***eCare project***

Technical Solution Description

Plate Alexej

Content

[Introduction 3](#_Toc464556164)

[Technologies and frameworks 4](#_Toc464556165)

[Additional features 5](#_Toc464556166)

[Database 6](#_Toc464556167)

[Scheme 6](#_Toc464556168)

[Description 7](#_Toc464556169)

[Explication of the model 8](#_Toc464556170)

[Modules 9](#_Toc464556171)

[User interface 11](#_Toc464556172)

[Fundamentals 11](#_Toc464556173)

[List of available pages 12](#_Toc464556174)

[Additional features 13](#_Toc464556175)

[Services 15](#_Toc464556176)

[List of methods 16](#_Toc464556177)

[Entities and repository 18](#_Toc464556178)

[Entities 18](#_Toc464556179)

[Repository 21](#_Toc464556180)

[Screenshots 22](#_Toc464556181)

[Tests 26](#_Toc464556182)

[Backend tests 26](#_Toc464556183)

[Frontend tests 28](#_Toc464556184)

[Additional features 30](#_Toc464556185)

[Custom utils 31](#_Toc464556186)

[Exceptions 31](#_Toc464556187)

# Introduction

eCare poject – web application for mobile operator. This project gives permission to manage the options, tariffs, customers and contracts. Functionality of the application includes creation and observing of contracts and options. Comfortable option system permits to define complicated relationships between them without destructions of consistency. Users with administrator privileges have an opportunity of new customers’ registration, addition, editing, removing and blocking of customers’ contracts. Usual users have a right to inspect options and tariffs, and manage their personal contracts.

Special Telegram bot allows to use basic service in convenient format.

# Technologies and frameworks

* Spring
  + Spring MVC
  + Spring Data JPA
  + Spring AOP
  + Spring security
* Hibernate
* jQuery, css
* EJB
* JSF
* Maven
* Python (for telegram bot)

# Additional features

Password recovery via email with temporary code sending.

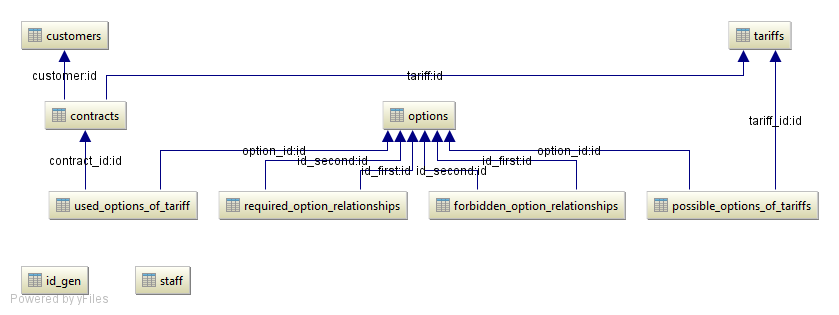
New customers defined in system without password until password creation.

Telegram bot, that allows to get tariffs and options information, view contracts and change password.

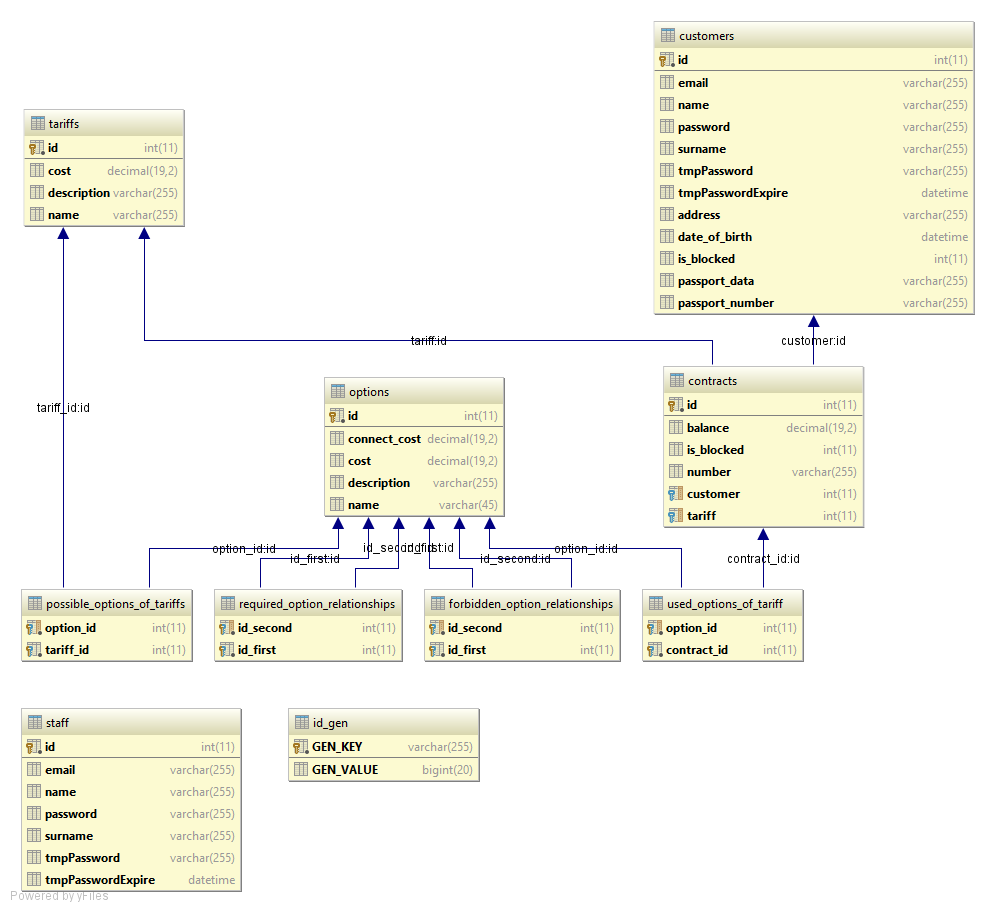
# Database

## Scheme

Database diagram (small)



Database diagram (full)10/18/2016



## Description

Main tables

|  |  |
| --- | --- |
| **Table** | **Description** |
| Customers | Customers information. Contains name, surname, email, password, etc. |
| Contracts | Contracts information. Contains number, balance, block level, etc. |
| Tariffs | Tariffs information. Contains name, cost, description. |
| Options | Options information. Contains name, cost, description, etc |
| Staff | Staff information. Contains name, surname, email, etc. |
| Id\_gen | Helper table. (See below) |

Relationships table

|  |  |
| --- | --- |
| **Table** | **Description** |
| Possible\_options\_of\_tariffs | Options – Tariffs many-to-many relationship.  Define options, that available for tariff. |
| Required\_options\_relationships | Options – Options many-to-many relationship. Define “required” relationship between options. |
| Forbidden\_options\_relationships | Options – Options many-to-many relationship. Define “incompatible” relationship between options. |
| Used\_options\_of\_tariff | Options – Contracts many-to-many relationship. Define chosen options by contract. |

In code, tables “Customers” and “Staff” have same abstract parent “User”.

Table “id\_gen” is used for defining ids for “Customer” and “Staff” tables.

# Explication of the model

**Customer** – usual user of service. Has access to tariffs and options description. Can manipulate with personal contracts.

**Staff** – user of service with administration privileges. Has permission for creating new tariffs and options. Can register new customers and add new contracts to existing customers. Contract, blocked by staff, could not be used and unblocked by customer.

**Tariff** –plan of contract. Main “rule” of contract. Defines options, that can be used by contract. Every contract has exactly one tariff.

**Option** – additional “rule” for contract. Provided by used tariff. Every option provided by one or more tariffs. Some options have relationships “required” and “incompatible” between them.

**Contract** – combination of used tariff and options. Defined by customer’s phone number. Every contract has exactly one customer. Can be blocked by customer or eCare. If contract is blocked by eCare, customer cannot unblock it.

**Basket** – saved instances of edited contracts. If customer begins to change contract and then leaves page, all changes would be saved and shown on return.

# Modules

Web application is designed using REST architecture and can be used by every REST client.

Resources, provided by REST:

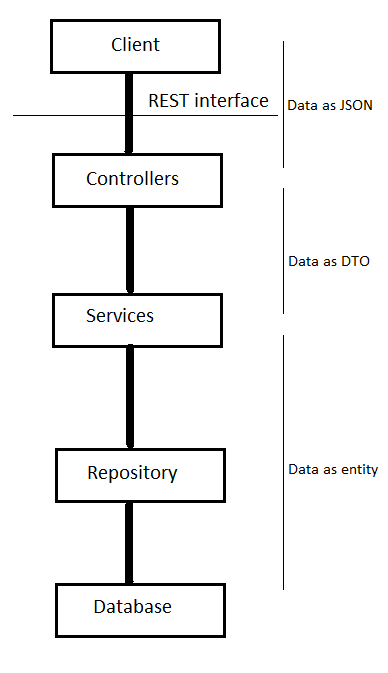
* Tariff
* Option
* Customer
* Contract

Backend is designed as 3 tier architecture. It contains controllers, services and repository layers.

Data transfer:

* Database – service | as entity
* Service – controller | as DTO
* Controller – client | as JSON

Modules interaction



# User interface

## Fundamentals

User interface created using HTML5, CSS and JavaScript.

Used libraries and plugins:

* jQuery
* DataTable
* Metis menu
* Notify
* Font-awesome
* Sb-admin
* Bootstrap

Interface is based on sb-admin-2 template.

Main files of user interface are:

* login.jsp – start page for user login.
* restore.jsp – page for password restore
* index.jsp – main page (see below)
* pieces.html – file with html pieces (see below)
* script.js – file with all scripts

Interface use HTML import to show separate pages. At the first login user loads index.jsp and pieces.html files. Pieces.html contains all html divs, that will be used through session.

On every menu click jQuery script cleans main page area and loads new content from pieces.html. The only thing that must be loaded – data through ajax call to rest api.

## List of available pages

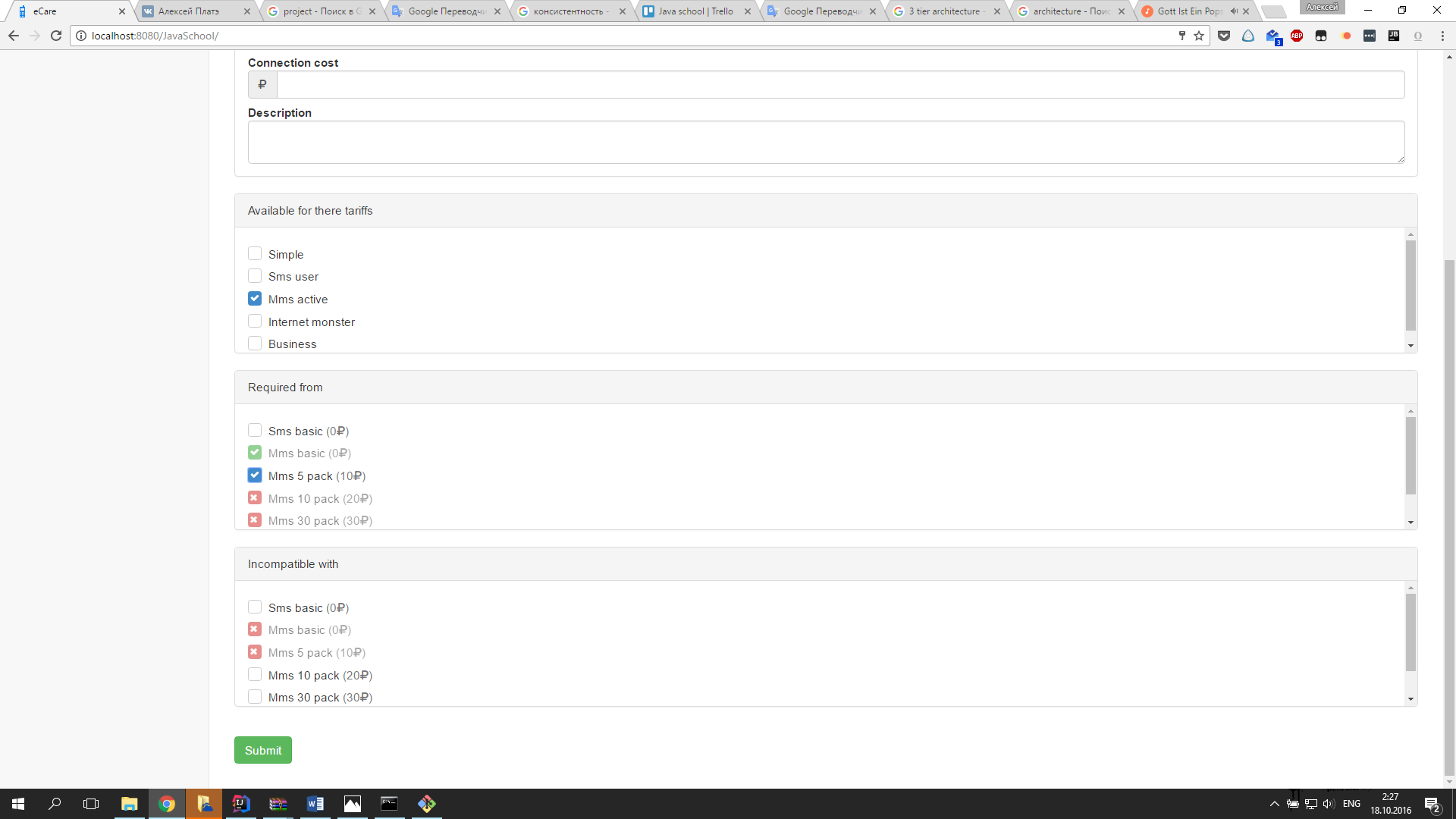
* login – login to service or go to page with password restoring
* restore – restore password with temporary code
* admin pages (menu points)
  + Show all customers – table of all customers. Searching and switch to specific customer. Show contract of customer.
  + Show all tariffs – table of all tariffs. Searching and list of available options
  + Show all options – table of all options. Searching and list of tariffs
  + Show all contracts – table of all contract. Searching and used options. Switch to customer with chosen contract
  + Add new options – add new option. Defining name, cost, tariff, relationships etc.
  + Add new tariff – add new tariff. Defining name, cost, options etc.
  + Add new customer – add new customer. Defining name, surname, passport data, new contract data, etc
  + Specific customer page – page of customer. Allows to add, edit, block and delete used contracts.
* Customer pages (menu points)
  + Me – page of customer. Allows to edit, block and delete contracts
  + Show all tariffs – table of all tariffs. Searching and list of available options
  + Show all options – table of all options. Searching and list of tariffs
* Change password – page for password changing. Need to enter old and new passwords.

## Additional features

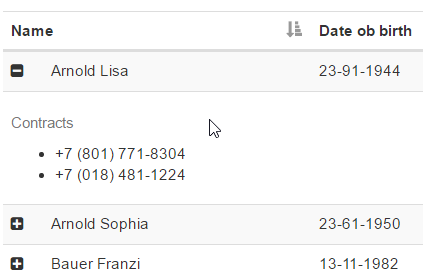
Web interface provided smart options relationships resolving system.

If one option is chosen, all options, that are required would be chosen and customer cannot uncheck it. All Incompatible options would be disabled and unchecked.

This rules are defined for each case and prevent wrong option configuration.

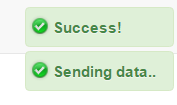


Every row in every table provided addition information, hidden by button



User notifying





# Services

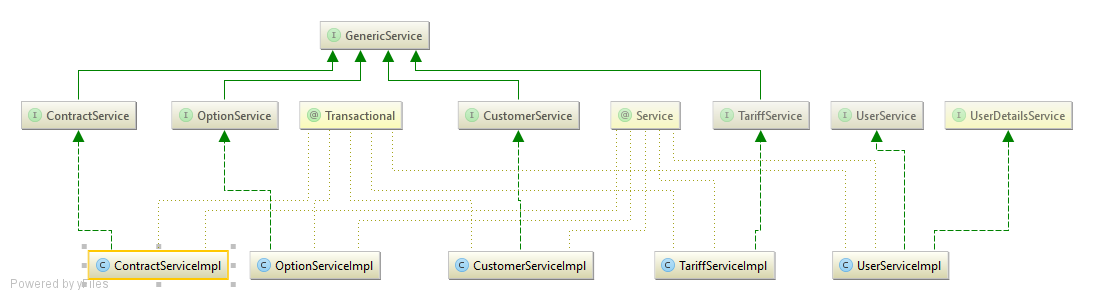
Application contains five services and one generic service

* Generic service
* Customer service
* Contract service
* Tariff service
* Options service
* User service

All services have interfaces and it’s implementation.

In service layer take place DTO – Entity convention.

All services are extended by GenericService



## List of methods

**GenericService**

|  |  |
| --- | --- |
| Method | Description |
| addNew | Add new entity to database |
| loadByKey | Load entity from database by id |
| remove | Remove entity from database |
| loadAll | Load all entities from database |

**ContractService**

|  |  |
| --- | --- |
| Method | Description |
| setBlock | Set block level for contract |
| updateContract | Update contract with new option and tariffs |
| findByNumber | Find contract by given number |
| findByTariffName | Find all contracts by given tariff |

**CustomerService**

|  |  |
| --- | --- |
| Method | Description |
| removeContract | Remove contract of customer |

**OptionService**

No specific methods

**TariffService**

No specific methods

**UserService (Not extended by GenericService)**

|  |  |
| --- | --- |
| Method | Description |
| changePassword | Change password by old password |
| changePasswordWithCode | Change password by temporary code |
| generateTempCode | Generate new code for user |

# Entities and repository

## Entities

Entities package contains 6 classes

* Contract
* Customer
* Option
* Staff
* Tariff
* User

**User** – abstract class, represents user of system.

|  |  |
| --- | --- |
| Field | Description |
| id | ID of user. Generated with help of “ID\_GEN” database |
| name | First name of user |
| surname | Last name of user |
| email | Email of user |
| password | Password of user. Hashed with BCrypt |
| tmpPassword | Temporary code of user. User for password changing. Hashed with BCrypt |
| tmpPasswordExpire | Temporary code expiring time |

**Staff** – extended by User. Represents user with administration privileges. Has no additional fields.

**Customer** – extended by User. Represents usual user of service.

|  |  |
| --- | --- |
| Field | Description |
| dateOfBirth | Birthday of customer |
| passportNumber | Passport number of customer. Must be unique in system |
| passportData | Additional passport data |
| address | Address of user |
| isBlocked | Define block level of user |
| contracts | Set of user’s contracts |

**Contract** – represents contract. Has unique phone numbers. Has only one customer.

|  |  |
| --- | --- |
| Field | Description |
| id | ID of contract. Auto generated by database |
| number | Contract phone number. Must be unique |
| isBlocked | Define block level of contract |
| balance | Balance of contract |
| customer | Contract’s customer |
| tariff | Used tariff of contract |
| usedOptions | Set of used options |

**Option –** represents options. Have one or more contracts.

|  |  |
| --- | --- |
| Field | Description |
| id | ID of option. Auto generated by database |
| name | Name of option. Must be unique |
| cost | Cost per month of option |
| connectionCost | Cost of option switching-on |
| description | Description of option |
| required | Set of options. Define “required from” relationships |
| requiredMe | Set of options. Define “this options require me” relationships. Mapping to “required” field |
| forbidden | Set of options. Define “incompatible” relationships |
| possibleTariffsOfOption | Set of tariffs, that use this option |
| contractsThoseUseOption | Set of contracts, that use this option |

**Tariff** – represents tariffs. Has zero of more options.

|  |  |
| --- | --- |
| Field | Description |
| id | ID of tariff. Auto generated by database |
| name | Name of tariff. Must be unique |
| cost | Cost per month of tariff |
| description | Description of tariff |
| possibleOptions | Set of options. Define available options for tariff. |

## Repository

Application created with Spring Data framework and JPA Hibernate implementation.

Data source defined by JNDI on server.

Full repository package represented as interfaces. Implementation of interfaces provided by Spring framework. All interfaces extended by JpaRepository, that allows to use basic CRUD operations.

Some interfaces have additional methods. Spring framework provides implementation by method’s name. All methods have clear to understand name and don’t require any description.

**ContractRepository**

* findByNumber
* findByTariff\_Name

**CustomerRepository**

* findByPassportNumberOrEmail
* findByContract\_Number

**OptionRepository**

* findByName

**TariffRepository**

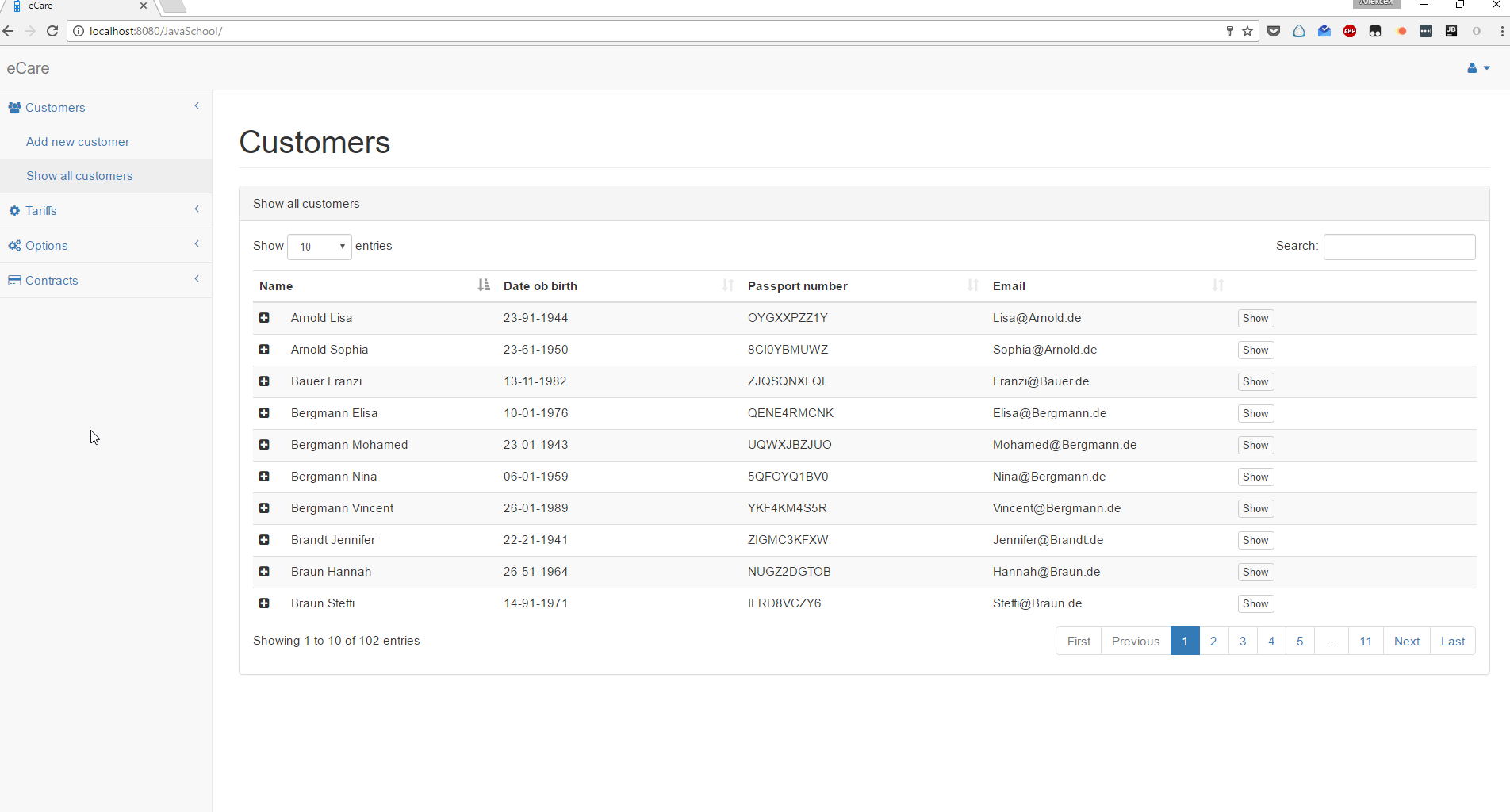
* findByName

**UserRepository**

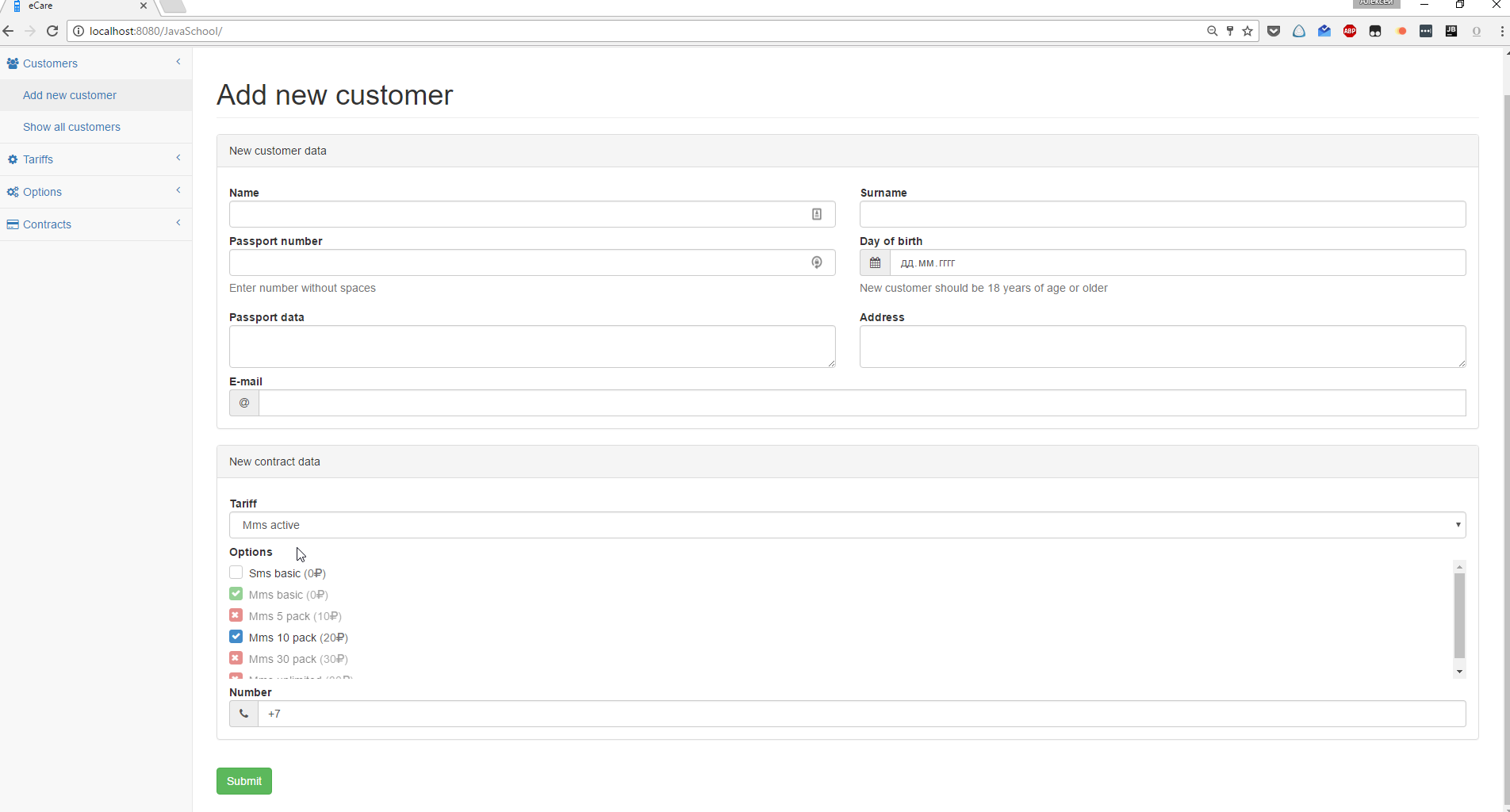
* findByEmail

# Screenshots

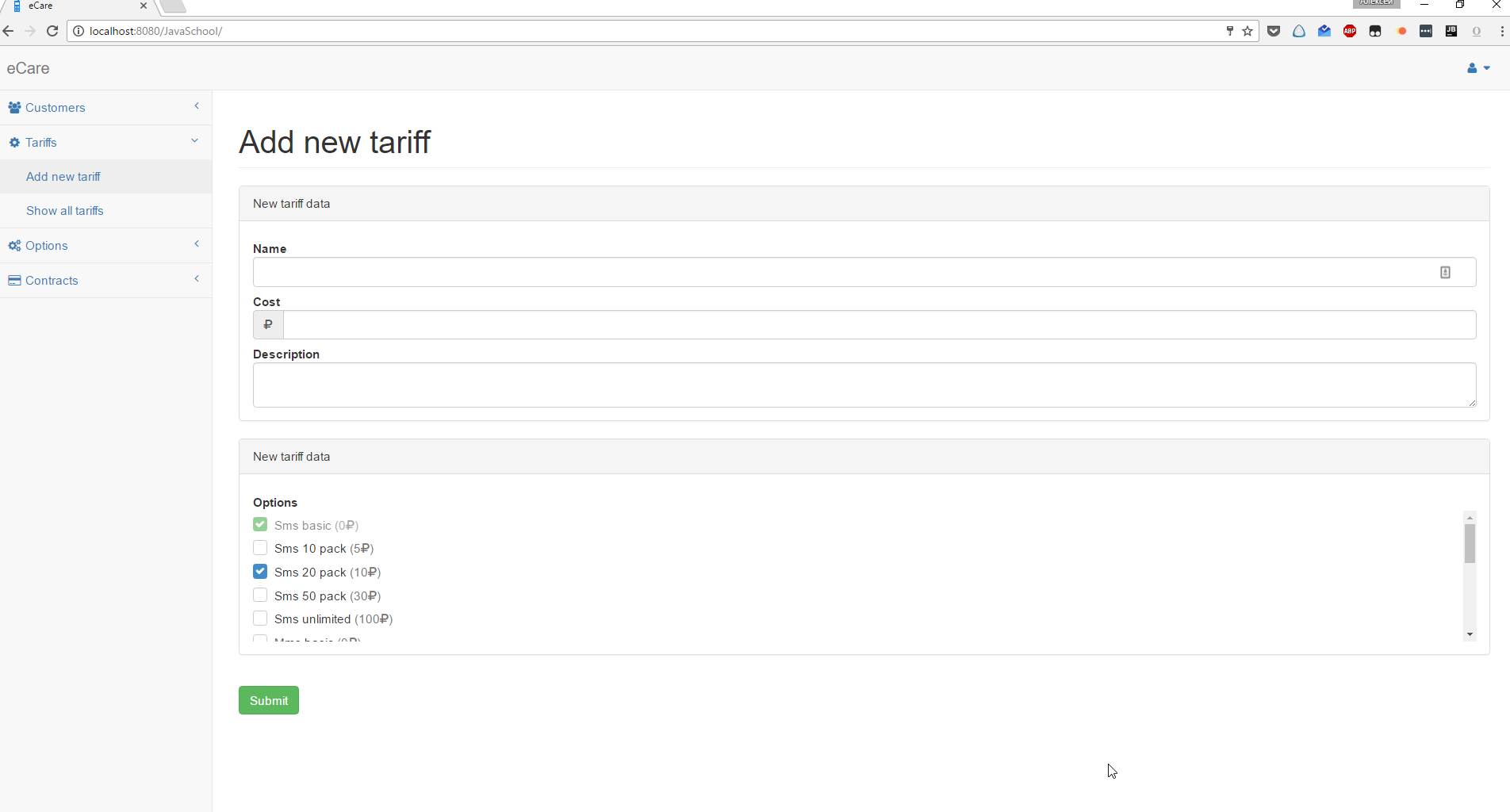
Customers table



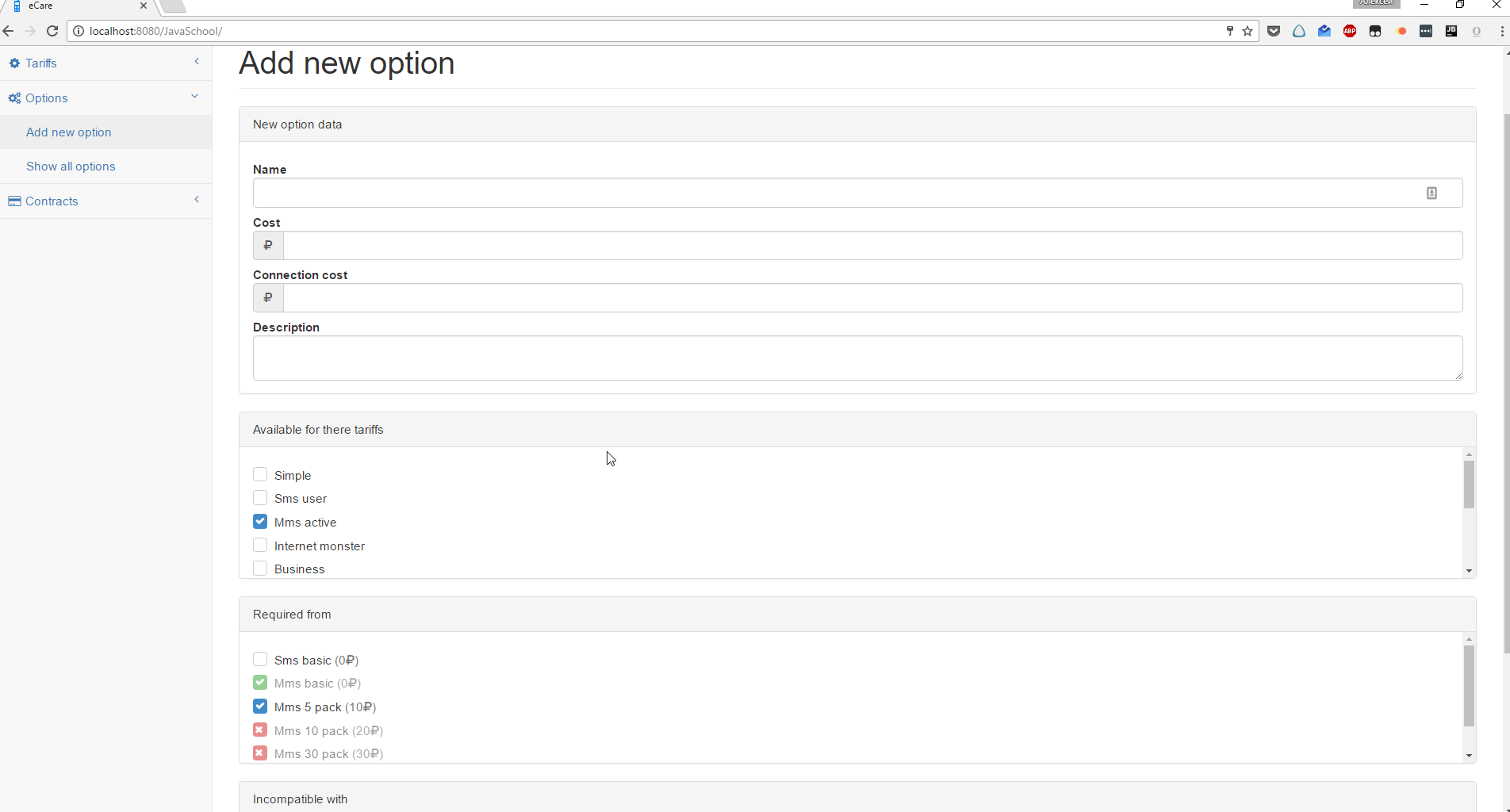
Customer register



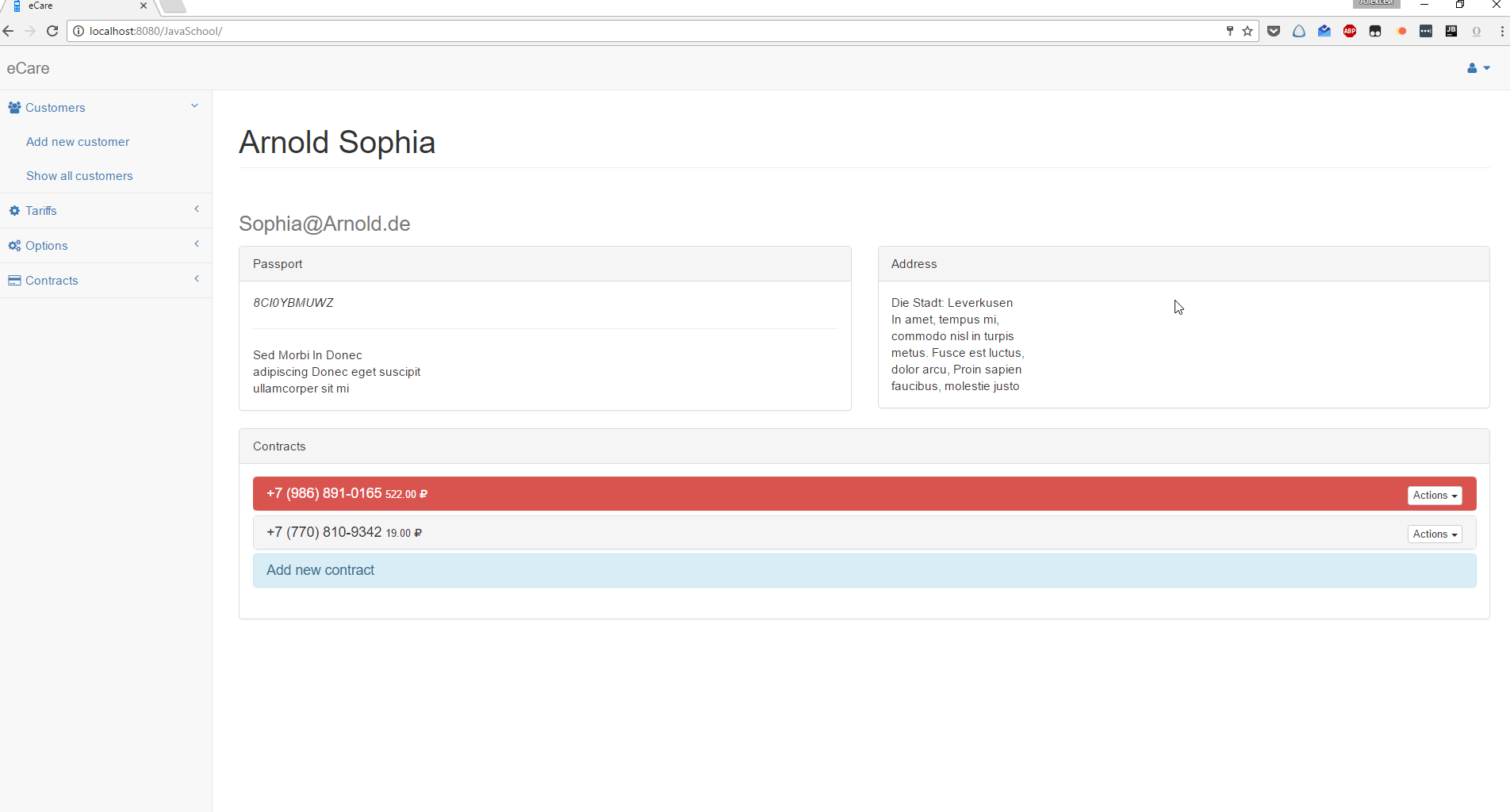
Tariff adding



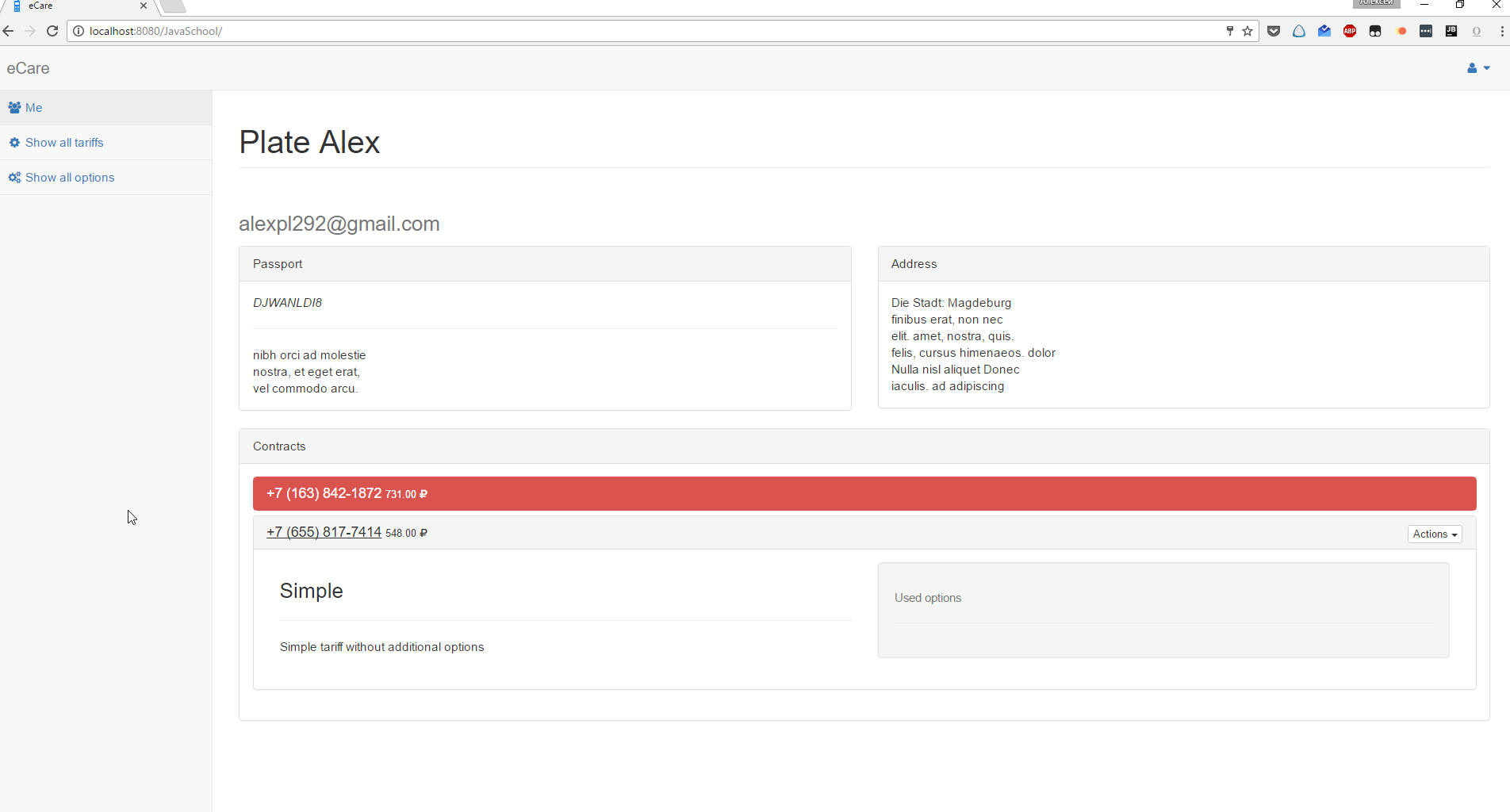
Option adding



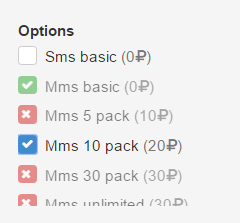
Customer page



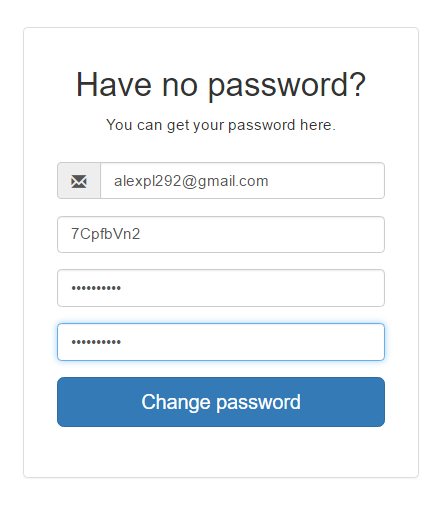
Customer page (customer view)



Options edit



Password restore



# Tests

Tests created for most complicated methods of business logic and user interface.  
UI tests grouped in methods by their functionality and contains both positive and negative scenarios.

Used frameworks: Selenium(with browsermob) and TestNG

## Backend tests

For ContractService

|  |  |
| --- | --- |
| Method | Description |
| testUpdateContractSameOptions | Updating contract with old options. Balance must be not changed |
| testUpdateContractOneChange | Updating contract with one of two used options. Balance must be changed partially |
| testUpdateContractAllChange | Updation contract with two of two used options. Balance must be changed. |

For OptionService

Testing relationships creating. Description has graphical explanation from JavaDoc

"#" (incompatible relationship)

-> (required relationship)

|  |  |
| --- | --- |
| Method | Description |
| testAddReqWithReq | Add option to require with required  new --> 1 --> 2  Expected:  new  / \  v v  1 --> 2 |
| testAddReqWithForb | Add option to required with forbidden  new -> 1 ## 2  Expected:  new  / #  v # 1 ### 2 |
| testAddReqReqAreForb | Add 2 required options, that are forbidden  new  / \  v v  1 ## 2  Expected: Exception errorCode = 2 |
| testAddReqForb | Add option, that is forbidden and required  new  / #  v#  1  Expected: Exception errorCode = 1 |
| testAddReqForbAreReq | new  / # v # 1 ---> 2  Expected: Exception errorCode = 3 |
| testAddForbWithReq | Add option to forbidden with required  new ### 1 --> 2  Expected:  new   # # # # 1 --> 2 |
| testAddForbWithReqMe | Add option to forbidden with required  new ### 1 <-- 2  Expected:  new  # # # # 1 <-- 2 |

## Frontend tests

Ajax requests tested by browsermob framework

|  |  |
| --- | --- |
| Method | Description |
| loadTestThisWebsite | Website loading |
| login | Test login |
| customerTable | Testing table with customer (Exists, not empty) |
| optionsTable | Testing table with options (Exists, not empty) |
| contractsTable | Testing table with contracts (Exists, not empty) |
| tariffsTable | Testing table with tariffs (Exists, not empty) |
| addTariff | Testing new tariff adding. Check wrong and correct data |
| addOption | Testing new option adding. Check wrong and correct data |
| addCustomer | Testing new customer adding. Check wrong and correct data |
| testCustomer | Test customer page. Blocking/unblocking, editing |
| testCustomerContracts | Test customers’ contracts. Deleting, creating. Wrong and correct data |

For frontend testing created “utils” methods

* clickMenu – finding and clicking menu item. Handle error by metis menu.
* waitJquery – wait for jQuery and javascript loading
* catchHar – wait for last har loading (browsermob)
* catchHarPost – wait and search for har with post request
* deleteEntity – delete created entity (by Located header in har response)
* hasAllClasses – test that element has all of given classes
* wv - syntax sugar for  
  wait.until(ExpectedConditions.visibilityOfElementLocated(by))
* wi – syntax sugar for  
  wait.until(ExpectedConditions.invisibilityOfElementLocated(by))

# Additional features

Application built and deployed via maven with wildfly plugin

Application logging configured with usage of Spring AOP. Hibernate and other frameworks logs stored in JavaSchool.log file. Application specified logs stored in js.log file.

All transferred data in rest service is logged.

Log example:

2016-10-18 11:56:24 DEBUG jslog:72 - Method Return value : [TariffDto{id=1, name='Simple', cost=0.00, description='Simple tariff without additional options', possibleOptions…

2016-10-18 11:56:24 DEBUG jslog:73 - -------------------------------------------------

2016-10-18 11:56:24 DEBUG jslog:96 - Method com.tsystems.javaschool.controllers.rest.TariffRest.loadOptions () execution time : 4 ms

2016-10-18 11:56:24 DEBUG jslog:100 - Args: [1]

2016-10-18 11:56:24 DEBUG jslog:102 - -------------------------------------------------

2016-10-18 11:56:24 DEBUG jslog:72 - Method Return value : []

2016-10-18 11:56:24 DEBUG jslog:73 - -------------------------------------------------

2016-10-18 11:56:31 DEBUG jslog:96 - Method com.tsystems.javaschool.controllers.rest.ContractRest.addNew () execution time : 120 ms

2016-10-18 11:56:31 DEBUG jslog:100 - Args: [ContractDto{id=null, number='+7 (432) 432-4324', isBlocked=null, …

2016-10-18 11:56:31 DEBUG jslog:102 - -------------------------------------------------

2016-10-18 11:56:31 DEBUG jslog:72 - Method Return value : <201 Created,ContractDto{id=188, number='+7 (432) 432-4324', isBlocked=0, … email='Lisa@Arnold.de', isBlocked=0, contracts=[]}, tariff=TariffDto{id=1, name='Simple', cost=0.00, description='Simple tariff without additional options', possibleOptions=[]}, balance=100.00, usedOptions=[]},{Location=[/rest/contracts/188]}>

2016-10-18 11:56:31 DEBUG jslog:73 - -------------------------------------------------

## Custom utils

**DTO** pattern used for data transferring between service and controller layers. Custom mapper implemented for dto-entity mapping.

**FillData** class created for filling database with data. Class fills database only if it’s empty.

**DataBaseValidator** contains all validation rules using database.

**EmailHelper** helperclass for sending emails.

**ErrorResponse** wrapper class (over HashMap) for error response representation.

**LoggingHandler** logger for application. Use Spring AOP

## Exceptions

**JSException** parent exception for all application specific exceptions

**NoEntityInDBException** thrown if database don’t contain entity with id. Used in databasevalidator class.

**OptionNotAvailableForTariffException** thrown if chosen option not available for chosen tariff.

**ResourceNotFoundException** thrown if REST resource is not found.

**UniqueFieldDuplicateException** thrown if entity with unique field already exists (for ex: Tariff with name “Simple”)

**WrongOptionConfigurationException** thrown if wrong combination of options is passed to persist.

## Sonar

All critical issues are closed (except 22 “field is marked "javax.validation.constraints.NotNull" but is not initialized in this constructor” that is marked as “won’t fix”)

81 % of code covered with documentation.

Technical debt can be closed in 2 days.

