*Technical Solution Description*

*«eCare»*

*Kalashnikov Sergey*.

Welcome to a maven based project, which represents an information system of mobile operator. This is a two independent applications, running in different scopes. Both of them, implements rest service to share data and Jms realisation to push the changes.

Subject area:

First : https://github.com/Kalashnikofffs/Tsystems is an MVC

application based on Spring Framework using jsp for view level.

Second:https://github.com/Kalashnikofffs/myEjbBoard

Java EE application using EJB to inversion of control and Jsf as a view technology.

Technologies:

⦁ Bootstrap 4.0

⦁ DB – MySQL 8

⦁ EJB 3

⦁ Spring 5.2.1

⦁ Spring security 5.2.0

⦁ Wildfly 18.0.0.Final

⦁ ActiveMQ

⦁ REST

⦁ GSON

⦁ Hibernate 5.4.6

⦁ Java 8

⦁ JavaScript

⦁ JSF 2.1

⦁ JSP 2.1

⦁ JUnit 5

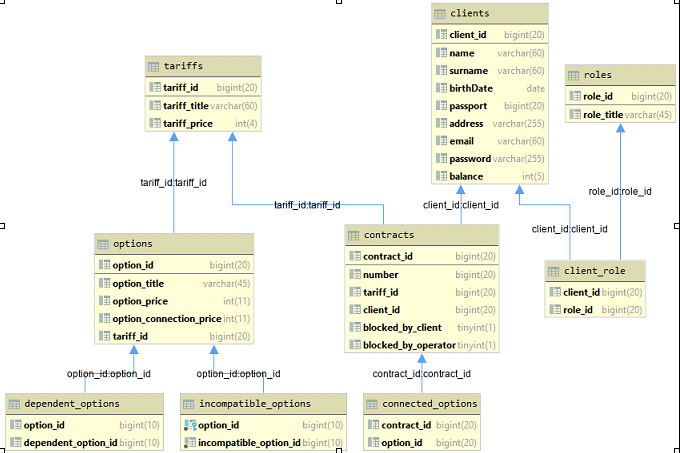
⦁ Log4j 1.2.17

**DataBase and DAO:**

Database built according to business logic and in concordance with NF3. Every entity related to model class. Every model class build using JPA. For each entity and their inner sets of relationships created 9 tables in the «ecare db» schema: clients, roles, client-role, contracts, connected\_options, tariffs, options, dependent\_options, and incompatible\_options.

Last two tables were made to solve a tricky case with incompatible and dependent options of tariffs. Some options depend on tariffs, some options forbid each other. It's actualized on a data layer for a strong reliability.

Other adjacent tables made for base normalisation. Every relation in database was reflected in the model level of application one.



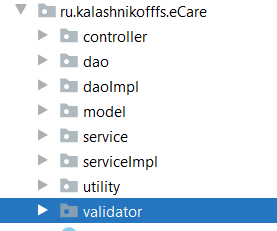
Complex complex relationships mapped according to database schema. Project use MySql 8 and database was created by Sql script.

Dao layers execute all methods to transaction data between database and application.

Also some methods push messages by Jms to notify consumers about changes.

Data-source configured with xml in resources directory.

**Spring MVC and Project Structure:**

Project based on Spring Framework and MVC pattern:

Each entity created a separate service interface and class-implementation for working with the database. All service implementation classes execute methods to produce a simple logic separated from view and data scope.

Service methods don't know about JPA connections and use a dao layer.

**JMS AND REST:**

The rest client integrated the main application interface and built on EJB and JSF technologies.

Information system(1st app) produces jms messaging to notify the Board(2nd app) about changing tariffs or options. There are 3 configuration beans using spring JMS api to configure correct production.

*ConnectionFactory*

*JmsTemplate*

*Convertor*

Together, they produce remote connection to JMS server, located in a different Jvm Scope.



Project one uses remote http connection to jms without JNDI. That makes both apps independent, by working and different jvm. Jms consumer is a MessageDrivenBean. It executes a rest request to get data on every Jms message to update the values on Board.

Board gets json data using a jersey.



Json object becomes POJO after receiving.

**Usage:**

There are two roles : User and Admin.

User has a profile, he can edit it and add funds on account. He can choose a tariff and some options, pay for that, and make a deal.

Admin can see all tariffs and options, edit it and delete. He can request a list of all users, edit their profiles, and funds on any account. He can see all contracts and change their details.

When tariff or option is added, edited, deleted : information on Board is changing automatically.

**Improvement:**

Project can be overwritten using spring boot.

Code can be refactor to be more SOLID.

Final variables can be added.

Frontend view can be improved by new quality style and scripts.

All logic methods should be tested by JUnit.

Project can run on Docker containers.

Browser driver autotests can be written.