System description:

This project presents my vision of educational task “e-care”. This project presents the system to administrate the work of mobile operator. System allows you to control, administrators, clients, client’s contracts and operator’s goods.

Used technologies and frameworks:

Application server: Tomcat, WildFly

Database: MySQL, Hibernate

Data transfer protocols: HTTP, WebSocket

Frontend: JSP + Bootstrap + JSTL, Angular, CSS

Testing: Junit, Mockito, Selenium

MQ server: RabbitMQ

Frameworks: EJB, Spring (Web, Security, Core, AOP)

Additional: Lombok, Maven

My features:

As additional features, we added notification to user by email, when creates user’s account. And another feature was prototype of money system. This feature include functions to add money on user’s balance, count monthly fee for contract (counts tariff and connected options), and debited money when connect new options.

Database scheme:

In database we have 8 tables. 6 of them represents different entities and 2 supporting ones to provide Many to Many relationships.

Option types table represents options category. It was added to execute the rule, that says we have some options which we can’t connect with each other. In my realization this rule transformed in we can connect to one contract only one option from one option’s category.

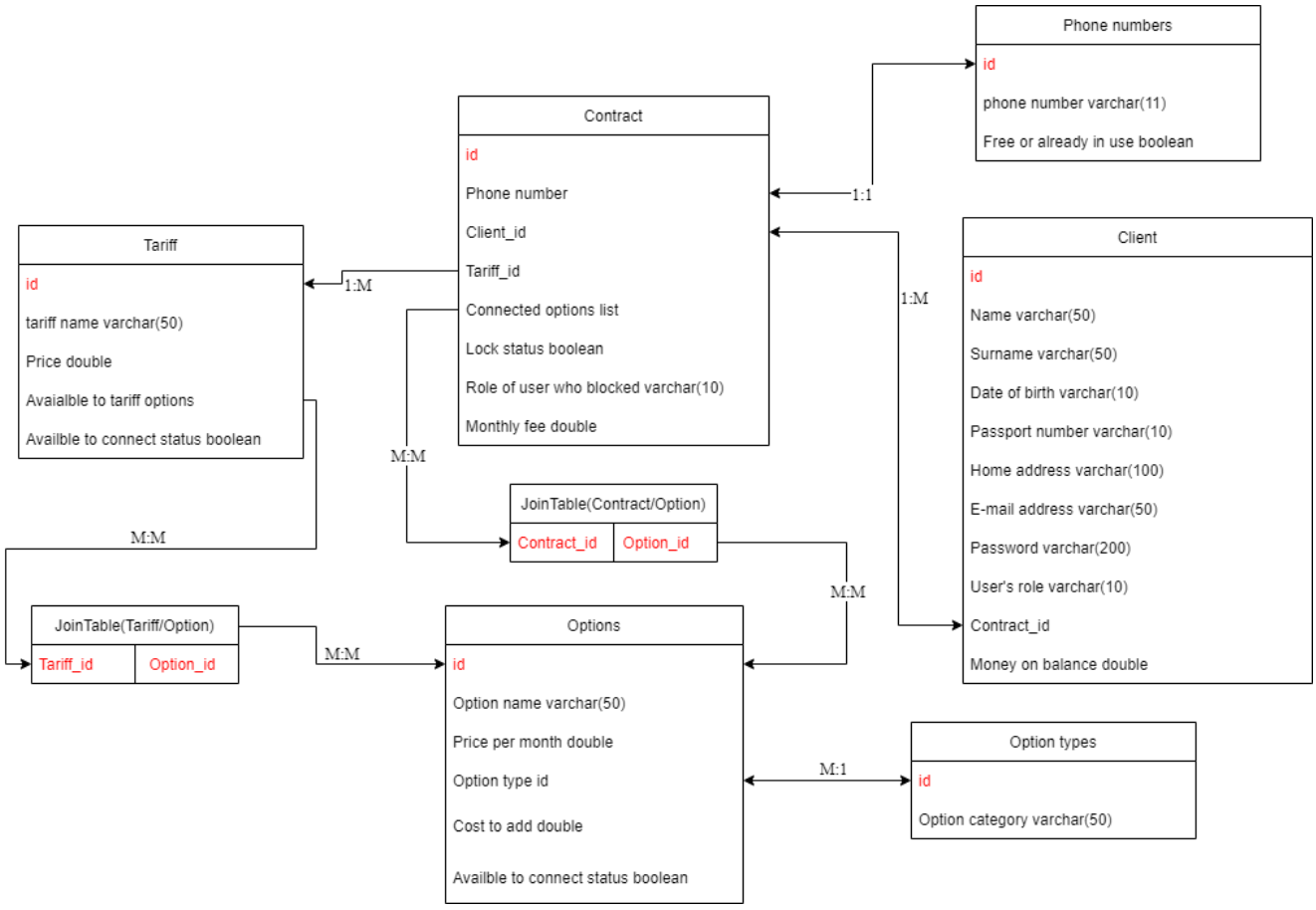
Options table represents model of option for tariffs. Option main fields was given in the task. Options and options types have many to one relationship, because we define that one option can belongs to one option’s type, but in system could be many options with the same option’s type

Tariff table represents model of tariff. Main fields for this entity were given in the task. Tariff and Options have many to many relationship because one tariff could have many options available to connect to him, and one option could be connected to many tariffs.

Phone numbers table was added, because we need to know which phone numbers already in use, and for this task we decided to take it out to another table. For make searching info about phone numbers easier than search info about them on every contract.

Client table represents the client's model in this system. Main fields of the clients were given in the task

Contract table represents the model of client's contract in the system. Contract and Phone numbers tables have one to one relationship because one phone number can be used only with one contract. Contract and Client tables have one to many relationship because contract could have only one client, but client could have many contracts connected to him. Contract and Tariff tables have one to many relationship because contract could have only one tariff connected and one tariff could be connected to many contracts. Contract and Options have many to many relationship, because many options could be connected to one contract, and one option could be connected to many contracts.



Picture 1 – Database scheme

Implementation of the models from the task:

Tariff model contains name, price, tariff connect status (could it be connected by new clients or not) and list of options which could be connected to the contract with this tariff.

Option model contains name, price per month, price to connect it to the tariff and connect status (could it be connected by new clients or not).

Client model contains name, surname, date of birth, passport number, home address, client's contracts id, email and password.

Contract model contains many relationships to other models. It has links to phone number, client who belongs this contract, id of the connected tariff, and link to the list of connected options for this contract.

Modules and their interaction:

<-! Пока хз>

UI:

In the first part of the task UI part made with help of JSP. On all pages is used header and footer template, and on some pages where represents the tables, used pagination template. As main décor library was used bootstrap 5, and partly was used css for header and footer.

Business logic:

Services represents in two parts. First one is a interfaces which describes what interface should do, and another one is implementation of the interface

ClientService – this service manages the work with clients, it controls CRUD operations, adding money on balance, and switch the password

ContractService – this service manages the work with contracts, it controls CRUD operations, checks options combination to validate, lock/unlock contracts, counts monthly fee for contracts.

NumberService – this service manages the work with phone numbers, it controls CRUD operations, and check if number already contains in system or not

OptionTypeService – this service manages the work with option types, it controls CRUD operations, and check if number already contains in system or not

OptionsService – this service manages the work with options, it controls CRUD operations,

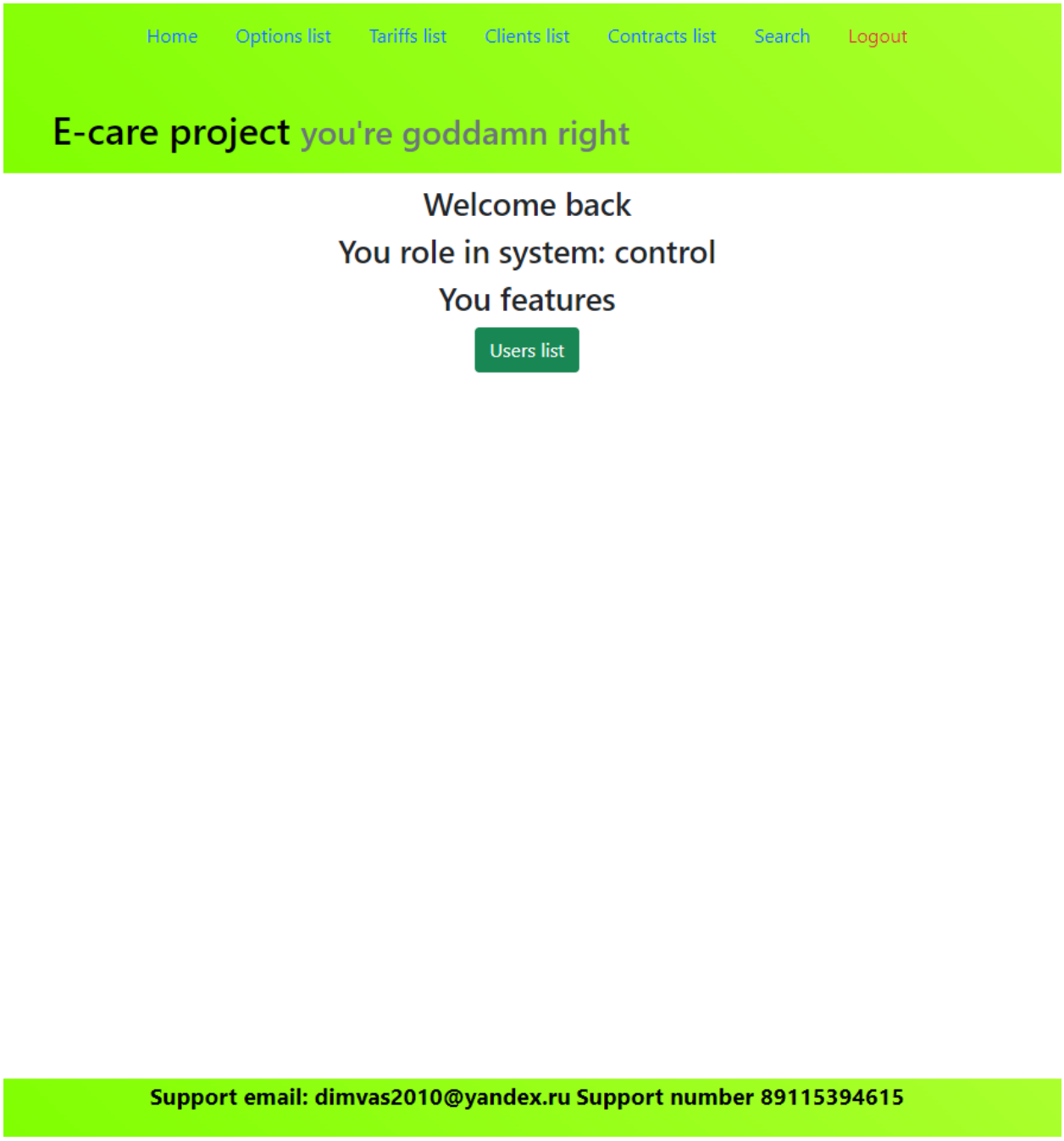
TariffService – this service manages the work with tariffs, it controls CRUD operations, and send notifications to MQ server when updates tariff.

Entities, DAO:

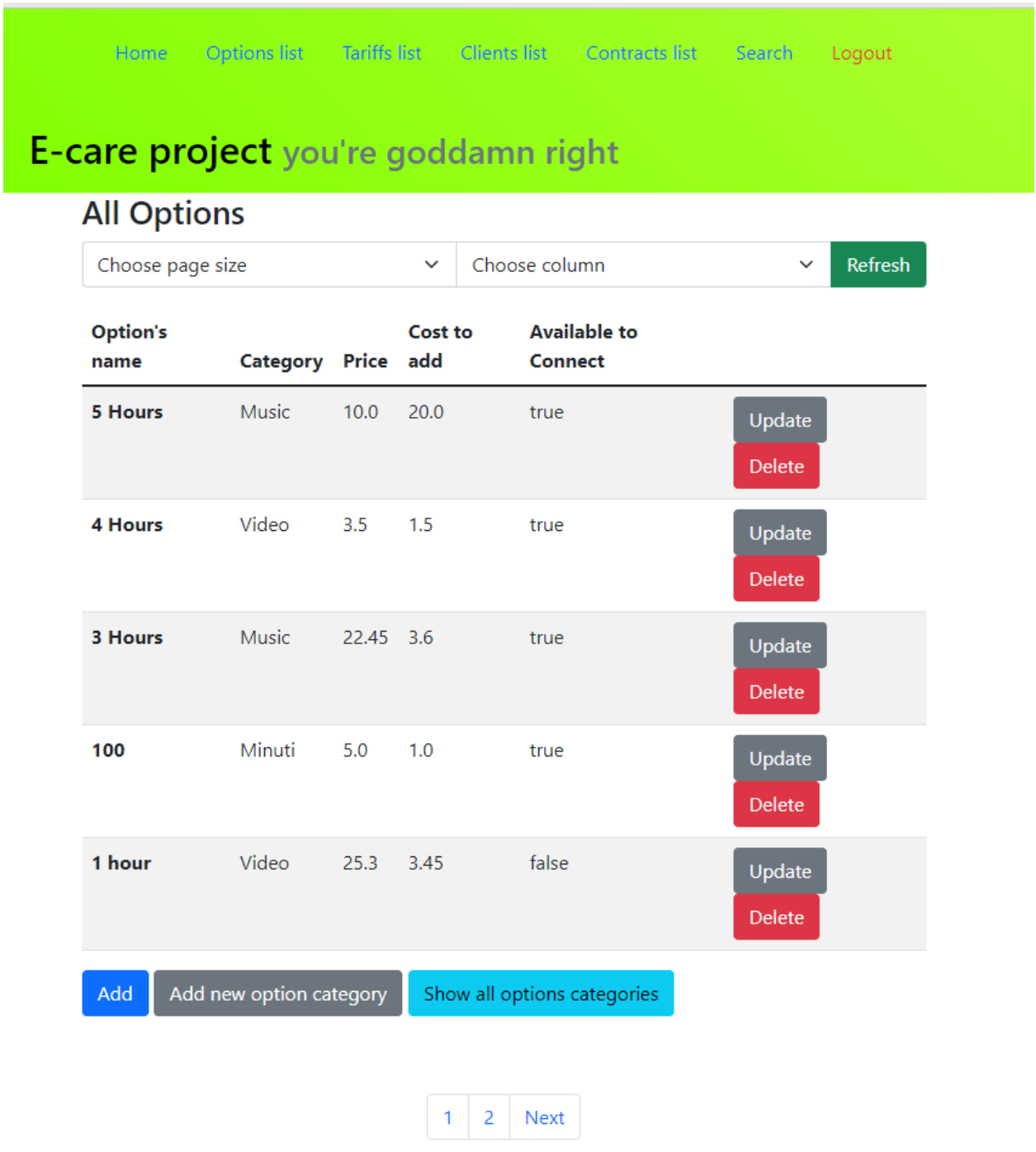
Entities represent 6 tables, which we described in database part. DAO layer represents CRUD operations, and pagination for all entities. In some cases like (phone numbers, options type, client's email) we asks dao to count records with the same information.

We isolated all operations with @Transactional, because we want to avoid accidental mistakes.

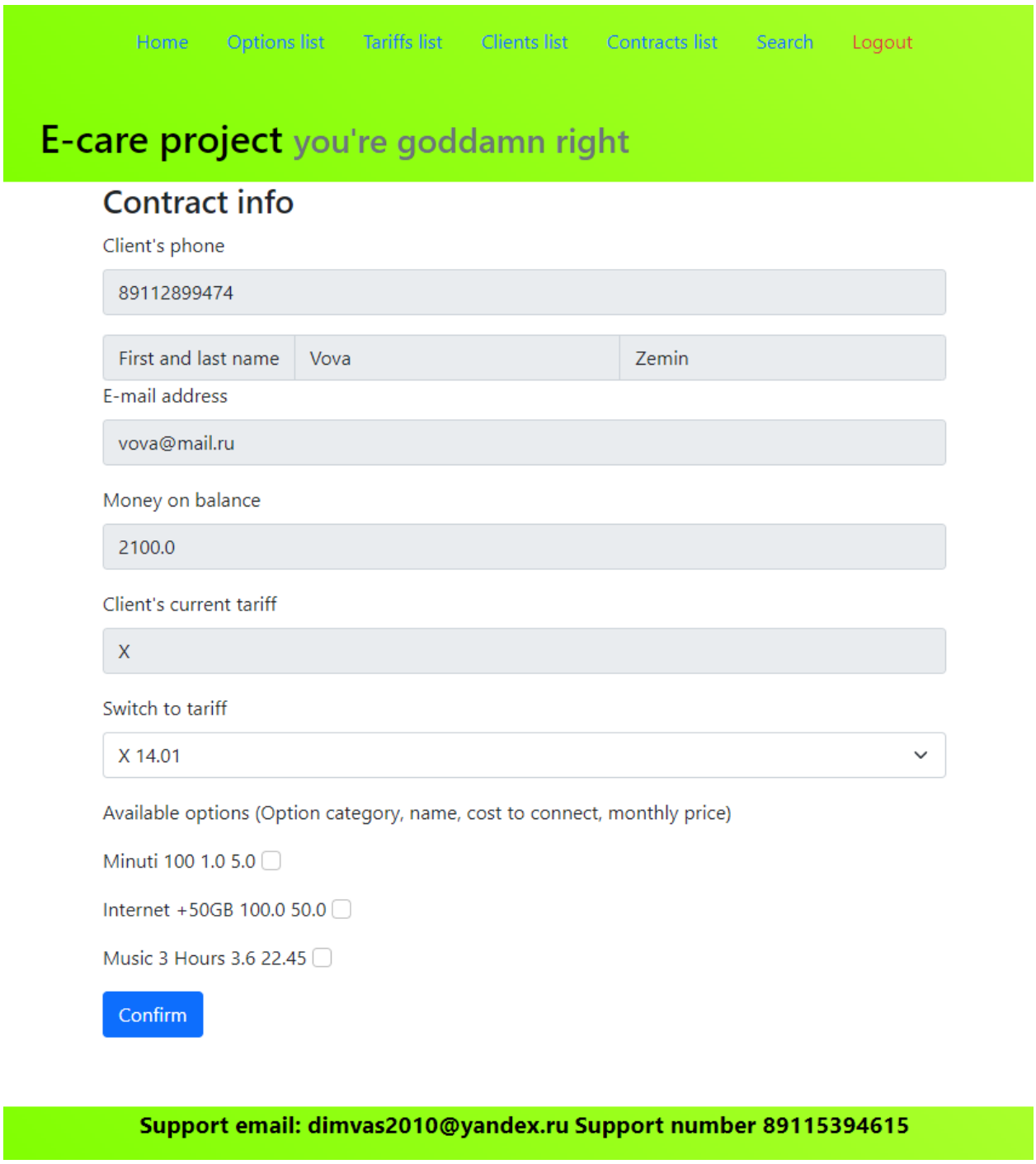
Screenshots:



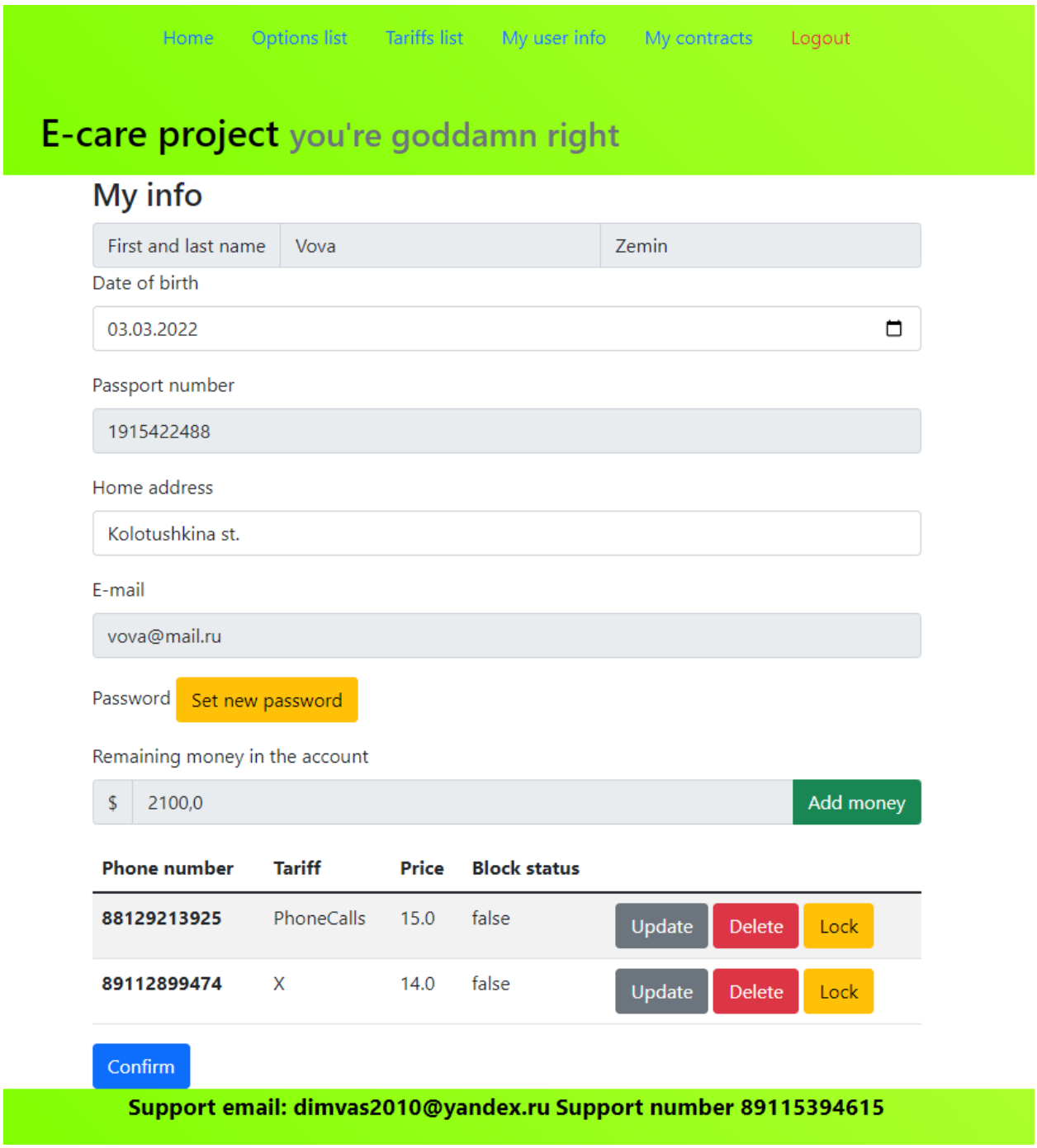
Picture 2 – Home page for admin



Picture 3 – Typical list page for administrations



Picture 4 – Manage contract page



Picture 5 – Client manage page for clients

Tests:

We have Junit tests and selenium tests.

Junit test have common part which connected with CRUD operations, it all check with them correct work, and we have common get tests which checks, if return value is correct. For every service we have part of the individual tests like check right options combination in contract parts.

Selenium test uses for test login part and forms for switch password and money add on user account.

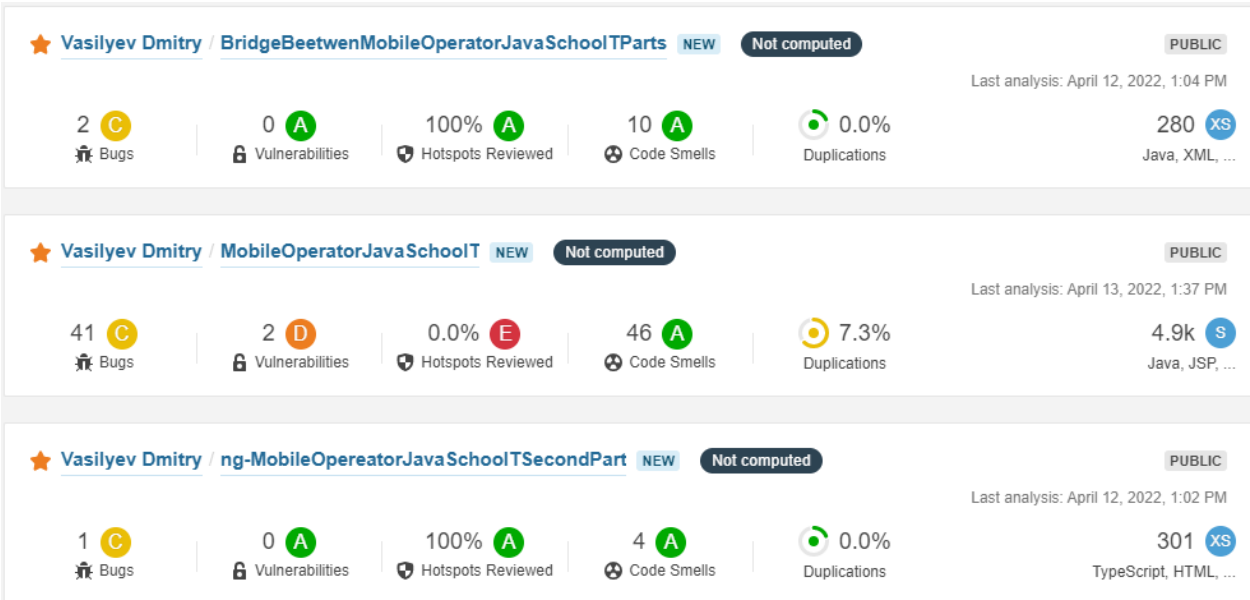
Logging:

In log we write info on the different layers. In trace layer we wrote controller's methods which were called. In info part we wrote information about how service methods work. And in the error layer we wrote information when handle the exception

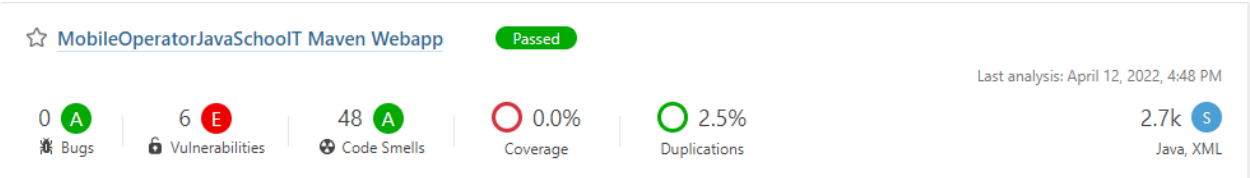
```
2022-04-13 14:21:00 INFO  LoggingAspect:67 - End method com.school.service.contracts.ClientService get
2022-04-13 14:21:04 TRACE LoggingAspect:31 - Begin method com.school.controller.ClientController patchClient
2022-04-13 14:21:04 INFO  LoggingAspect:62 - Begin method com.school.service.contracts.ClientService update
2022-04-13 14:21:04 ERROR exceptionAdvice:55 - User vova@mail.ru Can't update password. New passwords doesn't match
2022-04-13 14:21:04 TRACE LoggingAspect:31 - Begin method com.school.controller.ClientController changePassword
2022-04-13 14:21:04 INFO  LoggingAspect:62 - Begin method com.school.service.contracts.ClientService get
2022-04-13 14:21:04 INFO  LoggingAspect:67 - End method com.school.service.contracts.ClientService get
```

Picture 6 – Log example

Sonar statistic:



Picture 7 – Sonar cloud for all parts of project



Picture 8 – Sonar local server for first part

Improvement:

As the main problem we see that all dao methods is transactional. This is could be more serious problem if many users could connect to system. For that reason we want to improve system performance by fixing this problem in the next release.