Technical Solution Description

eCare mobile

Version 1.0

Last revised 10-06-2012

Status done

Author Andrei Makarevich

Imprint

|  |  |  |
| --- | --- | --- |
| **Issued by**  T-Systems RUS | | |
| **File name**  Technical solution description for eCare mobile.docx | **Document number**  not applicable | **Document name**  Technical solution description |
| **Version**  1.0 | **Last revised**  24-07-2016 | **Status**  done |
| **Author**  Andrei Makarevich | **Content checked by**  Andrei Makarevich  Saint Petersburg, 18.07.2016 | **Released by**  Andrei Makarevich  Saint Petersburg, 24.07.2016 |
| **Contacts**  Andrei Makarevich | **Telephone / fax**  8 (981) 946 56 66 | **E-Mail**  andrei.makarevich@t-systems.com |

Table of Contents

[1 Introduction 5](#_Toc457152694)

[2 Technologies and frameworks 6](#_Toc457152695)

[3 Additional features 7](#_Toc457152696)

[4 Database description 8](#_Toc457152697)

[5 Explication and implementation 11](#_Toc457152698)

[6 Modules 12](#_Toc457152699)

[7 UI 13](#_Toc457152700)

[8 Service layer 14](#_Toc457152701)

[9 Entities and DAO layer 17](#_Toc457152702)

[9.1 Entities 17](#_Toc457152703)

[9.2 DAO layer 19](#_Toc457152704)

[10 Screenshots of the application 21](#_Toc457152705)

[11 Junit tests 31](#_Toc457152706)

# Introduction

Mobile operator website is to be developed. Essential functionality includes plans and features management (CRUD and managing links and dependencies), processing customers and contracts for providing mobile services (each customer can have any amount of contracts with unique phone numbers), personal accounts where customers can block and unblock their contracts (only if respective contracts are not blocked by system’s manager). While proceeding with contract changes all data must be stored in basket on the client side. Only authenticated users with must be able to use personal account features. Only users with privileges of manager or admin must be able to use CRUD functions. Only users with privileges of admin must be able to change privileges.

As addition second application is to be developed. Essential functionality includes retrieving mobile plan usage statistics from the web service provided by the main application. Report in .pdf format must be generated using retrieved statistics.

# Technologies and frameworks

* Hibernate
* Spring MVC
* Spring Security
* HTML5, CSS3, JQuery

Also used in additional part:

* EJB
* JSF

# Additional features

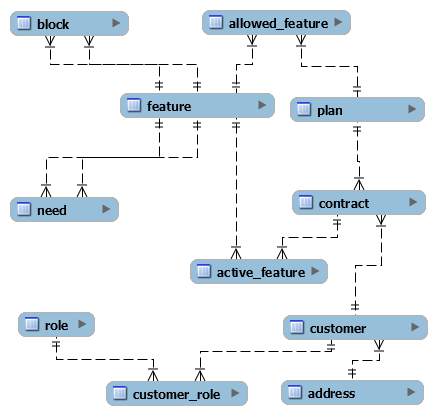
Passwords for new customers of the service are being delivered via SMS. Fresh password creation and sending is triggered when the first contract for this customer is created. Customer doesn’t have access to his profile at all until that.

# Database description

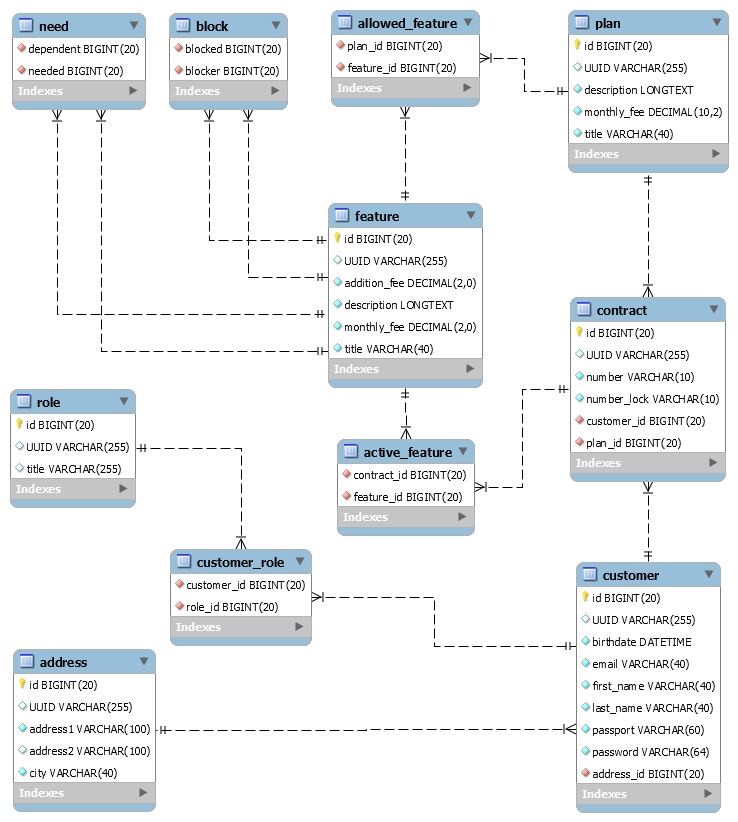
All database tables are listed in the table below.

|  |  |
| --- | --- |
| **Table** | **Description** |
| customer | Stores all users of application information. |
| contract | Stores information about contracts between customers and mobile service provider. |
| plan | Stores information about plans one and only one of which must be applied to each contract. |
| feature | Stores information about options that can be additionally applied to any contract. |
| address | Stores information about addresses where customers live. |
| role | Stores information about system roles for access restriction. |
| **Many-to-many support tables** | |
| block | Link from feature table to itself. Presents features being blocked by each other. |
| need | Link from feature table to itself. Presents situation when one of features depends on another. |
| allowed\_feature | Link from plan table to feature table. Presents feature allowed to be switched on when respective plan is activated for contract. |
| active\_feature | Link from contract table to feature table. Presents feature activated for repective contract. |
| customer\_role | Link from customer table to role table. Presents user having role. |

Minified database diagrams is show at Picture 4.1. Full database diagram with fields and types is shown at Picture 4.2.



Picture 4.1 – Database diagram (minified)



Picture 4.2 – Database diagram (full)

# Explication and implementation

**Customer** is a user of application. He can be a client of mobile service provider or a manager who works for it. Customer can have any number of contracts, but access to profile is only exists when at least one contract is owned.

**Profile** is special page where customer can manage his own contracts – change plan and options, block and unblock contracts.

**Contract** is document with unique phone number (10 digits) and it must have only one customer as its owner and only one mobile plan being active. It can have any amount of options being active but these options must be consistent and follow active plan restrictions and links between options.

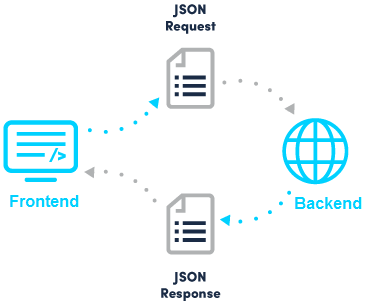
**Plan** is the base set of conditions applied to every contract. Basically, plan doesn’t allow any options but any option can be allowed to be used with this plan.

**Option** is a feature that allows to apply additional conditions to contract. Options can link each other in two ways: block each other, or be dependent one from another.

**Basket** is a special page that is only accessible when customer is in the middle of creation or editing of contract. All data about made changes are stored at the client side until customer will saves them.

# Modules

Application is designed using REST architecture (Picture 6.1).



Picture 6.1 – Interaction between client and server parts of application.

Backend provides REST API that can be used by any client capable of sending HTTP-requests. This architecture allows to make changes in backend and client separately (if API changes are not caused by these changes), or in prospect develop mobile application for any platform using just the same backend.

Table below contains short REST API description. There are some basic CRUDs for main entities and a lot of different state changing methods.

|  |  |  |
| --- | --- | --- |
| **URL** | **Method** | **Description** |
| ***Options CRUD*** | | |
| /option | POST | Creates new option. |
| /option | GET | Returns all options. |
| /option/{id} | GET | Returns single option with id = {id}. |
| /option | PUT | Updates existing option. |
| /option/{id} | DELETE | Deletes option with id = {id}. |
| ***Plans CRUD*** | | |
| /plan | POST | Creates new plan. |
| /plan | GET | Returns all plans. |
| /plan/{id} | GET | Returns single plan with id = {id}. |
| /plan | PUT | Updates existing plan. |
| /plan/{id} | DELETE | Deletes plan with id = {id}. |
| ***Customers CRUD*** | | |
| /customer | POST | Creates new customer. |
| /customer/{id} | GET | Returns single customer with id = {id}. |
| /customer | PUT | Updates existing customer. |
| /customer/{id} | DELETE | Deletes customer with id = {id}. |
| ***Contracts CRUD*** | | |
| /contract | POST | Creates new contract. |
| /contract/{id} | GET | Returns single contract with id = {id}. |
| /contract | PUT | Updates existing contract. |
| ***Miscellaneous*** | | |
| /allow | PUT | Changes allowed feature of plan. |
| /available | GET | Returns data about what features are available for contract. It gives count on plan's available features and currently enabled features. |
| /contract | PUT | Updates existing contract. |
| /customer/lock | PUT | Locks/unlocks customers contracts (entry point for managers). |
| /link | PUT | Changes link between options. |
| /profile | PUT | Returns profile information. |
| /profile/lock | PUT | Locks/unlocks customers contracts (entry point for own contracts). |
| /profile/contract | GET | Returns single contract data. |
| /profile/contract | PUT | Updates single existing contract data. |
| /report | GET | Returns statistics information about plan. |
| /role/activate | PUT | Activates/deactivates role for customer. |

Interaction between different levels of application is presented at Picture 6.2. Client side consumes REST API. Controllers call appropriate service method. Services interact with objects using data access objects. DAO use entities for storing database objects data.

Controllers

Services

DAO

Model

O

REST API

Client

Picture 6.2 – Application levels interaction.

# User interface

Dwgrnmhfj

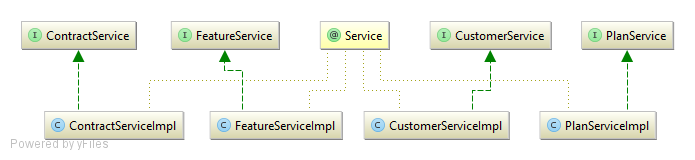
# Service layer

Business logic is represented by four services:

* ContractService – containing contracts processing logic;
* FeatureService – containing options processing logic;
* CustomerService – containing customers processing logic;
* PlanService – containing plans processing logic.

These services are described by interfaces and have Spring-specific implementation. All methods in service classes are annotated as transactional. All changes are being committed automatically at the end of successful method execution and rolled back in case of error.

Diagram with service layer structure is presented on Picture 8.1.



Picture 8.1 – Service layer structure

Listing of all methods of services with short description is available in tables below.

|  |  |
| --- | --- |
| **ContractService** | |
| **Method name** | **Description** |
| saveNewContract | Saves new contract and in case of need triggers sending sms with password. |
| saveContract | Saves contract (new or not). |
| getContract | Searches for single contract by id. |
| changeLock | Changes lock state for contract. |

|  |  |
| --- | --- |
| **CustomerService** | |
| **Method name** | **Description** |
| saveCustomer | Saves customer (new or not). |
| getCustomer | Searches customer by id. |
| getCustomerByEmail | Searches customer by email. |
| findCustomers | Searches customers by name and phone number. |
| deleteCustomer | Deletes customer. |
| activateRole | Activates or deactivates role for customer. |

|  |  |
| --- | --- |
| **FeatureService** | |
| **Method name** | **Description** |
| saveFeature | Saves option (new or not). |
| getFeature | Searches option by id. |
| getAllFeatures | Searches for all options. |
| getListedFeatures | Searches for options with ids contained in passed list. |
| getAvailableFeatures | Resolves options available for contract with some plan and some already activated options. |
| deleteFeature | Deletes option by id. |
| changeLinkedFeature | Updates link between two options. |

|  |  |
| --- | --- |
| **PlanService** | |
| **Method name** | **Description** |
| savePlan | Saves plan (new or not). |
| getPlan | Searches plan by id. |
| getAllPlans | Searches for all plans. |
| deletePlan | Deletes plan. |
| changeAllowedFeature | Changes allowed option status for plan. |
| getFeatureUsers | Calculates and returns statistics information about plan. |

# Entities and DAO layer

## Entities

Abstract Entity is a base for all other entities. It provides an equals and hashcode that will always work correctly in all circumstances. This avoids frequent errors related to the implementation of those same methods.

|  |  |
| --- | --- |
| **Column** | **Description** |
| id | Automatically generated primary key |
| UUID | Unique value for every entity that is used for providing equals and |

Role – represents access role in system. Only two predefined roles exist at the moment, but separate entity is designed to make access system to be extendable.

|  |  |
| --- | --- |
| **Column** | **Description** |
| title | Automatically generated primary key |

Feature – represents option that can be applied to contract in addition to plan. One contract can have any amount of options being active simultaneously. Options may block each other from usage or depend from other options.

|  |  |
| --- | --- |
| **Column** | **Description** |
| title | Title for the option |
| description | Description for the option |
| additionFee | Cost of switching option on |
| monthlyFee | Cost per month of using option |
| blockers | List of options that block this option |
| blockedFeatures | List of options blocked by this option |
| neededFeatures | List of options that needed for this option |
| dependentFeatures | List of options that dependent from this option |

Plan – represents mobile plan that (one and only one) must be applied to each contract. Plan may have any options allowed to add to contracts with this plan.

|  |  |
| --- | --- |
| **Column** | **Description** |
| title | Title for the plan |
| description | Description for the plan |
| monthlyFee | Cost per month of using option |
| allowedFeatures | List of options that are allowed for this plan |

Address – represents address of customers.

|  |  |
| --- | --- |
| **Column** | **Description** |
| address1 | Address line 1 |
| address2 | Address line 2 |
| city | City part of address |

Customer – represents customer of mobile operator. Each customer can have any amount of contracts.

|  |  |
| --- | --- |
| **Column** | **Description** |
| firstName | First name |
| lastName | Last name |
| birthdate | Date of birth |
| passport | Passport data |
| email | Email of customer |
| password | Hash of password |
| contracts | List of contracts owned by customer |
| address | Address of customer |
| roles | List of roles applied to customer |

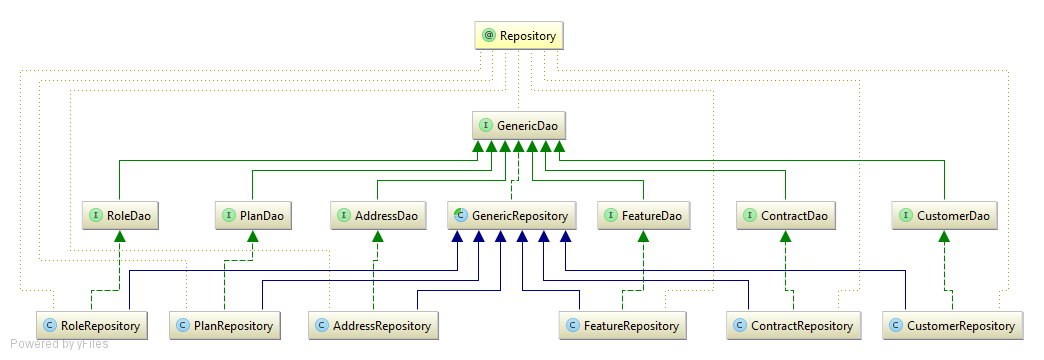
Contract – contract between mobile operator and customer with unique phone number.

|  |  |
| --- | --- |
| **Column** | **Description** |
| number | Unique phone number |
| customer | Customer who owns this contract |
| plan | Plan applied to contract |
| activeFeatures | Options activated for contract |
| numberLock | State of number (can be unlocked, locked by user, locked by manager) |

## DAO layer

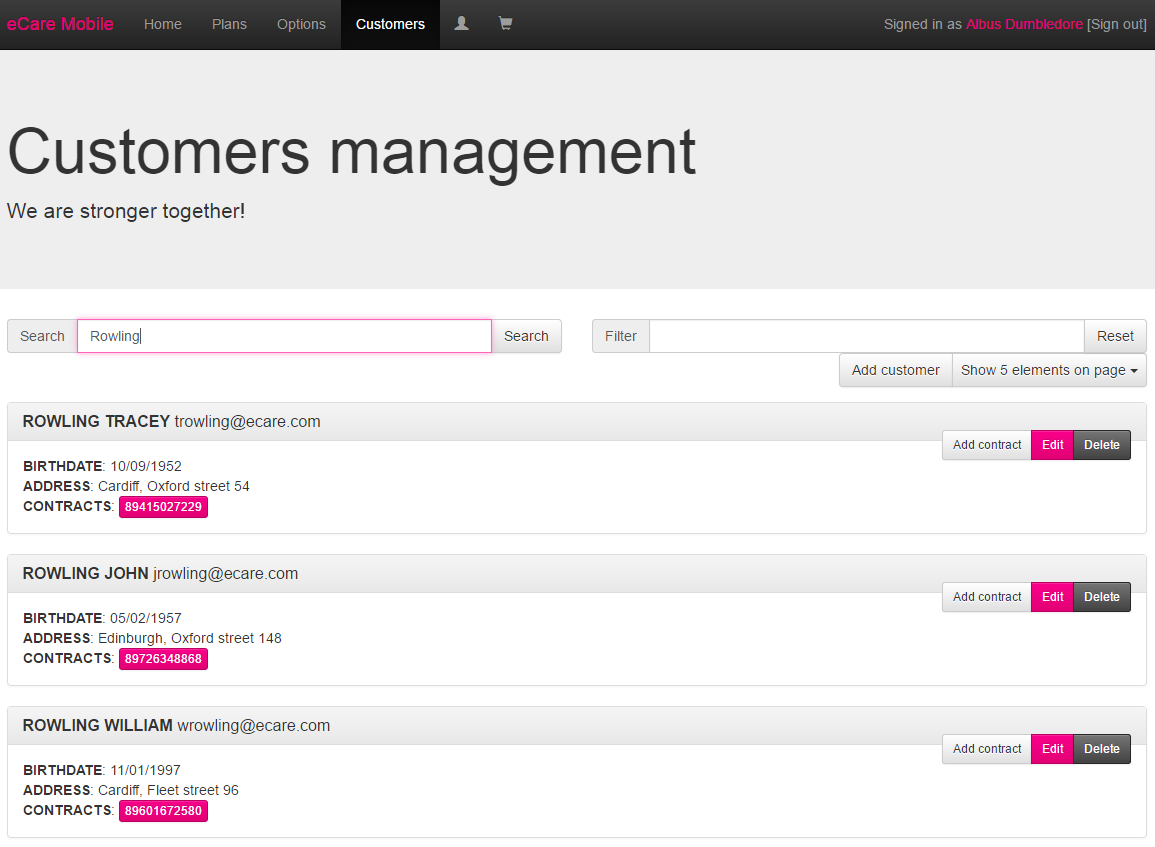
DAO layer is designed to be generified: interface GenericDao contains some methods for operation over entities that are common for all types of entities. This methods are implemented in GenericRepository class that is Spring specific implementation.

Other entity-specific methods are implemented in specific classes. Diagram of DAO layer structure is presented on Picture 9.1.

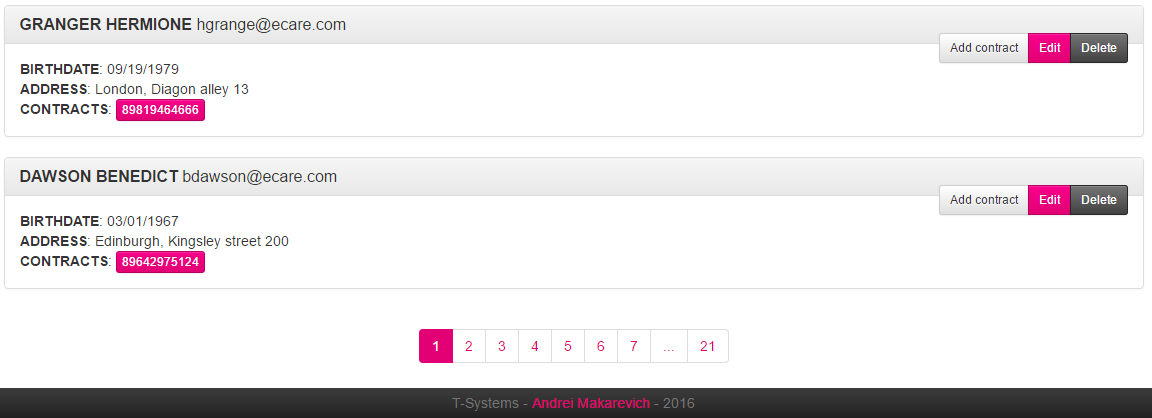


Picture 9.1 – DAO layer structure

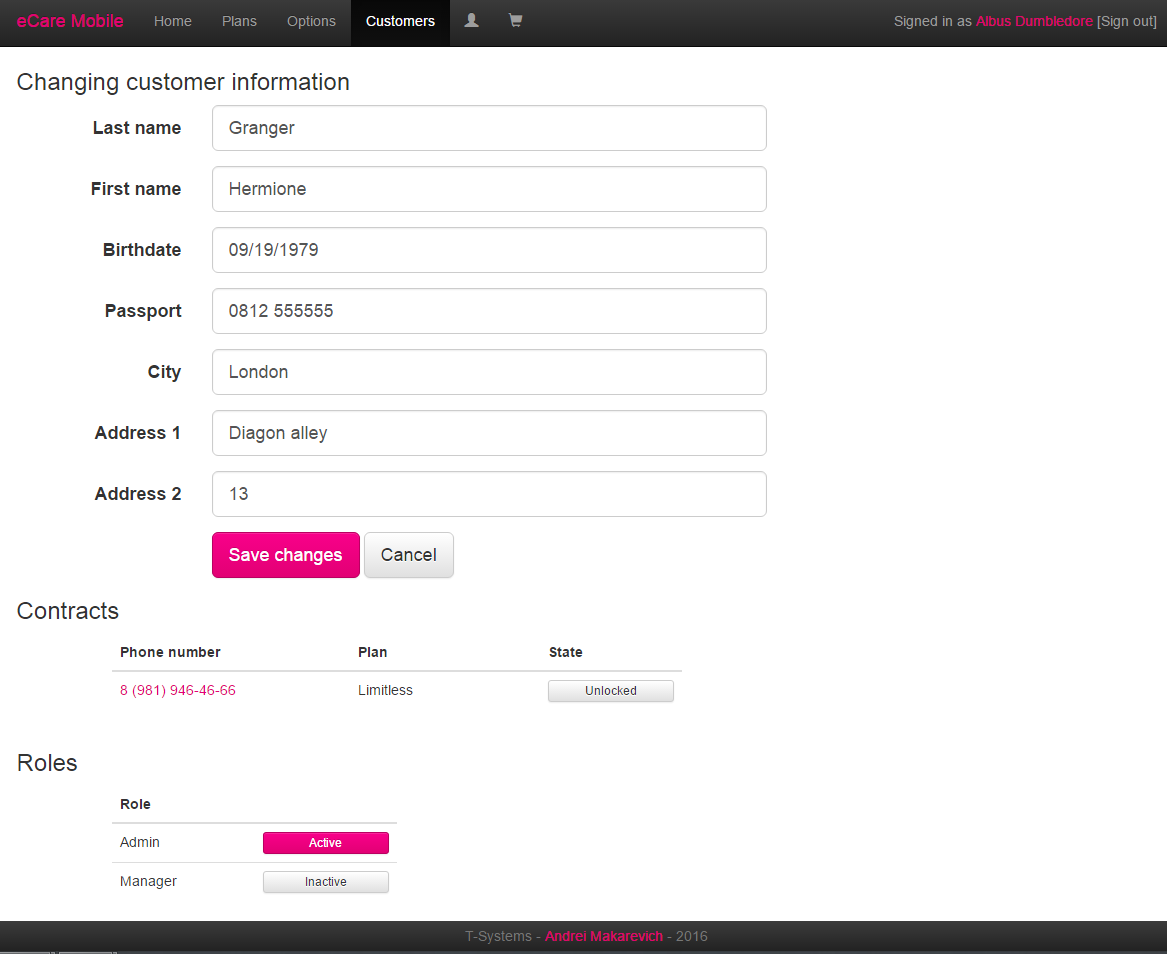
# Screenshots of the application



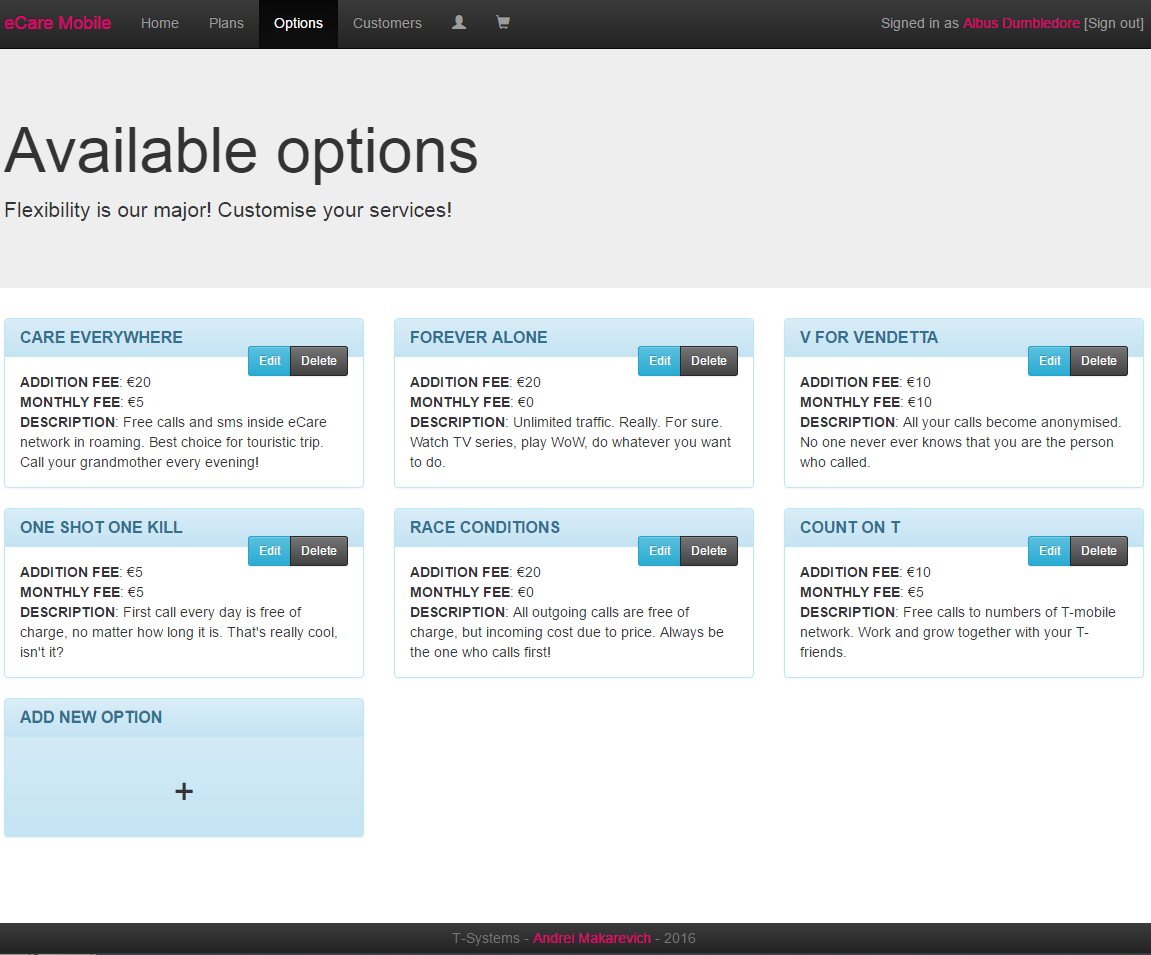
Picture 10.1 – Customers page top



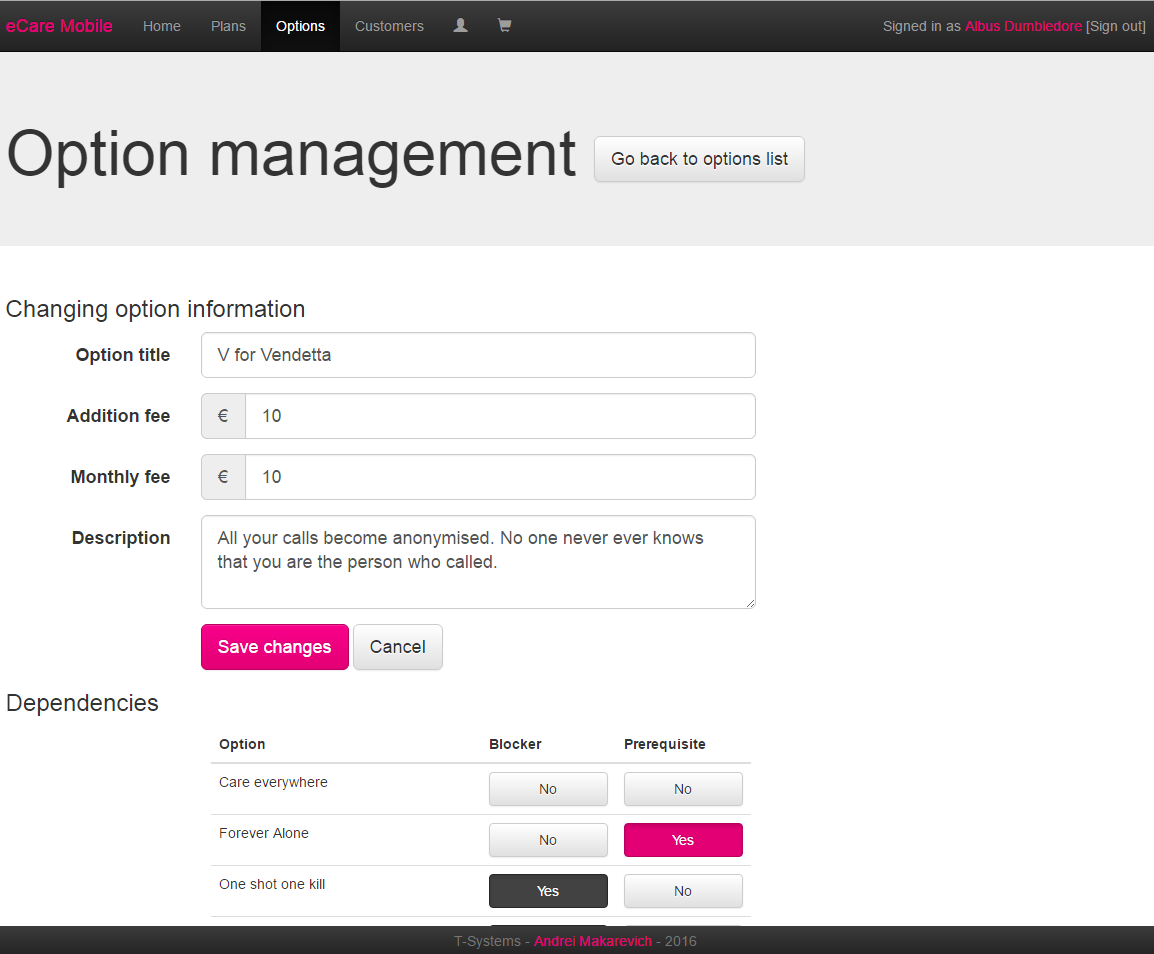
Picture 10.2 – Customers page bottom



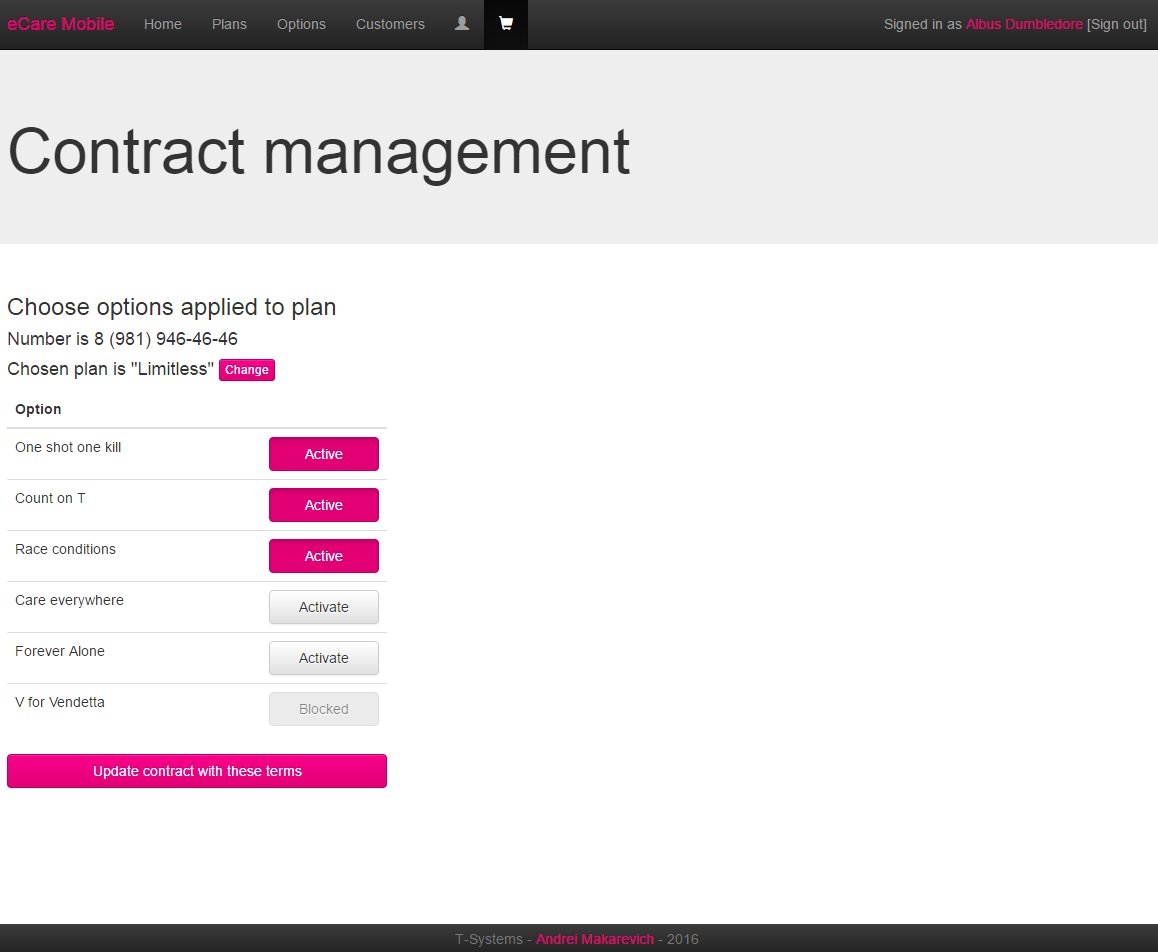
Picture 10.3 – Customers information editing page



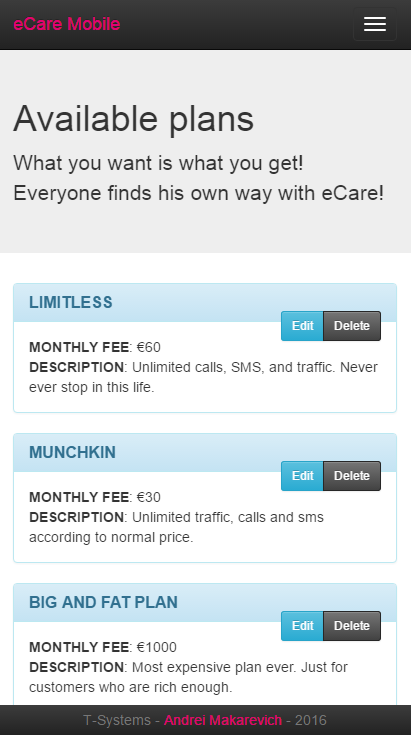
Picture 10.4 – Options page (manager access)



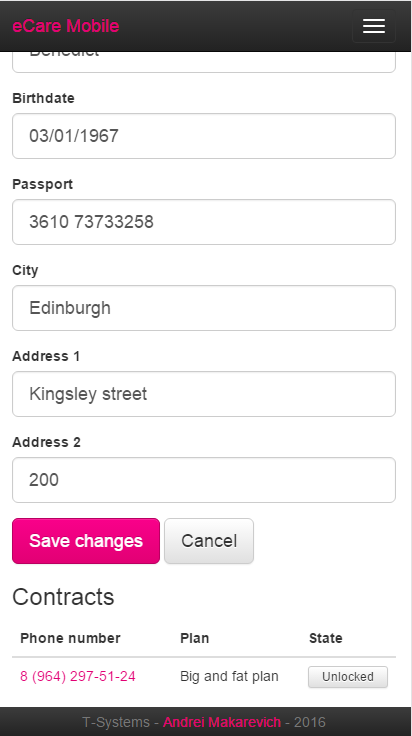
Picture 10.5 – Option editing page



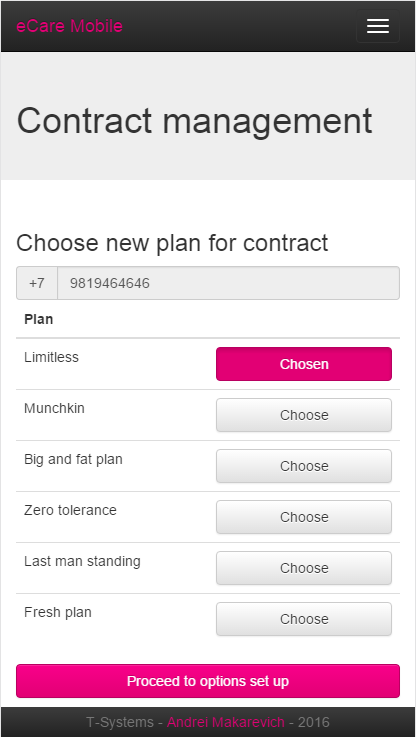
Picture 10.6 – Contract editing page (basket)



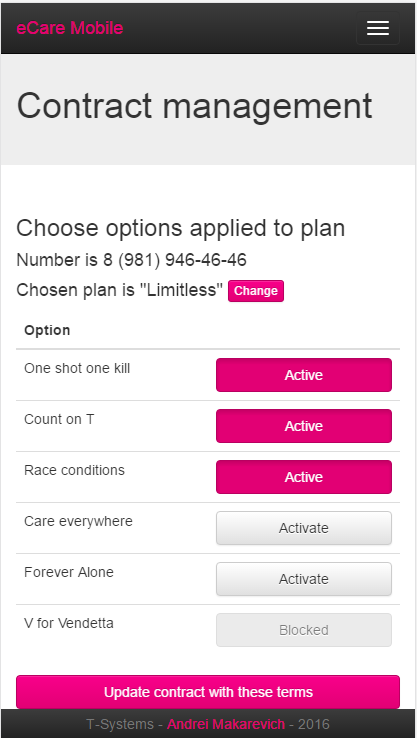
Picture 10.7 – Plans page (mobile view)



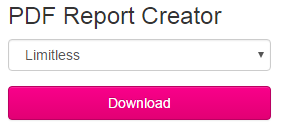
Picture 10.8 – Customer information editing page (mobile view)



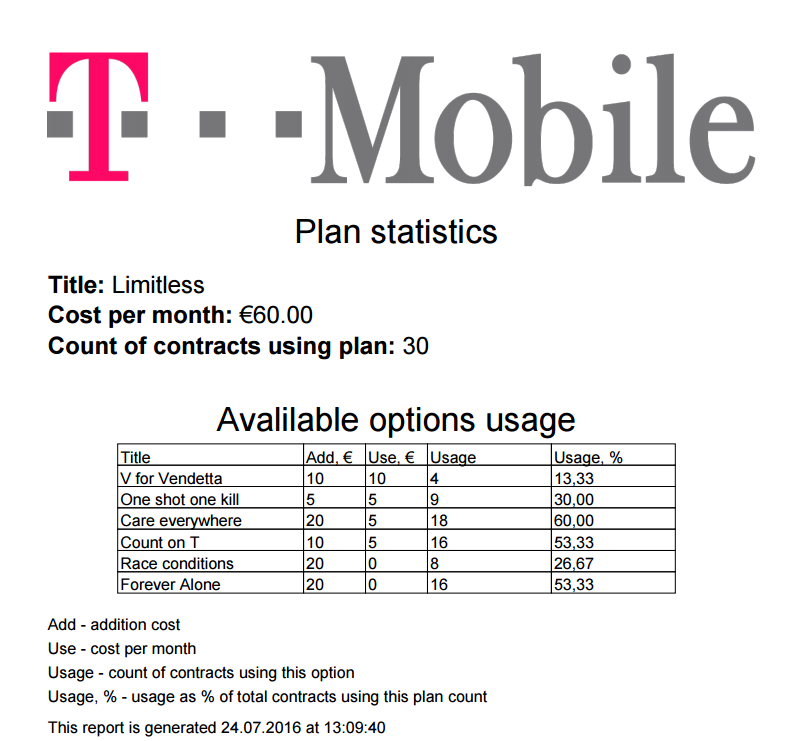
Picture 10.9 – Choosing plan for contract page (mobile view)



Picture 10.10 – Choosing options for contract page (mobile view)



Picture 10.11 – PDF-report creator page



Picture 10.12 – PDF-report example

# Junit tests

|  |  |
| --- | --- |
| **PlanServiceTest** | |
| **Test case name** | **What does it test** |
| savePlanNew | That new plan with valid data is saved successfully. |
| savePlanExisting | That existing plan valid data for update is saved successfully. |
| savePlanWithNonexistentId | That attempt to save plan with ID that doesn’t exist causes IllegalArgumentException. |
| savePlanWithBlankTitle | That attempt to save plan with blank title causes IllegalArgumentException. |
| savePlanWithTooLongTitle | That attempt to save plan with title that has more then 40 characters causes IllegalArgumentException. |
| savePlanWithBlankDescription | That attempt to save plan with blank description causes IllegalArgumentException. |
| savePlanWithTooLongDescription | That attempt to save plan with description that has more then 1000 characters causes IllegalArgumentException. |
| savePlanWithNullMonthlyFee | That attempt to save plan with null value for monthly fee causes IllegalArgumentException. |
| savePlanWithNegativeMonthlyFee | That attempt to save plan with negative value for monthly fee causes IllegalArgumentException. |
| getPlan | That plan can be successfully retrieved by ID. |
| getPlanWithNonexistentId | That attempt to retrieve plan with ID that doesn’t exist causes IllegalArgumentException. |
| getAllPlans | That all plans can be successfully retrieved. |
| deletePlan | That plan can be successfully deleted by ID. |
| changeAllowedFeature | That allowed feature state for plan can be successfully changed. |
| changeAllowedFeatureWithNonexistentPlanId | That attempt to change allowed feature state using plan ID that doesn’t exist causes IllegalArgumentException. |
| changeAllowedFeatureWithNonexistentFeatureId | That attempt to change allowed feature state using feature ID that doesn’t exist causes IllegalArgumentException. |

|  |  |
| --- | --- |
| **FeatureServiceTest** | |
| **Test case name** | **What does it test** |
| saveFeatureNew | That new option with valid data is saved successfully. |
| saveFeatureExisting | That existing option valid data for update is saved successfully. |
| saveFeatureWithNonexistentId | That attempt to save option with ID that doesn’t exist causes IllegalArgumentException. |
| saveFeatureWithBlankTitle | That attempt to save option with blank title causes IllegalArgumentException. |
| saveFeatureWithTooLongTitle | That attempt to save option with title that has more then 40 characters causes IllegalArgumentException. |
| saveFeatureWithBlankDescription | That attempt to save option with blank description causes IllegalArgumentException. |
| saveFeatureWithTooLongDescription | That attempt to save option with description that has more then 1000 characters causes IllegalArgumentException. |
| saveFeatureWithNullMonthlyFee | That attempt to save option with null value for monthly fee causes IllegalArgumentException. |
| saveFeatureWithNegativeMonthlyFee | That attempt to save option with negative value for monthly fee causes IllegalArgumentException. |
| saveFeatureWithNullMonthlyFee | That attempt to save option with null value for monthly fee causes IllegalArgumentException. |
| saveFeatureWithNegativeMonthlyFee | That attempt to save option with negative value for monthly fee causes IllegalArgumentException. |
| getFeature | That option can be successfully retrieved by ID. |
| getFeatureWithNonexistentId | That attempt to retrieve option with ID that doesn’t exist causes IllegalArgumentException. |
| getAllFeatures | That all options can be successfully retrieved. |
| getListedFeaturesExact | That listed options can be successfully retrieved by string with ids that only contains existing options. |
| getListedFeaturesMore | That listed options can be successfully retrieved by string with ids that contains not only existing options. |
| getListedFeaturesWithIngalidIds | That attempt to retrieve listed options with invalid value for ids causes IllegalArgumentException. |
| getListedFeaturesWithNullIds | That attempt to retrieve options with null value for ids causes IllegalArgumentException. |
| getAvailableFeatures | That available options can be successfully resolved and retrieved. |
| getAvailableFeaturesWithNullPlanId | That attempt to retrieve available options with null value for plan ID causes IllegalArgumentException. |
| getAvailableFeaturesWithNonexistentPlanId | That attempt to retrieve available options with nonexistent value for plan ID causes IllegalArgumentException. |
| getAvailableFeaturesWithNullIds | That attempt to retrieve available options with null value for ids causes IllegalArgumentException. |
| getAvailableFeaturesWithInvalidIds | That attempt to retrieve available options with invalid value for ids causes IllegalArgumentException. |
| deleteFeature | That option can be successfully deleted by ID. |
| changeLinkedFeatureBlock | That link of block type between two options can be successfully created. |
| changeLinkedFeatureUnblock | That link of block type between two options can be successfully deleted. |
| changeLinkedFeatureNeed | That link of need type between two options can be successfully created. |
| changeLinkedFeatureNoneed | That link of need type between two options can be successfully deleted. |
| changeLinkedFeatureWithNullFeature1 | That attempt to link options with null value for feature1 causes IllegalArgumentException. |
| changeLinkedFeatureWithNonexistentFeature1 | That attempt to link options with nonexistent ID value for feature1 causes IllegalArgumentException. |
| changeLinkedFeatureWithNullFeature2 | That attempt to link options with null value for feature2 causes IllegalArgumentException. |
| changeLinkedFeatureWithNonexistentFeature2 | That attempt to link options with nonexistent ID value for feature2 causes IllegalArgumentException. |
| changeLinkedFeatureWithNullBlock | That attempt to link options with null value for block causes IllegalArgumentException. |
| changeLinkedFeatureWithNullLinked | That attempt to link options with null value for linked causes IllegalArgumentException. |
| changeLinkedFeatureLinkToItself | That attempt to link option to itself causes IllegalArgumentException. |

|  |  |
| --- | --- |
| **CustomerServiceTest** | |
| **Test case name** | **What does it test** |
| saveCustomerNew | That new customer with valid data is saved successfully. |
| saveCustomerExisting | That existing customer valid data for update is saved successfully. |
| saveCustomerWithNonexistentId | That attempt to save customer with ID that doesn’t exist causes IllegalArgumentException. |
| saveCustomerWithBlankLastName | That attempt to save customer with blank value for last name causes IllegalArgumentException. |
| saveCustomerWithBlankFirstName | That attempt to save customer with blank value for first name causes IllegalArgumentException. |
| saveCustomerWithBlankPassport | That attempt to save customer with blank value for passport causes IllegalArgumentException. |
| saveCustomerWithBlankCity | That attempt to save customer with blank value for city title causes IllegalArgumentException. |
| saveCustomerWithBlankAddress1 | That attempt to save customer with blank value for first lane of address causes IllegalArgumentException. |
| saveCustomerWithLastNameTooLong | That attempt to save customer with last name value that has more then 40 characters causes IllegalArgumentException. |
| saveCustomerWithFirstNameTooLong | That attempt to save customer with first name value that has more then 40 characters causes IllegalArgumentException. |
| saveCustomerWithPassportTooLong | That attempt to save customer with passport value that has more then 60 characters causes IllegalArgumentException. |
| saveCustomerWithCityTooLong | That attempt to save customer with city title value that has more then 40 characters causes IllegalArgumentException. |
| saveCustomerWithAddress1TooLong | That attempt to save customer with first lane of address value that has more then 100 characters causes IllegalArgumentException. |
| saveCustomerWithAddress2TooLong | That attempt to save customer with second lane of address value that has more then 100 characters causes IllegalArgumentException. |
| saveCustomerWithNullBirthate | That attempt to save customer with null value for birthdate causes IllegalArgumentException. |
| getCustomer | That customer can be successfully retrieved by ID. |
| getCustomerWithNullId | That attempt to retrieve plan with null ID causes IllegalArgumentException. |
| getCustomerWithNonexistentId | That attempt to retrieve customer with ID that doesn’t exist causes IllegalArgumentException. |
| getCustomerByEmail | That customer can be successfully retrieved by email. |
| getCustomerByEmailWithNullEmail | That attempt to retrieve plan with null email causes IllegalArgumentException. |
| getCustomerByEmailWithNonexistentEmail | That attempt to retrieve customer with email that doesn’t exist causes IllegalArgumentException. |
| findCustomers | That customers can be successfully retrieved by query string with search in name and phone numbers. |
| deleteCustomer | That customer can be successfully deleted by ID. |
| activateRoleActivate | That role can be successfully activated for customer. |
| activateRoleDeactivate | That role can be successfully deactivated for customer. |

|  |  |
| --- | --- |
| **ContractServiceTest** | |
| **Test case name** | **What does it test** |
| saveNewContract | That new contract with valid data is saved successfully. |
| saveNewContractWithExitingNumber | That attempt to save new contract with existing number causes IllegalArgumentException. |
| saveNewContractWithBlankNumber | That attempt to save new contract with blank value for number causes IllegalArgumentException. |
| saveNewContractWithNullNumber | That attempt to save new contract with null value for number causes IllegalArgumentException. |
| saveNewContractWithNullPlan | That attempt to save new contract with null value for plan causes IllegalArgumentException. |
| saveNewContractWithNonexistentPlan | That attempt to save new contract with plan that doesn’t exist causes IllegalArgumentException |
| saveNewContractWithNullCustomer | That attempt to save new contract with null value for customer causes IllegalArgumentException. |
| saveNewContractWithNonexistentCustomer | That attempt to save new contract with customer that doesn’t exist causes IllegalArgumentException. |
| saveContract | That existing plan valid data for update is saved successfully. |
| saveContractWithNonexistentId | That attempt to save customer with ID that doesn’t exist causes IllegalArgumentException. |
| saveContractWithNullPlan | That attempt to save contract with null value for plan causes IllegalArgumentException. |
| saveContractWithNonexistentPlan | That attempt to save contract with plan that doesn’t exist causes IllegalArgumentException |
| saveCustomerWithNullBirthate | That attempt to save customer with null value for birthdate causes IllegalArgumentException. |
| getContract | That contract can be successfully retrieved by ID. |
| getContractOwn | That user’s own contract can be successfully retrieved by ID if email is correct. |
| getContractOwnWithWrongEmail | That not own contract can’t be successfully retrieved by ID if email is wrong and the attempt causes IllegalArgumentException. |
| changeLockLock | That lock can be successfully created. |
| changeLockUnlock | That lock can be successfully deleted. |
| changeLockUnlockOwnUserlocked | That userlock can be successfully created. |
| changeLockUnlockOwnLocked | That userlock can be successfully deleted. |
| changeLockUnlockOwnLockedWrongEmail | Attempt to delete lock that doesn’t belong to user with passed email causes IllegalArgumentException. |

|  |  |
| --- | --- |
| **Utility methods tests** | |
| **Test case name** | **What does it test** |
| sha256 | That sha256 hash is being generated right. |
| generatePassword | That randomly generated passwords are different and of appropriate length. |
| getStringOfLength | That generated filler string is of appropriate length. |

|  |  |
| --- | --- |
| **FrontendWithRestrictedAccessTest (Selenium)** | |
| **Test case name** | **What does it test** |
| unauthorized | That unauthorized user can access unrestricted pages and can open and close login form. |
| authorizedUser | That user can’t log in with wrong credentials, that user without manager or admin rights can successfully log in and is able to access and manage his profile. |
| authorizedManager | That user with manager rights can successfully log in and is able to manage options, plans, customers. |
| authorizedAdmin | That user with admin rights can successfully log in and is able to change access rights to users. |