Web-service for the first space communication operator

Task

To develop web-application for mobile network operator "Multiverse mobile" company. The application must be able to perform the required user cases, keep and represent business data.

User cases :

For clients :
- view one's contracts with included tariff and options;
- add/remove options to/from contract;
- change current tariff;
- block/unblock number;
- view available tariffs and options;

For managers:
- create/update/delete options and tariffs;
- create clients, read/update clients data;
- create contracts for clients with unique auto-generated phone number, block/unblock and change contract features;
- perform searching for clients and contract;

As an extra service to develop one-page web-application (app 2) to show special tariffs based on notification from main application.

Application description

Main application provides an authentication mechanism that restricts access to the web site pages depends on particular user role. There are two user roles: *manager* and *client*.

As *client* one can sign up and then sign in using phone number as login and access the *cabinet* page displays all information about contract: current tariff, activated options and also all tariffs and options available for purchase. *Client* can block the contract, change tariff, buy or delete options.

As *manager* one can sign in with provided login and access *management* page in order to manipulate with options, tariffs, contract and clients data. There is also extra feature to send news messages into the official company telegram channel.

Application can send messages to another services or applications using JMS API.

All business data is stored in reliable database. All user's passwords are stored in database in encrypted format.

App2 is able to receive up-to-date information from main application and display it on web-page in asynchronous manner.

Technologies

Back-end

API
- Java EE 8
- Java Persistence
- JavaServer Faces 2.3
- JavaServer Pages 2.1
- **Java** Message Service
- JNDI
- Telegram Bot

IDE:
- Intellij IDEA 2018.1 Ultimate edition

Database:
- MySQL 8.0.13

Project build and reporting management:
- Apache Maven 3.5.4

Application server:
- Wildfly 14.0.0.Final

Frameworks and libraries:
- Spring 5.0 as main model for java-based enterprise application
- Spring Security 5.0.5 for authentication and access-control
- Spring AOP
- Hibernate 5.2.17 as an Object/Relational Mapping (ORM) framework
- Google Gson (json format serialization)
- Apache Log4j 1.2.7 for logging
- Enterprise JavaBeans (server logic for extra app)

Testing instruments:
- H2 database Engine
- Docker 18.03.0
- Selenium WebDriver for automatic UI testing
- JUnit 5 framework
- Mockito 2.0.2 framework
- SonarQube 7.4 for code continuous inspection

Front-end

- Javascript
- AJAX
- jQuery
- HTML/CSS (web pages structure and style)
- Bootstrap 4.0.0 library (web pages style)

Architecture

The application structure implements *client-server* model in which many clients (web-browsers) request and receive service from a centralized server (host computer). Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them.

The *server* module implements *model-view-controller* design pattern:
- the *model* is responsible for managing the data.
- the view represent data as web-pages;
- the controller responds to the user input and performs interactions on the data model objects. The controller receives the input, passes the input to the model and then updates views.
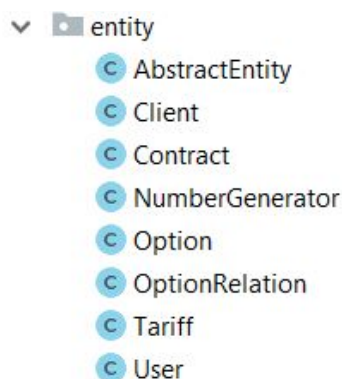
Picture 1 represents general application modules:
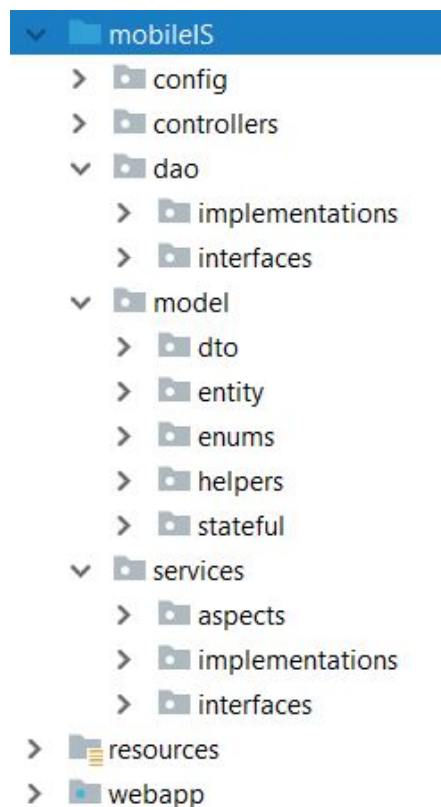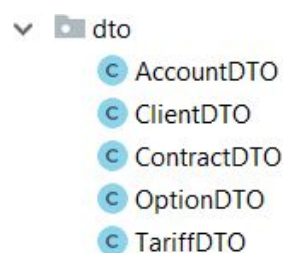

Picture 1

Model layer

Picture 2 represents entities stored in database.

Picture 3 represents **data transfer object (DTO)**: helper objects that keep user inputs when pass data from *controllers* to *model services* or keeps entities properties when pass data from *model* to *views*. **DTO** objects prevent persisted objects from unchecked mutates and isolate them from *controller* and *view* layer.

Picture 2   Picture 3

- entity
  - C AbstractEntity
  - C Client
  - C Contract
  - C NumberGenerator
  - C Option
  - C OptionRelation
  - C Tariff
  - C User

- dto
  - C AccountDTO
  - C ClientDTO
  - C ContractDTO
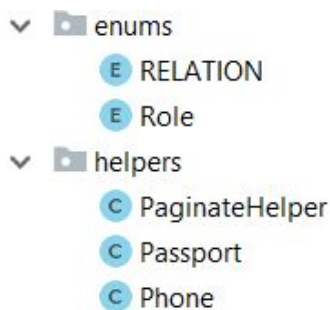 - C OptionDTO
  - C TariffDTO

Helpers **Passport** and **Phone** are special non-persisted object passed into *view* and used to validate user input (user passport and phone number must meet particular format).

PaginateHelper is used to retrieve from database limited number of entities in specific range. (Picture 4).

Cart is a special object that keep state during user session in order to keep current purchase. (Picture 5).

Picture 4   Picture 5

- enums
  - E RELATION
  - E Role
- helpers
  - C PaginateHelper
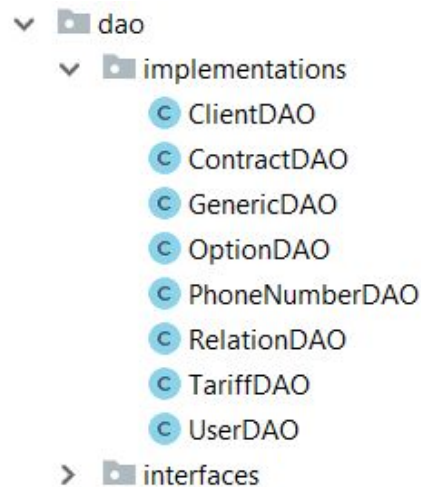 - C Passport
  - C Phone

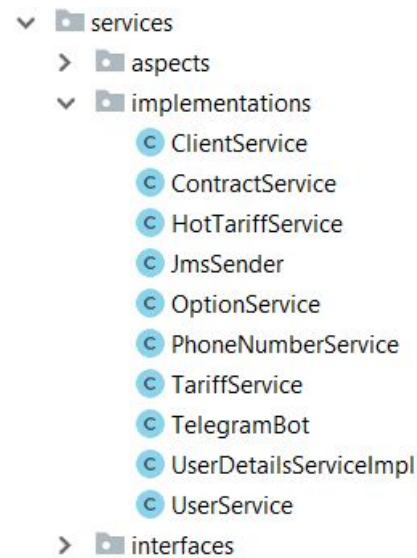- stateful
  - C Cart
  - I CartInterface
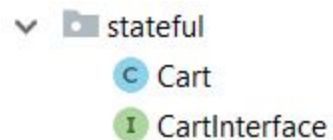
Picture 6,8 represents object used to access database (data accessible objects, or DAO). Each **DAO** implements corresponding interface.

Picture 7 represents *model services.* These services perform all business logic. Each service implements corresponding interface.

Picture 6   Picture 7

Picture 8
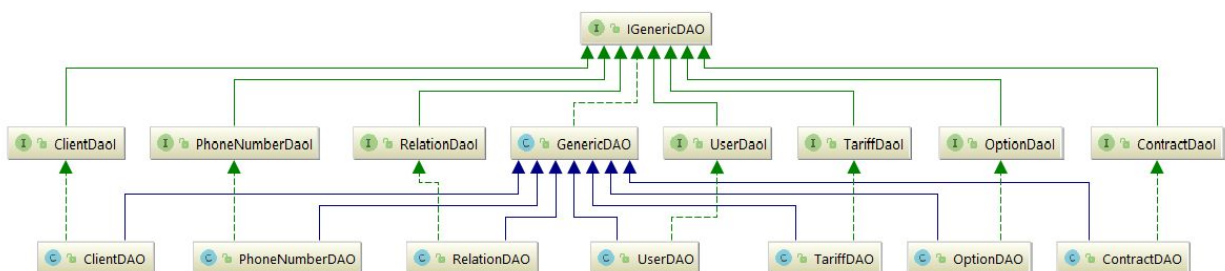
HotTariffService deals with tariffs marked as 'hot'.

JmsSender performs sending messages into server message queue via JMS API.

PhoneNumberService auto-generates unique phone numbers.

TelegramBot sends messages for the telegram channel.

UserDetailServiceImpl maintains security mechanism.

Picture 9 represents services implementing via AOP. **HotTariffAspect** is used to catch "hot" tariff events and notify **HotTariffService**. LoggAspect is used to log about application lifecycle.

Picture 9

- ∨ 📁 services
  - ∨ 📁 aspects
    - © HotTariffAspect
    - © LoggAspect
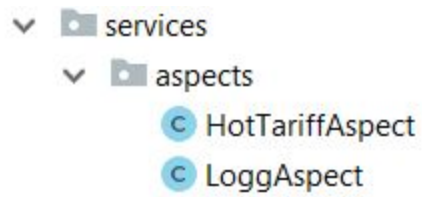
Controller layer

Picture 10 represents controller layer.

```
∨  controllers
      C AuthController
      C ClientController
      C ContractController
      C GlobalExceptionHandler
      C HotTariffController
      C MainPageController
      C ManagerCabinet
      C OptionController
      C TariffController
      C UserCabinet
```
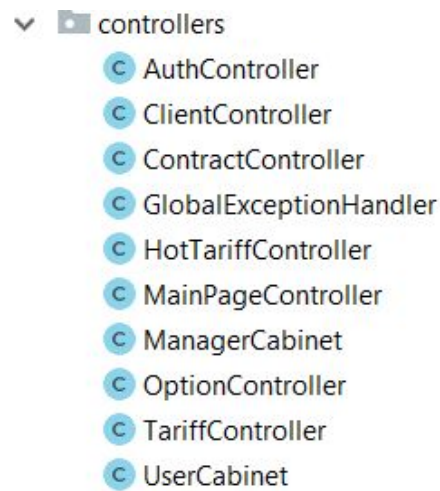
View layer

Picture 11 represents view layer.

```
∨  webapp
   >  resources
   ∨  WEB-INF
      ∨  views
         ∨  management
            >  client
            >  contract
            >  option
            >  tariff
               manager-cabinet.jsp
         ∨  user
               user-cabinet.jsp
         404.jsp
         error.jsp
         index.jsp
         login.jsp
         sign-up.jsp
```
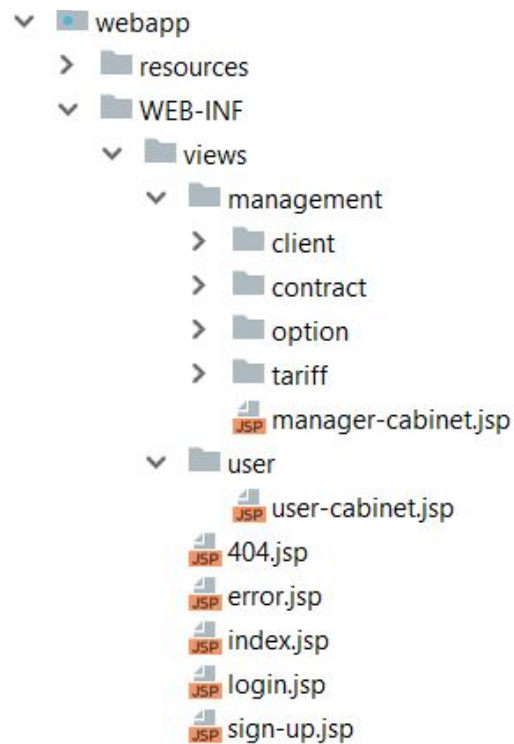
App2 architecture

Extra application is a one-page application based on EJB and JSF frameworks. **Cash** is a managed bean used as model for keeping data. JSF page is used as view for representing data. **Tariff** is a helper class to organize data in view. **Receiver** perform message receiving from main application via JMS API.
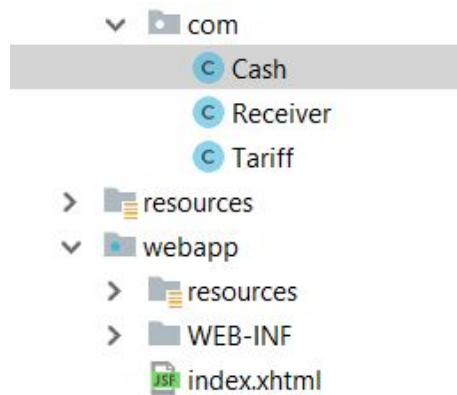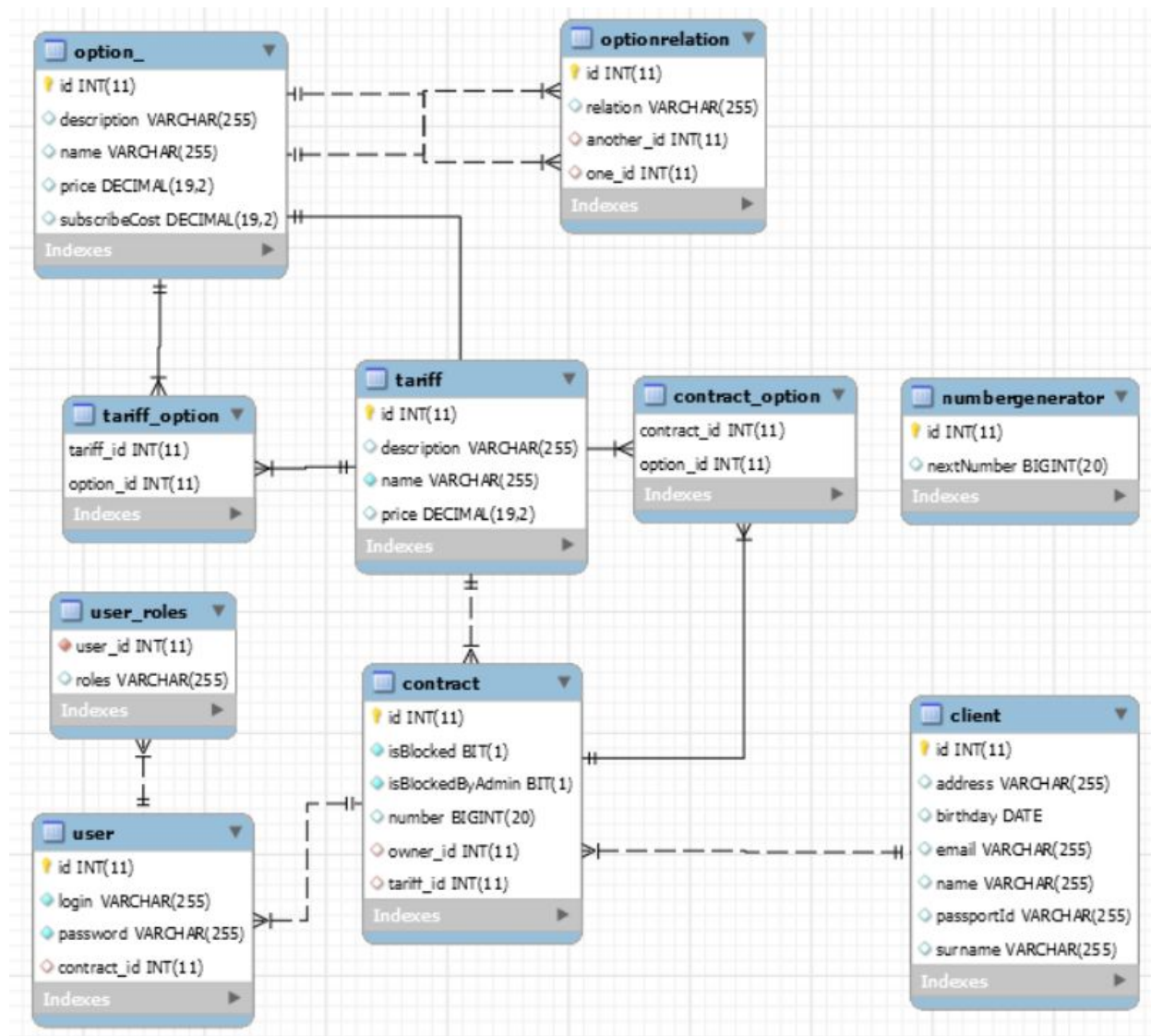
Database schema

Optimisation points

User cases

For manager:

- Manager can sign in with login and password provided by administrator and access *manager cabinet page.*
- Manager can list all options, tariffs, contract and clients; find client by passport id or by phone number (if client has at least one contract); find contract by phone number;
- Manager can create new and update existed options. During this process manager can set up options compatibility rules: some options can be incompatible with each other, come options can be mandatory for others. If setted rules violate existing rules/logic and can't be performed, there would be an error message presented on the page.
- Manager can create new or edit existed tariff. Tariff is just set of options. If options in tariff are incompatible with each other, there would be an error message presented on the page.
- Manager can create client with personal info. Some personal info must be unique. If this restriction is broken, there would be an error message presented on the page.
- Manager can create new contract for specific client. Contract cannot exist without owner-client. One client can has many contracts. Contract consist of tariff and may include any number of options. Options in contract must be compatible with contract tariff. Manager can block/unblock contracts (client may not change any options in blocked contract and may not unblock it)
- As additional option, manager can send message for the official telegram channel.

For client:

- Client can register himself on web site using her/his phone number as login. One account per phone number is created. If input phone number doesn't exist, register won't success.
- Registered client can sign in with login and password and access the *cabinet* page displays all information about contract: current tariff, activated options and also all tariffs and options available for purchase. Client can block the contract, change tariff, buy or delete options. Blocked contract can't be updated. Client can not unblock contract that was blocked by manager.

Hot tariffs

There are tariff marked as 'hot'. Information about hot tariffs is used in App2 and is sent via JMS. The first hot tariff message is sent immediately after application is started in case App2 is already waiting for data. Messages have short time-to-live in order to keep in queue only fresh data. If any of hot tariffs got any updates, a new message with corresponding data is sent.

App2 send GET request to the main application during startup asking for up-to-date hot tariffs data. As message has been received, view is updated asynchronosly via Websocket technology.

Telegram Bot

Main application generates random news from current business information and sends messages automatically to the telegram channel in regular manner. Also *manager* can send custom message for the official telegram channel right from manager cabinet.

Testing environment

Unit tests

Unit tests cover all service layer logic. All external services and resources in unit testing are mocked using Mockito library.

Integration tests

Integration tests run in Spring webcontext with only required component being initialized. A lightweight in-memory database H2 Engine populated with test data is used here instead of production database MySql.

UI integration tests

These tests run in Docker container, an isolated virtual environment.
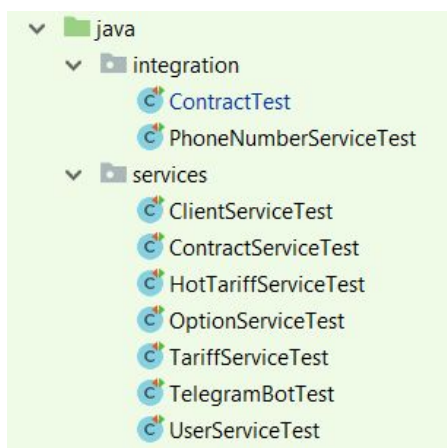Before tests:
- the main application is packaged into war archive;
- container with Wildfly server and Mysql server are up and run;
- database is created and populated with test data;
- application is deployed on Wildfly server;

During these steps test main thread is waiting for successful application startup (with timeout period). If so, the UI tests runs using Selenium WebDriver.

After test docker containers and images are destroyed, and test database data remains in original state. Also log files from containers are copied to the host folder.

Code Coverage

Testing reports are automatically generated by Maven plugins during test building phase and used in Sonarqube code inspection reports.

```
v  java                          [INFO] Tests run: 36, Failures: 0, Errors: 0, Skipped: 0
  v  integration                 [INFO]
     C  ContractTest             [INFO]
     C  PhoneNumberServiceTest    [INFO] --- maven-war-plugin:3.0.0:war (default-war) @ mobile ---
  v  services                    [INFO] Packaging webapp
     C  ClientServiceTest        [INFO] Assembling webapp [mobile] in [C:\Public\mobileIS\mobileIS\target\mobile]
     C  ContractServiceTest      [INFO] Processing war project
     C  HotTariffServiceTest     [INFO] Copying webapp resources [C:\Public\mobileIS\mobileIS\src\main\webapp]
     C  OptionServiceTest        [INFO] Webapp assembled in [563 msecs]
     C  TariffServiceTest        [INFO] Building war: C:\Public\mobileIS\mobileIS\src\test\resources\mobile\mobile.war
     C  TelegramBotTest          [INFO] ------------------------------------------------------------
     C  UserServiceTest          [INFO] BUILD SUCCESS
                                  [INFO] ------------------------------------------------------------
```

Code quality

SonarQube measures are presented below:
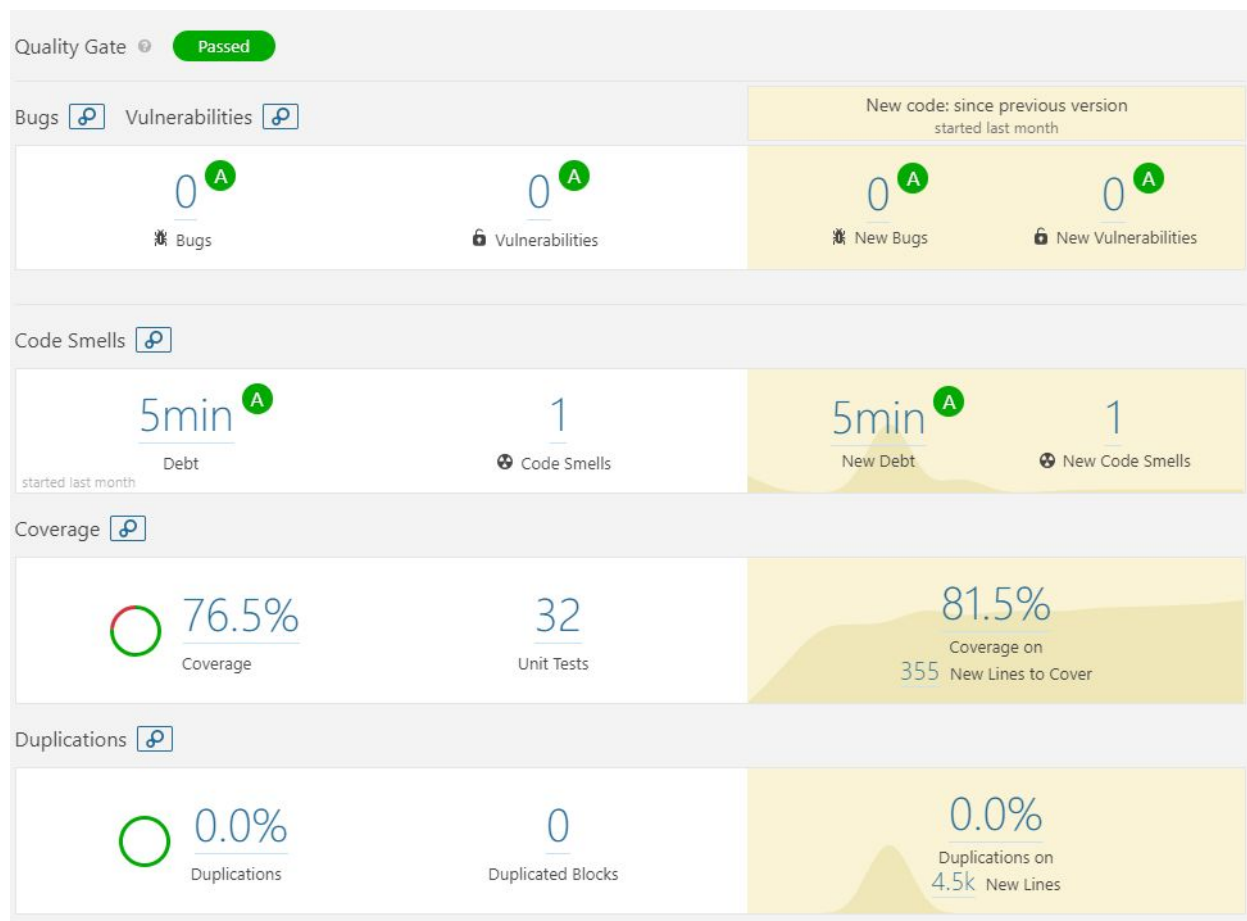
Project overview:

Size	Lines of Code	
New Lines		5,114
Lines of Code		3,357
Lines		5,066
Statements		1,119
Functions		472
Classes		73
Files		73
Directories		12
Comment Lines		362
Comments (%)		9.7%

UI

Main page

SIGN IN | SIGN UP

Multiverse mobile
Feel free to communicate through space and time

We offer

Any destination
Calls without time-lag to/from all over the existing universe

Safety
We provide reliable service of time-paradox defence

No time limitation
Call to the past or future with our tachyon-tunnel technology

Options

Time paradox protection
Don't worry about time paradox issues while communicating with past and future!

Tariffs

Home planet	Star trek	Solar System
Unlimited calls and messages within the orbit of specific planet	Calls and messages within the visible part of Universe	Unlimited calls and messages within the Solar System (Pluto excluded)
$99.99 per month	$1701.00 per month	$180.00 per month

Made by Veshka

Client cabinet

Cabinet   LOG OUT

Phone number: 9990000004   Cart:

Tariff: Home planet

Unlimited calls and messages within   Total: $0   Buy
the orbit of specific planet

Cost: $99.99 per month

Block number

Active options   Available options   Tariffs

Deep space   **Black Hole Alert**   **Spolier ban**

Provide compatibility with long-   Get notification about black holes   Prevent receiving calls and messages
distance services   approaching   from future

$0.00 per month $15.50 per month $0.00 per month

$0.00 for subscribe $300.00 for subscribe $28.95 for subscribe

Get   Get   Get

<

Show more >

Manager cabinet

Find client by phone:

| 1234567890 | Find |

Find client by passport id:

| 1234567890 | Find |

All clients

id	Name	Surname	Passport	e-mail		
1	Edsger	Dijkstra	1111111111	coolalgs@nl.com	Edit	Show details
2	Donald	Knuth	2222222222	computer_art@us.com	Edit	Show details
3	Robert	Sedgewick	3333333333	princeton_rs@us.com	Edit	Show details

| Previous | 1 | 2 | Next |

Add new client

Back to tariffs

Tariff details

Name:

Price:

| 0 |

Description:

Set options:

Showing all 8   Empty list

| Filter | | Filter |

| ➔ ➔ | | ← ← |

Time paradox protection
Hologram
Deep space
Tachyon protocol
I love Pluto
Spolier ban

Save

Build and deploy steps

1. Run Wildfly Server:
   standalone.bat -c standalone-full.xml

2. Run docker-machine:
   docker-machine start

3. Main application installation (in mobileIS/ directory):
   mvn clean install

4. App2 installation installation (in mobile2/ directory):
   mvn clean install

5. Perform SonarQube analysis (in mobileIS/ directory):
   mvn sonar:sonar

6. Main application depoy (in mobileIS/ directory):
   mvn wildfly:deploy-only

7. App2 deploy (in mobile2/ directory)
   mvn wildfly:deploy-only

Known bugs

Further improvements

There are several ways for improvements.

Functionality:
- add two-factor authentication, OAuth support;
- let user change his/her password;
- send notifications via email, sms;
- collect and represent statistic information;
- develop bot for advice best tariff/options for user based on specific requirements;
- add site-search;

Optimization:
- improve performance by using more asynchronous requests and RESTful services;
- Use optimisation possibilities of ORM framework;
- Perform load testing to find out weak nodes;
- implement cache mechanism to decrease number of database accesses;