextInt(), in.
    nextInt());
91                      break;
92                  case "6":
93                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ,
    PSPsPjPµCЂ CЃC‡PµC,P° PsC,PїCЂP°PIPëC,PµP»CЏ Pë PSPsPjPµCЂ CЃC‡PµC,P°
    PїPsP»CґC‡P°C,PµP»CЏ");
94                      central.transferMoneyOnBalance(bank, in.nextInt(),
     in.nextInt(), bank, in.nextInt(), client);
95                      break;
96                  case "7":
97                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P° ");
98                      central.cancelRefill(bank, in.nextInt(), in.
    nextInt());
99                      break;
100                 case "8":
101                     System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P° ");
102                     central.cancelWithdrawal(bank, in.nextInt(), in.
    nextInt());
103                     break;
104                 case "9":
105                     System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ,
    PSPsPjPµCЂ CЃC‡PµC,P° PsC,PїCЂP°PIPëC,PµP»CЏ Pë PSPsPjPµCЂ CЃC‡PµC,P°
    PїPsP»CґC‡P°C,PµP»CЏ");
106                     central.cancelTransfer(bank, in.nextInt(), in.
    nextInt(), bank, in.nextInt());
107                     break;
108                 case "10":
109                     System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P° ");
110                     central.checkTime( in.nextInt(), bank.getAccountId
    ( in.nextInt()));
111                     break;
112
113                 case "exit":
114                     System.out.println("finish");
```

```
115                         break;
116                     }
117                 }
118             }
119 }
```

Листинг 1.2: Bank.java

```
1  package models;
2
3  import services.EventManager;
4  import services.IAccount;
5  import tools.BanksException;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 public class Bank {
11     public EventManager eventManager;
12     private List<Client> clients;
13     private List<IAccount> accountsOfClient;
14     public PercentOfDeposit PercentsOfDeposit;
15     public String nameOfBank;
16     public double commissionOfCreditOfBank;
17     public double limitOfCreditOfBank;
18     public double percentOfDebitOfBank;
19     public double limitationOfBank;
20
21     public Bank(String name, double commissionOfCredit, double
    limitOfCredit, double percentOfDebit, double limitation,
22                 PercentOfDeposit percentOfDeposit) throws
    BanksException {
23         if (percentOfDeposit == null) throw new BanksException("
    Invalid percents of deposit");
24         PercentsOfDeposit = percentOfDeposit;
25         if (name == null) throw new BanksException("Invalid name");
26         nameOfBank = name;
27         commissionOfCreditOfBank = commissionOfCredit;
28         limitOfCreditOfBank = limitOfCredit;
29         percentOfDebitOfBank = percentOfDebit;
30         clients = new ArrayList<>();
31         limitationOfBank = limitation;
32         accountsOfClient = new ArrayList<>();
33         eventManager = new EventManager("Change nameOfBank",
34                 "Change commissionOfCreditOfBank", "Change
    commissionOfCreditOfBank",
35                 "Change limitOfCreditOfBank", "Change
    percentOfDebitOfBank", "Change limitationOfBank");
36     }
37
38     public void addClientInBank(Client client) throws BanksException {
39         if (client == null) throw new BanksException("Invalid client")
    ;
40         if (clients.stream().anyMatch(n -> n.nameOfClient == client.
    nameOfClient) &&
41                 clients.stream().anyMatch(n -> n.surnameOfBank ==
    client.surnameOfBank))
```

```
42              throw new BanksException("Client already in use");
43          clients.add(client);
44      }
45
46      public void setNameOfBank(String name) {
47           this.nameOfBank = name;
48           eventManager.notify("Change nameOfBank");
49      }
50      public String getNameOfBank() {
51          return this.nameOfBank;
52      }
53
54      public void setCommissionOfCredit(double commissionOfCredit)
55      {
56          this.commissionOfCreditOfBank = commissionOfCredit;
57          eventManager.notify("Change commissionOfCreditOfBank");
58      }
59      public double getCommissionOfCredit()
60      {
61          return this.commissionOfCreditOfBank;
62      }
63
64      public void setLimitOfCredit(double limitOfCredit)
65      {
66          this.limitOfCreditOfBank = limitOfCredit;
67          eventManager.notify("Change limitOfCreditOfBank");
68      }
69      public double getLimitOfCredit()
70      {
71          return this.limitOfCreditOfBank;
72      }
73
74      public void setPercentOfDebit(double percentOfDebit)
75      {
76          this.percentOfDebitOfBank = percentOfDebit;
77          eventManager.notify("Change percentOfDebitOfBank");
78      }
79      public double getPercentOfDebit()
80      {
81          return this.percentOfDebitOfBank;
82      }
83
84      public void setLimitation(double limitation)
85      {
86          this.limitationOfBank = limitation;
87          eventManager.notify("Change limitationOfBank");
88      }
89      public double getLimitation()
90      {
91          return this.limitationOfBank;
```

```
92      }
93
94      public void addDepositInIAccount(Deposit deposit) throws
    BanksException {
95          if (deposit == null) throw new BanksException("Invalid deposit
    ");
96          accountsOfClient.add(deposit);
97      }
98
99      public void addDebitInIAccount(Debit debit) throws BanksException
    {
100         if (debit == null) throw new BanksException("Invalid debit");
101         accountsOfClient.add(debit);
102     }
103
104     public void addCreditInIAccount(Credit credit) throws
    BanksException {
105         if (credit == null) throw new BanksException("Invalid credit")
    ;
106         accountsOfClient.add(credit);
107     }
108
109     public boolean isOperationInAccount(int sum) {
110         return accountsOfClient.stream().allMatch(x -> x.withdraw(sum)
    );
111     }
112
113     public IAccount getAccountId(int id) {
114         IAccount account = null;
115         for (int i = 0; i < accountsOfClient.size(); i++) {
116             if (accountsOfClient.remove(i).getAccountId() == id)
    account = accountsOfClient.remove(i);
117         }
118         return account;
119     }
120
121     public int countClients() {
122         return clients.size();
123     }
124
125     public int countAccounts() {
126         return accountsOfClient.size();
127     }
128 }
```

Листинг 1.3: CentralBank.java

```
1  package models;
2
3  import services.IAccount;
4  import tools.BanksException;
5
6  import java.util.ArrayList;
7  import java.util.List;
8
9  public class CentralBank
10 {
11     private static int countOfId = 0;
12     private List<Bank> banks;
13
14     public CentralBank() {
15
16         banks = new ArrayList<>();
17     }
18
19     public void registerBank(Bank bank) throws BanksException {
20         if (bank == null) throw new BanksException("Invalid Bank");
21         banks.add(bank);
22     }
23
24     public IAccount creationAccount(Bank bank, int balance, Client
    client, int time, String option) throws BanksException {
25         if (option == null) throw new BanksException("Invalid option")
    ;
26         if (client == null) throw new BanksException("Invalid client")
    ;
27         if (bank == null) throw new BanksException("Invalid Bank");
28         if (banks.stream().anyMatch(b -> b == bank)) return
    creationAccountForClient(bank, client, balance, time, option);
29         else throw new BanksException("Invalid bank");
30     }
31
32     public void checkTime(int time, IAccount account) throws
    BanksException {
33         if (account == null) throw new BanksException("Invalid account
    ");
34         account.benefitPay(time);
35     }
36
37     public IAccount creationAccountForClient(Bank bank, Client client,
     int balance, int time, String option) throws BanksException {
38         switch (option)
39         {
40             case "Credit":
41
42                 Credit credit = new Credit(countOfId++, bank.
```

```
    commissionOfCreditOfBank, bank.limitOfCreditOfBank, balance);
43                  bank.addCreditInIAccount(credit);
44                  return credit;
45
46              case "Debit":
47
48                  Debit debit = new Debit(balance, bank.
    percentOfDebitOfBank, countOfId++);
49                  bank.addDebitInIAccount(debit);
50                  return debit;
51
52              case "Deposit":
53
54                  double depositPercent = 0;
55                  int f = 0;
56                  for ( Integer key : bank.PercentsOfDeposit.percents.
    keySet() )  {
57                  f++;
58                  if (balance < key) {
59                      depositPercent = bank.PercentsOfDeposit.percents.
    get(key);
60                  }
61                  if( bank.PercentsOfDeposit.percents.keySet().size() -
    f == 1 && depositPercent == 0)
62                      depositPercent = bank.PercentsOfDeposit.percents.
    get(key);
63              }
64                  Deposit deposit = new Deposit(countOfId++, balance,
    time, depositPercent);
65              bank.addDepositInIAccount(deposit);
66              return deposit;
67              default:
68                  throw new BanksException("Option not found");
69          }
70      }
71
72      public void refillMoneyOn(Bank bank, int balance, int id) throws
    BanksException {
73          if (bank == null) throw new BanksException("Invalid Bank");
74          bank.getAccountId(id).refillOperation(balance);
75      }
76
77      public void withdrawalMoney(Bank bank, int balance, int id, Client
        client) throws BanksException {
78          if (!checkClientPassportAndAddress(client) && balance > bank.
    limitationOfBank && bank.isOperationInAccount(balance)) {
79              throw new BanksException("Invalid Withdrawal");
80          }
81          bank.getAccountId(id).withdrawalOperation(balance);
82      }
```

```
83
84      public void transferMoneyOnBalance(Bank bank1, int balance, int
    id1, Bank bank2, int id2, Client client) throws BanksException {
85          if (!checkClientPassportAndAddress(client) && balance > bank1.
    limitationOfBank && bank1.isOperationInAccount(balance)) {
86              throw new BanksException("Invalid Transfer");
87          }
88          bank1.getAccountId(id1).transferOperation(bank2.getAccountId(
    id2), balance);
89      }
90
91      public void cancelRefill(Bank bank, int balance, int id) throws
    BanksException {
92          IAccount operation = bank.getAccountId(id);
93          if (operation == null) throw new BanksException("Refill don't
    exists");
94          operation.cancelRefillOperation(balance);
95      }
96
97      public void cancelTransfer(Bank bank1, int balance, int id1, Bank
    bank2, int id2) throws BanksException {
98          IAccount operation = bank1.getAccountId(id1);
99          if (operation == null) throw new BanksException("Transfer don'
    t exists");
100         operation.cancelTransferOperation(bank2.getAccountId(id2),
    balance);
101     }
102
103     public void cancelWithdrawal(Bank bank, int balance, int id)
    throws BanksException {
104         IAccount operation = bank.getAccountId(id);
105         if (operation == null) throw new BanksException("Withdrawal
    don't exists");
106         operation.cancelWithdrawalOperation(balance);
107     }
108
109     private boolean checkClientPassportAndAddress(Client client) {
110         return !client.passportOfBank.isBlank() || !client.
    passportOfBank.isBlank();
111     }
112     public int countBanks(){
113         return banks.size();
114     }
115 }
```

Листинг 1.4: Client.java

```
package models;

import tools.BanksException;

public class Client {
    private static int _id = 0;
    public int Id = 0;
    public String nameOfClient;
    public String surnameOfBank;
    public String addressOfBank;
    public String passportOfBank;

    public Client(String name, String surname, String address, String
   passport) throws BanksException {
        Id = _id++;
        if(name == null) {
            throw new BanksException("Invalid name");
        }
        nameOfClient = name;
        if(surname == null) {
            throw new BanksException("Invalid surname");
        }
        surnameOfBank = surname;
        addressOfBank = address;
        passportOfBank = passport;
    }
    public static void update(String event){
        System.out.print(event);
    }
}
```

Листинг 1.5: Credit.java

```
1  package models;
2
3  import services.IAccount;
4
5  public class Credit implements IAccount {
6      private double commissionTmp = 0;
7      public double commissionOfCredit;
8      public double limitOfBank;
9      public int idOfBank;
10     public int balanceOfBank;
11
12     public Credit(int id, double commission, double limit, int balance
       )
13     {
14         commissionOfCredit = commission;
15         limitOfBank = limit;
16         idOfBank = id;
17         balanceOfBank = balance;
18     }
19
20     public void withdrawalOperation(int sum) {
21
22         balanceOfBank -=sum;
23     }
24
25     public void cancelWithdrawalOperation(int sum) {
26
27         balanceOfBank +=sum;
28     }
29
30     public void refillOperation(int sum) {
31
32         balanceOfBank +=sum;
33     }
34
35     public void cancelRefillOperation(int sum) {
36
37         balanceOfBank -=sum;
38     }
39
40     public void transferOperation(IAccount other, int sum) {
41         balanceOfBank -=sum;
42         other.refillOperation(sum);
43     }
44
45     public void cancelTransferOperation(IAccount other, int sum) {
46         balanceOfBank +=sum;
47         other.withdrawalOperation(sum);
48     }
```

```
49
50      public void benefitPay(int time) {
51          commissionTmp += balanceOfBank * commissionOfCredit /300;
52          balanceOfBank -=(int) commissionTmp *time;
53          commissionTmp =0;
54      }
55
56      public boolean withdraw(int sum) {
57          return Math.abs(balanceOfBank -sum)< limitOfBank;
58      }
59
60      public int getAccountId(){
61
62          return idOfBank;
63      }
64
65      public int checkBalance() {
66
67          return balanceOfBank;
68      }
69  }
```

Листинг 1.6: Debit.java

```
1  package models;
2
3  import services.IAccount;
4
5  public class Debit implements IAccount {
6      private double benefit = 0;
7      public double Percent;
8      public int Balance;
9      public int Id;
10
11     public Debit(int balance, double percent, int id) {
12         Percent = percent;
13         Balance = balance;
14         Id = id;
15     }
16
17
18     public void withdrawalOperation(int sum) {
19
20         Balance -= sum;
21     }
22
23     public void cancelWithdrawalOperation(int sum) {
24
25         Balance += sum;
26     }
27
28     public void refillOperation(int sum) {
29
30         Balance += sum;
31     }
32
33     public void cancelRefillOperation(int sum) {
34
35         Balance -= sum;
36     }
37
38     public void transferOperation(IAccount account2, int sum) {
39         Balance -= sum;
40         account2.refillOperation(sum);
41     }
42
43     public void cancelTransferOperation(IAccount other, int sum) {
44         Balance += sum;
45         other.withdrawalOperation(sum);
46     }
47
48     public void benefitPay(int time) {
49         benefit = Balance * Percent / 300;
```

```
50          Balance += (int) benefit * time;
51          benefit = 0;
52      }
53
54      public boolean withdraw(int sum) {
55
56          return Balance >= sum;
57      }
58
59      public int getAccountId() {
60
61          return Id;
62      }
63
64      public int checkBalance() {
65
66          return Balance;
67      }
68  }
```

Листинг 1.7: Deposit.java

```
1  package models;
2
3
4  import services.IAccount;
5
6  public class Deposit implements IAccount {
7      private double benefit = 0;
8      public int idOfDeposit;
9      public int timeOfDeposit;
10     public double percentOfDeposit;
11     public int balanceOfDeposit;
12
13     public Deposit(int id, int balance, int time, double percent) {
14         idOfDeposit = id;
15         timeOfDeposit = time;
16         percentOfDeposit = percent;
17         balanceOfDeposit = balance;
18     }
19
20
21     public void withdrawalOperation(int sum) {
22
23         balanceOfDeposit -= sum;
24     }
25
26     public void cancelWithdrawalOperation(int sum) {
27
28         balanceOfDeposit += sum;
29     }
30
31     public void refillOperation(int sum) {
32
33         balanceOfDeposit += sum;
34     }
35
36     public void cancelRefillOperation(int sum) {
37
38         balanceOfDeposit -= sum;
39     }
40
41     public void transferOperation(IAccount other, int sum) {
42         balanceOfDeposit -= sum;
43         other.refillOperation(sum);
44     }
45
46     public void cancelTransferOperation(IAccount other, int sum) {
47         balanceOfDeposit += sum;
48         other.withdrawalOperation(sum);
49     }
```

```
50
51      public void benefitPay(int time) {
52          benefit = balanceOfDeposit * percentOfDeposit / 300;
53          balanceOfDeposit += (int) benefit * time;
54          benefit = 0;
55      }
56
57      public boolean withdraw(int sum) {
58
59          return balanceOfDeposit >= sum && timeOfDeposit == 0;
60      }
61
62      public int getAccountId() {
63
64          return idOfDeposit;
65      }
66
67      public int checkBalance() {
68
69          return balanceOfDeposit;
70      }
71  }
```

Листинг 1.8: PercentOfDeposit.java

```
1  package models;
2
3  import java.util.*;
4
5  public class PercentOfDeposit
6  {
7      public Map<Integer, Double> percents;
8      public PercentOfDeposit() {
9
10         percents = new HashMap<>();
11     }
12
13     public void addPercentAndSum(int sum, double percent)
14     {
15         percents.put(sum,percent);
16     }
17 }
```

Листинг 1.9: ClientBuilder.java

```
package services;

import models.Client;
import tools.BanksException;

public class ClientBuilder
{
    private String name;
    private String passport;
    private String surname;
    private String address;

    public ClientBuilder changeName(String name) throws BanksException
    {
        if(name == null) throw new BanksException("Invalid name");
        this.name = name;
        return this;
    }

    public ClientBuilder changeSurname(String surname) throws
    BanksException {
        if(surname == null) throw new BanksException("Invalid surname"
    );
        this.surname = surname;
        return this;
    }

    public ClientBuilder changePassport(String passport) throws
    BanksException {
        if(passport == null) throw new BanksException("Invalid
    passport");
        this.passport = passport;
        return this;
    }

    public ClientBuilder changeAddress(String address) throws
    BanksException {
        if(address == null) throw new BanksException("Invalid address"
    );
        this.address = address;
        return this;
    }

    public Client create() throws BanksException {
        return new Client(name, surname, address, passport);
    }
}
```

Листинг 1.10: EventManager.java

```
package services;


import models.Client;

import java.util.*;

public class EventManager {
    Map<String, List<Client>> listeners = new HashMap<>();

    public EventManager(String... operations) {
        for (String operation : operations) {
            this.listeners.put(operation, new ArrayList<>());
        }
    }

    public void subscribe(String eventType, Client listener) {
        List<Client> users = listeners.get(eventType);
        users.add(listener);
    }

    public void unsubscribe(String eventType, Client listener) {
        List<Client> users = listeners.get(eventType);
        users.remove(listener);
    }

    public void notify(String event) {
        List<Client> users = listeners.get(event);
        for (Client listener : users) {
            listener.update(event);
        }
    }
}
```

Листинг 1.11: IAccount.java

```
1  package services;
2
3  public interface IAccount
4  {
5      public void withdrawalOperation(int sum);
6      public void cancelWithdrawalOperation(int sum);
7      public void refillOperation(int sum);
8      public void cancelRefillOperation(int sum);
9      public void transferOperation(IAccount other, int sum);
10     public void cancelTransferOperation(IAccount other, int sum);
11     public void benefitPay(int time);
12     public boolean withdraw(int sum);
13     public int getAccountId();
14     public int checkBalance();
15 }
```

Листинг 1.12: BanksException.java

```
1 package tools;
2
3 public class BanksException extends Exception {
4     public BanksException(String message) {
5         super(message);
6     }
7 }
```

Листинг 1.13: Application.java

```
1  package com.kisssusha;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class Application {
8      public static void main(String[] args) {
9          SpringApplication.run(Application.class, args);
10     }
11 }
```

Листинг 1.14: SecurityConfiguration.java

```
1  package com.kisssusha.configuration;
2
3  import com.kisssusha.service.SecurityService;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.context.annotation.Bean;
6  import org.springframework.security.authentication.dao.
       DaoAuthenticationProvider;
7  import org.springframework.security.config.annotation.web.builders.
       HttpSecurity;
8  import org.springframework.security.config.annotation.web.
       configuration.EnableWebSecurity;
9  import org.springframework.security.config.annotation.web.
       configuration.WebSecurityConfigurerAdapter;
10 import org.springframework.security.crypto.bcrypt.
       BCryptPasswordEncoder;
11 import org.springframework.security.crypto.password.PasswordEncoder;
12 import org.springframework.stereotype.Component;
13
14 @EnableWebSecurity
15 @Component
16 public class SecurityConfiguration extends
       WebSecurityConfigurerAdapter {
17
18     @Autowired
19     SecurityService service;
20
21     protected void configure(HttpSecurity http) throws Exception {
22         http.httpBasic()
23                 .and()
24                 .authorizeRequests()
25                 .antMatchers("/admin/").hasRole("ADMIN")
26                 .antMatchers("/user/").authenticated()
27                 .and()
28                 .logout().logoutSuccessUrl("/")
29                 .and()
30                 .csrf().disable()
31                 .formLogin();
32     }
33
34     @Bean
35     public DaoAuthenticationProvider daoAuthenticationProvider() {
36         DaoAuthenticationProvider authenticationProvider = new
       DaoAuthenticationProvider();
37         authenticationProvider.setPasswordEncoder(passwordEncoder());
38         authenticationProvider.setUserDetailsService(service);
39         return authenticationProvider;
40     }
41
42     private PasswordEncoder passwordEncoder() {
```

```
43            return new BCryptPasswordEncoder();
44        }
45    }
```

Листинг 1.15: AdminController.java

```
1  package com.kisssusha.controller;
2
3  import com.kisssusha.DAO.dto.CatsDto;
4  import com.kisssusha.DAO.dto.FriendshipDto;
5  import com.kisssusha.DAO.dto.OwnersDto;
6  import com.kisssusha.DAO.dto.ShelterDto;
7  import com.kisssusha.service.KotikiService;
8  import com.kisssusha.service.tools.KotikiException;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.web.bind.annotation.*;
11
12 import java.util.List;
13
14 @RestController
15 @RequestMapping("/admin")
16 public class AdminController {
17     @Autowired
18     KotikiService service;
19
20     @GetMapping("/findAllOwner")
21     public List<OwnersDto> findAllOwner() {
22         return service.getOwners();
23     }
24
25     @PostMapping("/addOwner")
26     public boolean addOwner(@RequestBody OwnersDto ownersDto) {
27         return service.addOwner(ownersDto);
28     }
29
30     @DeleteMapping("/deleteOwner/{id}")
31     public boolean deleteOwner(@PathVariable("id") Long id) {
32         return service.deleteOwner(id);
33     }
34
35     @DeleteMapping("/break-shelter")
36     public boolean breakShelter(@RequestBody ShelterDto shelterDto)
   throws KotikiException {
37         return service.breakShelter(shelterDto.getIdCat(), shelterDto.
   getIdOwner());
38     }
39
40     @PostMapping("/make-shelter")
41     public boolean makeShelter(@RequestBody ShelterDto shelterDto) {
42         return service.makeShelter(shelterDto);
43     }
44
45     @GetMapping("/find-all-cats")
46     public List<CatsDto> findAllCats() {
47         return service.getCats();
```

```
48      }
49
50      @PostMapping("/add-cat")
51      public boolean addCat(@RequestBody CatsDto catsDto) {
52          return service.addCat(catsDto);
53      }
54
55      @DeleteMapping("/delete-cat/{id}")
56      public boolean deleteCat(@PathVariable("id") Long id) {
57          return service.deleteCat(id);
58      }
59
60      @DeleteMapping("/break-friendship")
61      public boolean breakFriendship(@RequestBody FriendshipDto
    friendshipDto) {
62          return service.breakFriendship(friendshipDto.getIdCat(),
    friendshipDto.getIdFriend());
63      }
64      @PostMapping("/make-friendship")
65      public boolean makeFriendship(@RequestBody FriendshipDto
    friendshipDto){
66          return service.makeFriendship(friendshipDto);
67      }
68
69  }
```

Листинг 1.16: OwnerController.java

```
1  package com.kisssusha.controller;
2
3  import com.kisssusha.DAO.dto.CatsDto;
4  import com.kisssusha.DAO.dto.FriendshipDto;
5  import com.kisssusha.DAO.dto.ShelterDto;
6  import com.kisssusha.service.KotikiService;
7  import com.kisssusha.service.tools.KotikiException;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.web.bind.annotation.*;
10
11 import java.util.List;
12
13 @RestController
14 @RequestMapping("/owner")
15 public class OwnerController {
16     @Autowired
17     KotikiService service;
18
19     @DeleteMapping("/break-shelter")
20     public boolean breakShelter(@RequestBody ShelterDto shelterDto)
       throws KotikiException {
21         return service.breakShelter(shelterDto.getIdCat(), shelterDto.
       getIdOwner());
22     }
23
24     @PostMapping("/make-shelter")
25     public boolean makeShelter(@RequestBody ShelterDto shelterDto) {
26         return service.makeShelter(shelterDto);
27     }
28
29     @GetMapping("/find-all-cats")
30     public List<CatsDto> findAllCats() {
31         return service.getCats();
32     }
33
34     @PostMapping("/add-cat")
35     public boolean addCat(@RequestBody CatsDto catsDto) {
36         return service.addCat(catsDto);
37     }
38
39     @DeleteMapping("/delete-cat/{id}")
40     public boolean deleteCat(@PathVariable("id") Long id) {
41         return service.deleteCat(id);
42     }
43
44     @DeleteMapping("/break-friendship")
45     public boolean breakFriendship(@RequestBody FriendshipDto
       friendshipDto) {
46         return service.breakFriendship(friendshipDto.getIdCat(),
```

```
    friendshipDto.getIdFriend());
47      }
48      @PostMapping("/make-friendship")
49      public boolean makeFriendship(@RequestBody FriendshipDto
    friendshipDto){
50          return service.makeFriendship(friendshipDto);
51      }
52  }
```

Листинг 1.17: ServiceController.java

```
package com.kisssusha.controller;

import com.kisssusha.DAO.dto.OwnersDto;
import com.kisssusha.service.KotikiService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ServiceController {

    @Autowired
    KotikiService service;

    @GetMapping("")
    public String startPage(){
        return "Start page";
    }

    @PostMapping("/registration")
    public String registration(@RequestBody OwnersDto owner) {
        service.addOwner(owner);
        return "Successful registration";
    }
}
```

Листинг 1.18: CatsDto.java

```
1  package com.kisssusha.DAO.dto;
2
3  import com.kisssusha.DAO.enums.MyColors;
4  import com.kisssusha.DAO.models.Cats;
5
6  import java.sql.Timestamp;
7  import java.util.Objects;
8
9  public class CatsDto {
10     private Long id;
11     private String name;
12     private Timestamp birth;
13     private String breed;
14     private MyColors color;
15
16     public CatsDto() {
17     }
18
19     public Long getId() {
20         return id;
21     }
22
23     public void setId(Long id) {
24         this.id = id;
25     }
26
27     public String getName() {
28         return name;
29     }
30
31     public void setName(String name) {
32         this.name = name;
33     }
34
35     public Timestamp getBirth() {
36         return birth;
37     }
38
39     public void setBirth(Timestamp birth) {
40         this.birth = birth;
41     }
42
43     public String getBreed() {
44         return breed;
45     }
46
47     public void setBreed(String breed) {
48         this.breed = breed;
49     }
```

```
50
51      public MyColors getColor() {
52          return color;
53      }
54
55      public void setColor(MyColors color) {
56          this.color = color;
57      }
58
59      public CatsDto(Cats cat) {
60          this.id = cat.getId();
61          this.birth = cat.getBirth();
62          this.breed = cat.getBreed();
63          this.name = cat.getName();
64          this.color = cat.getColor();
65      }
66
67      @Override
68      public boolean equals(Object o) {
69          if (this == o) return true;
70          if (o == null || getClass() != o.getClass()) return false;
71          CatsDto catDTO = (CatsDto) o;
72          return Objects.equals(id, catDTO.id) && color == catDTO.color
    && Objects.equals(name, catDTO.name)
73                  && Objects.equals(breed, catDTO.breed)
74                  && Objects.equals(birth, catDTO.birth);
75      }
76
77      @Override
78      public int hashCode() {
79          return Objects.hash(id, color, name, breed, birth);
80      }
81  }
```

Листинг 1.19: FriendshipDto.java

```
package com.kisssusha.DAO.dto;

import com.kisssusha.DAO.models.Friendship;

import java.util.Objects;

public class FriendshipDto {
    private Long id;
    private Long idFriend;
    private Long idCat;

    public FriendshipDto() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getIdFriend() {
        return idFriend;
    }

    public void setIdFriend(Long idFriend) {
        this.idFriend = idFriend;
    }

    public Long getIdCat() {
        return idCat;
    }

    public void setIdCat(Long idCat) {
        this.idCat = idCat;
    }

    public FriendshipDto(Friendship friends) {
        this.id = friends.getId();
        this.idFriend = friends.getIdFriend();
        this.idCat = friends.getIdCat();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        FriendshipDto that = (FriendshipDto) o;
```

```
50          return id.equals(that.id) && idFriend.equals(that.idFriend) &&
        idCat.equals(that.idCat);
51      }
52
53      @Override
54      public int hashCode() {
55          return Objects.hash(id, idFriend, idCat);
56      }
57
58
59  }
```

Листинг 1.20: OwnersDto.java

```
package com.kisssusha.DAO.dto;

import com.kisssusha.DAO.models.Owners;

import java.sql.Timestamp;
import java.util.Objects;

public class OwnersDto {
    private Long id;
    private String name;
    private Timestamp date;
    private String login;
    private String password;
    private String role;

    public OwnersDto(Owners owners){
        this.id = owners.getId();
        this.date = owners.getDate();
        this.name = owners.getName();
        this.login = owners.getLogin();
        this.password = owners.getPassword();
        this.role = owners.getRole();
    }

    public OwnersDto(){
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Timestamp getDate() {
        return date;
    }

    public void setDate(Timestamp date) {
        this.date = date;
```

```
50      }
51
52      public String getLogin() {
53          return login;
54      }
55
56      public void setLogin(String login) {
57          this.login = login;
58      }
59
60      public String getPassword() {
61          return password;
62      }
63
64      public void setPassword(String password) {
65          this.password = password;
66      }
67
68      public String getRole() {
69          return role;
70      }
71
72      public void setRole(String role) {
73          this.role = role;
74      }
75
76      @Override
77      public boolean equals(Object o) {
78          if (this == o) return true;
79          if (o == null || getClass() != o.getClass()) return false;
80          OwnersDto ownersDto = (OwnersDto) o;
81          return Objects.equals(id, ownersDto.id) && Objects.equals(name
    , ownersDto.name) && Objects.equals(date, ownersDto.date) &&
    Objects.equals(login, ownersDto.login) && Objects.equals(password,
    ownersDto.password) && Objects.equals(role, ownersDto.role);
82      }
83
84      @Override
85      public int hashCode() {
86          return Objects.hash(id, name, date, login, password, role);
87      }
88  }
```

Листинг 1.21: ShelterDto.java

```
1  package com.kisssusha.DAO.dto;
2
3  import com.kisssusha.DAO.models.Shelter;
4
5  import java.util.Objects;
6
7  public class ShelterDto {
8      private Long id;
9      private Long idCat;
10     private Long idOwner;
11     public ShelterDto(){
12     }
13
14     public Long getId() {
15         return id;
16     }
17
18     public void setId(Long id) {
19         this.id = id;
20     }
21
22     public Long getIdCat() {
23         return idCat;
24     }
25
26     public void setIdCat(Long idCat) {
27         this.idCat = idCat;
28     }
29
30     public Long getIdOwner() {
31         return idOwner;
32     }
33
34     public void setIdOwner(Long idOwner) {
35         this.idOwner = idOwner;
36     }
37
38     public ShelterDto(Shelter shelter){
39         this.id = shelter.getId();
40         this.idCat = shelter.getIdCat();
41         this.idOwner = shelter.getIdOwner();
42     }
43
44     @Override
45     public boolean equals(Object o) {
46         if (this == o) return true;
47         if (o == null || getClass() != o.getClass()) return false;
48         ShelterDto that = (ShelterDto) o;
49         return Objects.equals(id, that.id) && Objects.equals(idCat,
```

```
    that.idCat)
50                  && Objects.equals(idOwner, that.idOwner);
51      }
52
53      @Override
54      public int hashCode() {
55          return Objects.hash(id, idCat, idOwner);
56      }
57  }
```

Листинг 1.22: MyColors.java

```
1 package com.kisssusha.DAO.enums;
2
3 public enum MyColors {
4     White,
5     Brown,
6     Black,
7     Orange
8 }
```

Листинг 1.23: CatsDao.java

```
1 package com.kisssusha.DAO.implemetations;
2
3 import com.kisssusha.DAO.models.Cats;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface CatsDao extends JpaRepository<Cats, Long> {
9 }
```

Листинг 1.24: FriendshipDao.java

```
1  package com.kisssusha.DAO.implemetations;
2
3  import com.kisssusha.DAO.models.Friendship;
4
5  import org.springframework.data.jpa.repository.JpaRepository;
6  import org.springframework.stereotype.Repository;
7
8  @Repository
9  public interface FriendshipDao extends JpaRepository<Friendship, Long>
       {
10     void deleteAllByIdCatOrIdFriend(Long idCat, Long idFriend);
11     void deleteAllByIdCatAndIdFriend(Long idCat, Long idFriend);
12 }
```

Листинг 1.25: OwnersDao.java

```
1  package com.kisssusha.DAO.implemetations;
2
3  import com.kisssusha.DAO.models.Owners;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface OwnersDao extends JpaRepository<Owners, Long> {
9      Owners findByLogin(String login);
10 }
```

Листинг 1.26: ShelterDao.java

```
1  package com.kisssusha.DAO.implemetations;
2
3  import com.kisssusha.DAO.models.Shelter;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface ShelterDao extends JpaRepository<Shelter, Long> {
9      void deleteAllByIdCat(Long catId);
10     void deleteAllByIdOwner(Long ownerId);
11     void deleteAllByIdOwnerAndIdCat(Long ownerId, Long catId);
12 }
```

Листинг 1.27: Cats.java

```
package com.kisssusha.DAO.models;

import com.kisssusha.DAO.dto.CatsDto;
import com.kisssusha.DAO.enums.MyColors;

import javax.persistence.*;
import java.sql.Timestamp;
import java.util.Objects;

@Entity
@Table(name = "cats", schema = "public", catalog = "postgres")
public class Cats {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private Long id;
    @Basic
    @Column(name = "name", nullable = true, length = -1)
    private String name;
    @Basic
    @Column(name = "birth", nullable = true)
    private Timestamp birth;
    @Basic
    @Column(name = "breed", nullable = true, length = -1)
    private String breed;
    @Basic
    @Column(name = "color", nullable = true, length = -1)
    @Enumerated(EnumType.STRING)
    private MyColors color;

    public Cats() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

```

```
50      public Timestamp getBirth() {
51          return birth;
52      }
53
54      public void setBirth(Timestamp birth) {
55          this.birth = birth;
56      }
57
58      public String getBreed() {
59          return breed;
60      }
61
62      public void setBreed(String breed) {
63          this.breed = breed;
64      }
65
66      public MyColors getColor() {
67          return color;
68      }
69
70      public void setColor(MyColors color) {
71          this.color = color;
72      }
73
74      public Cats(CatsDto cat) {
75          this.birth = cat.getBirth();
76          this.breed = cat.getBreed();
77          this.name = cat.getName();
78          this.color = cat.getColor();
79      }
80
81      @Override
82      public boolean equals(Object o) {
83          if (this == o) return true;
84          if (o == null || getClass() != o.getClass()) return false;
85          Cats that = (Cats) o;
86          return Objects.equals(id, that.id) && Objects.equals(name,
    that.name) && Objects.equals(birth, that.birth)
87                  && Objects.equals(breed, that.breed) && Objects.equals
    (color, that.color);
88      }
89
90      @Override
91      public int hashCode() {
92          return Objects.hash(id, name, birth, breed, color);
93      }
94  }
```

Листинг 1.28: Friendship.java

```
1  package com.kisssusha.DAO.models;
2
3  import com.kisssusha.DAO.dto.FriendshipDto;
4
5  import javax.persistence.*;
6  import java.util.Objects;
7
8  @Entity
9  @Table(name = "friend_for_cat", schema = "public", catalog = "postgres
       ")
10 public class Friendship {
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     @Id
13     @Column(name = "id", nullable = false)
14     private Long id;
15     @Basic
16     @Column(name = "first_cat_id", nullable = false)
17     private Long idFriend;
18     @Basic
19     @Column(name = "second_cat_id", nullable = false)
20     private Long idCat;
21
22     public Friendship() {
23     }
24
25     public Long getId() {
26         return id;
27     }
28
29     public void setId(Long id) {
30         this.id = id;
31     }
32
33     public Long getIdFriend() {
34         return idFriend;
35     }
36
37     public void setIdFriend(Long idFriend) {
38         this.idFriend = idFriend;
39     }
40
41     public Long getIdCat() {
42         return idCat;
43     }
44
45     public void setIdCat(Long idCat) {
46         this.idCat = idCat;
47     }
48
```

```
49      public Friendship(FriendshipDto friends) {
50          this.idCat = friends.getIdCat();
51          this.idFriend = friends.getIdFriend();
52      }
53
54      @Override
55      public boolean equals(Object o) {
56          if (this == o) return true;
57          if (o == null || getClass() != o.getClass()) return false;
58          Friendship that = (Friendship) o;
59          return Objects.equals(id, that.id) && Objects.equals(idFriend,
        that.idFriend)
60                  && Objects.equals(idCat, that.idCat);
61      }
62
63      @Override
64      public int hashCode() {
65          return Objects.hash(id, idFriend, idCat);
66      }
67  }
```

Листинг 1.29: Owners.java

```
package com.kisssusha.DAO.models;

import com.kisssusha.DAO.dto.OwnersDto;

import javax.persistence.*;
import java.sql.Timestamp;
import java.util.Objects;

@Entity
@Table(name = "owners", schema = "public", catalog = "postgres")
public class Owners {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private Long id;
    @Basic
    @Column(name = "name", nullable = true, length = -1)
    private String name;
    @Basic
    @Column(name = "date", nullable = true)
    private Timestamp date;
    @Basic
    @Column(name = "login", nullable = true, length = -1)
    private String login;
    @Basic
    @Column(name = "password", nullable = true, length = -1)
    private String password;
    @Basic
    @Column(name = "role", nullable = true, length = -1)
    private String role;

    public Owners(OwnersDto own) {
        this.date = own.getDate();
        this.name = own.getName();
        this.login = own.getLogin();
        this.password = own.getPassword();
        this.role = own.getRole();
    }

    public Owners() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
```

```
50
51      public String getName() {
52          return name;
53      }
54
55      public void setName(String name) {
56          this.name = name;
57      }
58
59      public Timestamp getDate() {
60          return date;
61      }
62
63      public void setDate(Timestamp date) {
64          this.date = date;
65      }
66
67      public String getLogin() {
68          return login;
69      }
70
71      public void setLogin(String login) {
72          this.login = login;
73      }
74
75      public String getPassword() {
76          return password;
77      }
78
79      public void setPassword(String password) {
80          this.password = password;
81      }
82
83      public String getRole() {
84          return role;
85      }
86
87      public void setRole(String role) {
88          this.role = role;
89      }
90
91      @Override
92      public boolean equals(Object o) {
93          if (this == o) return true;
94          if (o == null || getClass() != o.getClass()) return false;
95          Owners owners = (Owners) o;
96          return Objects.equals(id, owners.id) && Objects.equals(name,
    owners.name) && Objects.equals(date, owners.date) && Objects.equals
    (login, owners.login) && Objects.equals(password, owners.password)
    && Objects.equals(role, owners.role);
```

```
97      }
98
99      @Override
100     public int hashCode() {
101         return Objects.hash(id, name, date, login, password, role);
102     }
103 }
```

Листинг 1.30: Shelter.java

```
package com.kisssusha.DAO.models;

import com.kisssusha.DAO.dto.ShelterDto;

import javax.persistence.*;
import java.util.Objects;

@Entity
@Table(name = "shelter", schema = "public", catalog = "postgres")
public class Shelter {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private Long id;
    @Basic
    @Column(name = "cat_id", nullable = false)
    private Long idCat;
    @Basic
    @Column(name = "owner_id", nullable = false)
    private Long idOwner;

    public Shelter() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getIdCat() {
        return idCat;
    }

    public void setIdCat(Long idCat) {
        this.idCat = idCat;
    }

    public Long getIdOwner() {
        return idOwner;
    }

    public void setIdOwner(Long idOwner) {
        this.idOwner = idOwner;
    }

    public Shelter(ShelterDto shelter) {
```

```
50          this.idCat = shelter.getIdCat();
51          this.idOwner = shelter.getIdOwner();
52      }
53
54
55      @Override
56      public boolean equals(Object o) {
57          if (this == o) return true;
58          if (o == null || getClass() != o.getClass()) return false;
59          Shelter that = (Shelter) o;
60          return Objects.equals(id, that.id) && Objects.equals(idCat,
    that.idCat)
61                  && Objects.equals(idOwner, that.idOwner);
62      }
63
64      @Override
65      public int hashCode() {
66          return Objects.hash(id, idCat, idOwner);
67      }
68  }
```

Листинг 1.31: DaoException.java

```
1  package com.kisssusha.DAO.tools;
2
3  public class DaoException extends Exception {
4      public DaoException() {
5          super();
6      }
7
8      public DaoException(String message) {
9          super(message);
10     }
11
12     public DaoException(String message, Throwable cause) {
13         super(message, cause);
14
15     }
16 }
```

Листинг 1.32: KotikiService.java

```
1  package com.kisssusha.service;
2
3  import com.kisssusha.DAO.dto.CatsDto;
4  import com.kisssusha.DAO.dto.FriendshipDto;
5  import com.kisssusha.DAO.dto.OwnersDto;
6  import com.kisssusha.DAO.dto.ShelterDto;
7  import com.kisssusha.DAO.implemetations.CatsDao;
8  import com.kisssusha.DAO.implemetations.FriendshipDao;
9  import com.kisssusha.DAO.implemetations.OwnersDao;
10 import com.kisssusha.DAO.implemetations.ShelterDao;
11 import com.kisssusha.DAO.models.Cats;
12 import com.kisssusha.DAO.models.Friendship;
13 import com.kisssusha.DAO.models.Owners;
14 import com.kisssusha.DAO.models.Shelter;
15 import com.kisssusha.service.tools.KotikiException;
16 import org.springframework.beans.factory.annotation.Autowired;
17 import org.springframework.stereotype.Service;
18 import org.springframework.transaction.annotation.Transactional;
19
20 import java.util.List;
21 import java.util.stream.Collectors;
22
23 @Service
24 public class KotikiService {
25     @Autowired
26     CatsDao catDAO;
27     @Autowired
28     OwnersDao ownerDAO;
29     @Autowired
30     FriendshipDao friendshipDao;
31     @Autowired
32     ShelterDao shelterDao;
33
34     public List<CatsDto> getCats(){
35         return catDAO.findAll().stream().map(CatsDto::new).collect(
       Collectors.toList());
36     }
37
38     public boolean addCat(CatsDto catsDto){
39         catDAO.save(new Cats(catsDto));
40         return true;
41     }
42     public boolean makeShelter(ShelterDto shelterDto) {
43         shelterDao.save(new Shelter(shelterDto));
44         return true;
45     }
46
47     public boolean makeFriendship(FriendshipDto friendshipDto) {
48         friendshipDao.save(new Friendship(friendshipDto));
```

```
49          return true;
50      }
51      public boolean breakFriendship(Long firstCatId, Long secondCatId)
    {
52          friendshipDao.deleteAllByIdCatAndIdFriend(firstCatId,
    secondCatId);
53          return true;
54      }
55      public boolean breakShelter(Long catId, Long ownerId) throws
    KotikiException {
56          shelterDao.deleteAllByIdOwnerAndIdCat(ownerId, catId);
57          return true;
58      }
59
60
61      @Transactional
62      public boolean deleteCat(Long idCat){
63          shelterDao.deleteAllByIdCat(idCat);
64          friendshipDao.deleteAllByIdCatOrIdFriend(idCat, idCat);
65          catDAO.deleteById(idCat);
66          return true;
67      }
68      public List<OwnersDto> getOwners(){
69          return ownerDAO.findAll().stream().map(OwnersDto::new).collect
    (Collectors.toList());
70      }
71
72      public OwnersDto findOwnerByLogin(String login) {
73          return new OwnersDto(ownerDAO.findByLogin(login));
74      }
75
76      public boolean addOwner(OwnersDto ownersDto){
77          ownerDAO.save(new Owners(ownersDto));
78          return true;
79      }
80      @Transactional
81      public boolean deleteOwner(Long idOwner){
82          shelterDao.deleteAllByIdOwner(idOwner);
83          ownerDAO.deleteById(idOwner);
84          return true;
85      }
86  }
```

Листинг 1.33: SecurityService.java

```
1  package com.kisssusha.service;
2
3  import com.kisssusha.DAO.dto.OwnersDto;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.security.core.GrantedAuthority;
7  import org.springframework.security.core.authority.
       SimpleGrantedAuthority;
8  import org.springframework.security.core.userdetails.User;
9  import org.springframework.security.core.userdetails.UserDetails;
10 import org.springframework.security.core.userdetails.
       UserDetailsService;
11 import org.springframework.security.core.userdetails.
       UsernameNotFoundException;
12 import org.springframework.stereotype.Service;
13 import org.springframework.transaction.annotation.Transactional;
14
15
16 import java.util.Collection;
17 import java.util.List;
18 import java.util.stream.Collectors;
19
20 @Service("securityService")
21 public class SecurityService implements UserDetailsService {
22
23     @Autowired
24     private KotikiService service;
25
26     public Collection<? extends GrantedAuthority>
       mapRolesToAuthorities(Collection<String> roles) {
27         return roles.stream().map(SimpleGrantedAuthority::new).collect
       (Collectors.toList());
28     }
29
30     public User mapUserBDtoUserDetails(OwnersDto owner) {
31         return new User(owner.getLogin(), owner.getPassword(),
       mapRolesToAuthorities(List.of(owner.getRole())));
32     }
33
34     @Override
35     @Transactional
36     public UserDetails loadUserByUsername(String login) throws
       UsernameNotFoundException {
37         OwnersDto owner = service.findOwnerByLogin(login);
38         if (owner == null) {
39             throw new UsernameNotFoundException("Owner doesn't exist")
       ;
40         }
41         return mapUserBDtoUserDetails(owner);
```

```
42        }
43    }
```

Листинг 1.34: KotikiException.java

```
1  package com.kisssusha.service.tools;
2
3  public class KotikiException extends Exception {
4      public KotikiException() {
5          super();
6      }
7
8      public KotikiException(String message) {
9          super(message);
10     }
11
12     public KotikiException(String message, Throwable cause) {
13         super(message, cause);
14
15     }
16 }
```

Листинг 1.35: OwnerControllerTest.java

```
1  package com.kisssusha.controller;
2
3
4  import com.kisssusha.DAO.dto.OwnersDto;
5  import com.kisssusha.DAO.implemetations.OwnersDao;
6  import com.kisssusha.DAO.models.Owners;
7  import com.kisssusha.service.KotikiService;
8  import com.kisssusha.service.SecurityService;
9  import org.junit.jupiter.api.BeforeEach;
10 import org.junit.jupiter.api.Test;
11 import org.mockito.Mockito;
12 import org.springframework.beans.factory.annotation.Autowired;
13 import org.springframework.boot.test.autoconfigure.web.servlet.
     WebMvcTest;
14 import org.springframework.boot.test.mock.mockito.MockBean;
15 import org.springframework.test.web.servlet.MockMvc;
16 import org.springframework.test.web.servlet.request.
     MockMvcRequestBuilders;
17
18 import org.springframework.security.test.web.servlet.request.
     SecurityMockMvcRequestPostProcessors;
19
20 import static org.springframework.test.web.servlet.result.
     MockMvcResultMatchers.status;
21
22 @WebMvcTest(OwnerController.class)
23 public class OwnerControllerTest {
24
25     @MockBean
26     KotikiService service;
27
28     @MockBean
29     OwnersDao ownerDao;
30
31     @MockBean
32     SecurityService securityService;
33
34     @Autowired
35     private MockMvc mockMvc;
36     private Owners owner;
37     private  OwnersDto ownerDto;
38
39     @BeforeEach
40     public void setUp(){
41         owner = new Owners();
42         owner.setLogin("Ksusha");
43         owner.setPassword("$2a$12$NlJDln/
     GpsyGKxR5nG2W1eMyFxLRZsNuQHMKUdQtnTbGa46lmV90q");
44         owner.setRole("ROLE_USER");
```

```
45          ownerDto = new OwnersDto(owner);
46          Mockito.when(securityService.loadUserByUsername("Ksusha")).
47                  thenReturn(securityService.mapUserBDtoUserDetails(
    ownerDto));
48          Mockito.when(service.findOwnerByLogin("Ksusha")).thenReturn(
    ownerDto);
49          Mockito.when(ownerDao.findByLogin("Ksusha")).thenReturn(owner)
    ;
50      }
51
52      @Test
53      public void dontAllowPageUser() throws Exception {
54          this.mockMvc
55                  .perform(MockMvcRequestBuilders
56                          .get("http://localhost:8080/admin")
57                          .with(SecurityMockMvcRequestPostProcessors.
58                                  user("Ksusha")
59                                  .password("$2a$12$NlJDln/
    GpsyGKxR5nG2W1eMyFxLRZsNuQHMKUdQtnTbGa46lmV90q")
60                                  .roles("USER")))
61                  .andExpect(status().isForbidden());
62      }
63
64  }
```