Gordeev Aleksei
Aleksei.a.Gordeev@gmail.com

# CONTENTS

# Task

It is required to create a web-application that simulates the functionality of mobile network operator information system. Details of subject area and technical requirements are given below.

## Subject area

There are following kinds of entities:

*Tariff*

- Title
- Price
- List of available options

*Option*

- Title
- Price
- Cost of connection

*Client*

- Name
- Last name
- Birth date
- Passport data
- Address
- List of contracts (telephones numbers of client)
- E-mail
- Password

*Contract*

- Telephone number
- Tariff
- Connected options for tariff

The application must provide the following functionality:

For clients

- Browse of the contract in a personal cabinet;
- Browse of all available tariffs and change a tariff;
- Browse of all available options for tariff, add new options, disable the existing ones;
- Lock / Unlock of a number (if a number were locked, it is not allowed to change the tariff and options; if a number was locked not by a client, he can't unlock it);

For employees

- Conclusion of contract with a new client: the choice of a new telephone number with the tariff and options. The phone number should be unique.
- Browse of all clients and contracts;
- Lock and unlock of client's contract;
- Looking up client's contract by phone number;
- Change tariff, add and remove options of contract;
- Add new tariffs, remove existing one;
- Add / remove option available for specific tariff;
- Option management: options may be inconsistent each other or ought to be added with certain options, employee adds and removes these rules.

On the each page during operating with contract before saving the changes the basket must be displayed. It should contain the client's choices.

# Solution description

## Overview

The application runs under WildFly 9.0 Application Server. It has a common three-layered design consisting of presentation, services and persistence layers.

The persistence (DAO) layer is used for basic (create, read, update and delete) operations on the database. It is based on Java Persistence API. The Application Server provides the implementation of it by the Hibernate.

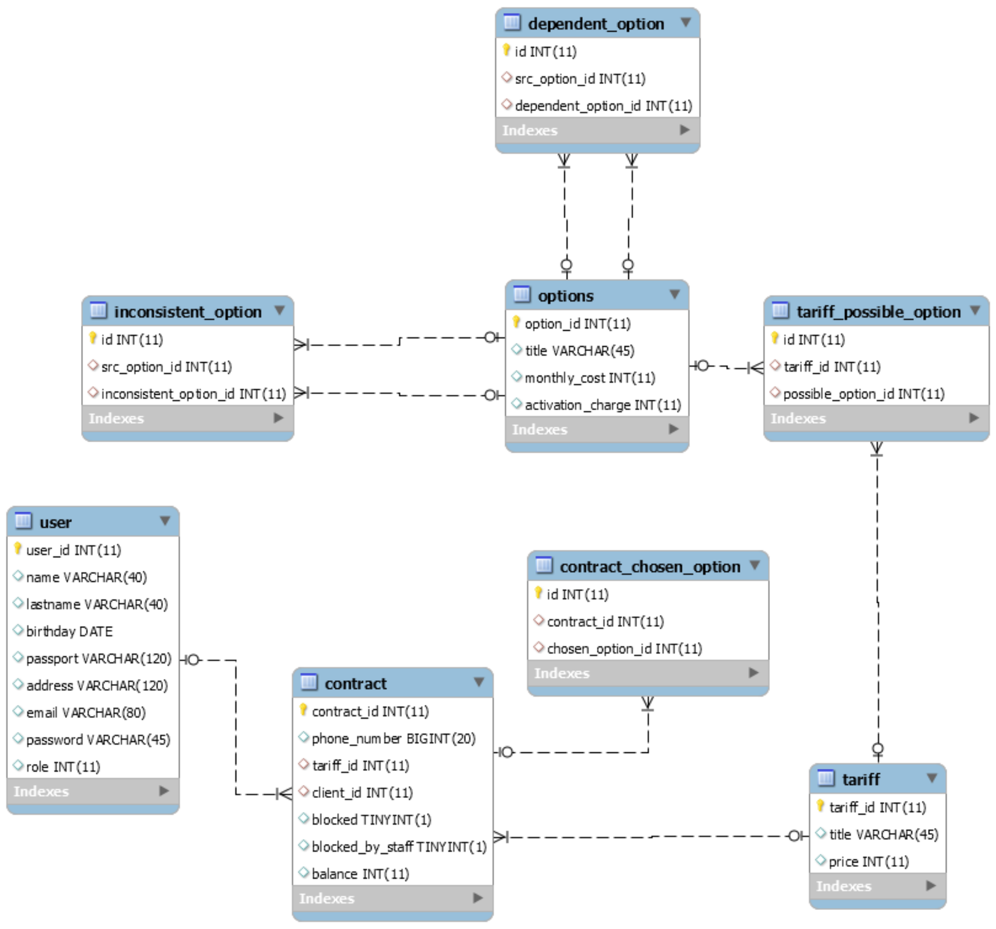
The application runs with a help of the Spring Framework. To be exact there are Spring Core and Spring MVC here. The Spring Core helps to solve the task of injecting of dependencies and transaction handling. Spring MVC is responsible for presentation layer. Spring Web-controller resolves view to a regular JSP.

## Data source and transactions

The Application Server uses MySQL as a database management system.

The Application Server is configured in such a way that it creates and managed the Entity manager. And the Entity manager is obtained from JNDI. Spring framework is also responsible for transactions handling. All the public methods in service layer are transactional.

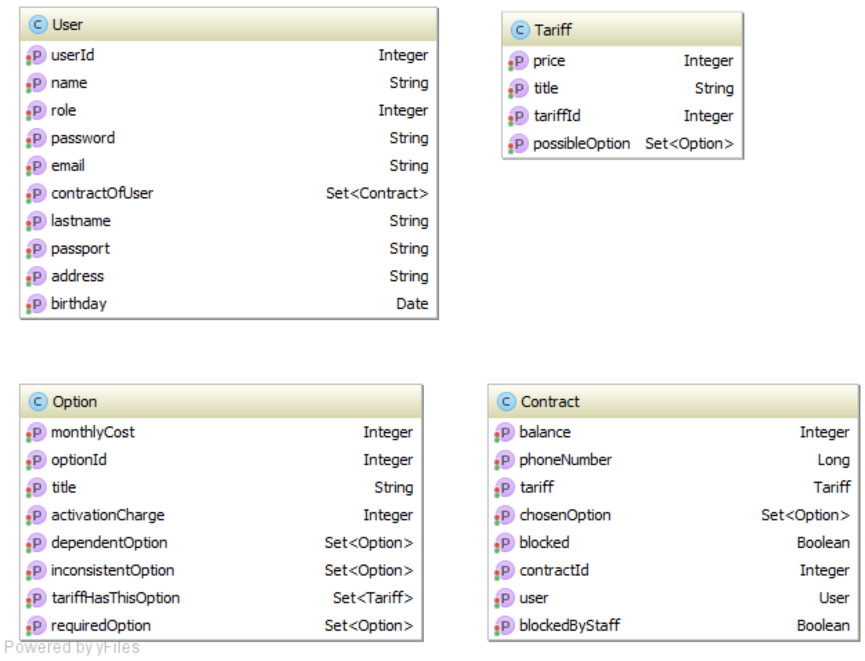
## Database design



All the foreign keys in each table have constrains on update and delete operations. Performing these operations with primary key will cause a cascade operation on foreign keys. The only exception is the **tariff** table. The removing of a tariff must not lead to removing of connected contracts. So in that case the cascade operations set to «SET NULL».

## Entities

Each class of entity linked with tables in database by the special annotation @Table in Entity class. Additional tables are associated with sets in the entities using annotations @JoinTable.



| User | |
|---|---|
| userId | Integer |
| name | String |
| role | Integer |
| password | String |
| email | String |
| contractOfUser | Set<Contract> |
| lastname | String |
| passport | String |
| address | String |
| birthday | Date |

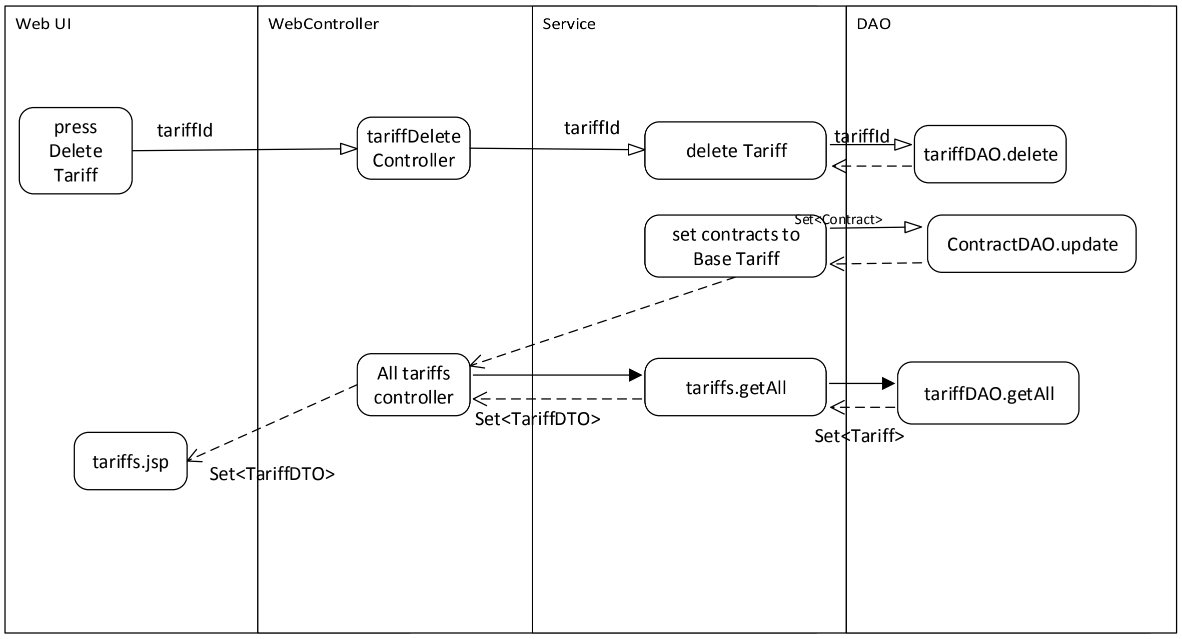
| Tariff | |
|---|---|
| price | Integer |
| title | String |
| tariffId | Integer |
| possibleOption | Set<Option> |

| Option | |
|---|---|
| monthlyCost | Integer |
| optionId | Integer |
| title | String |
| activationCharge | Integer |
| dependentOption | Set<Option> |
| inconsistentOption | Set<Option> |
| tariffHasThisOption | Set<Tariff> |
| requiredOption | Set<Option> |

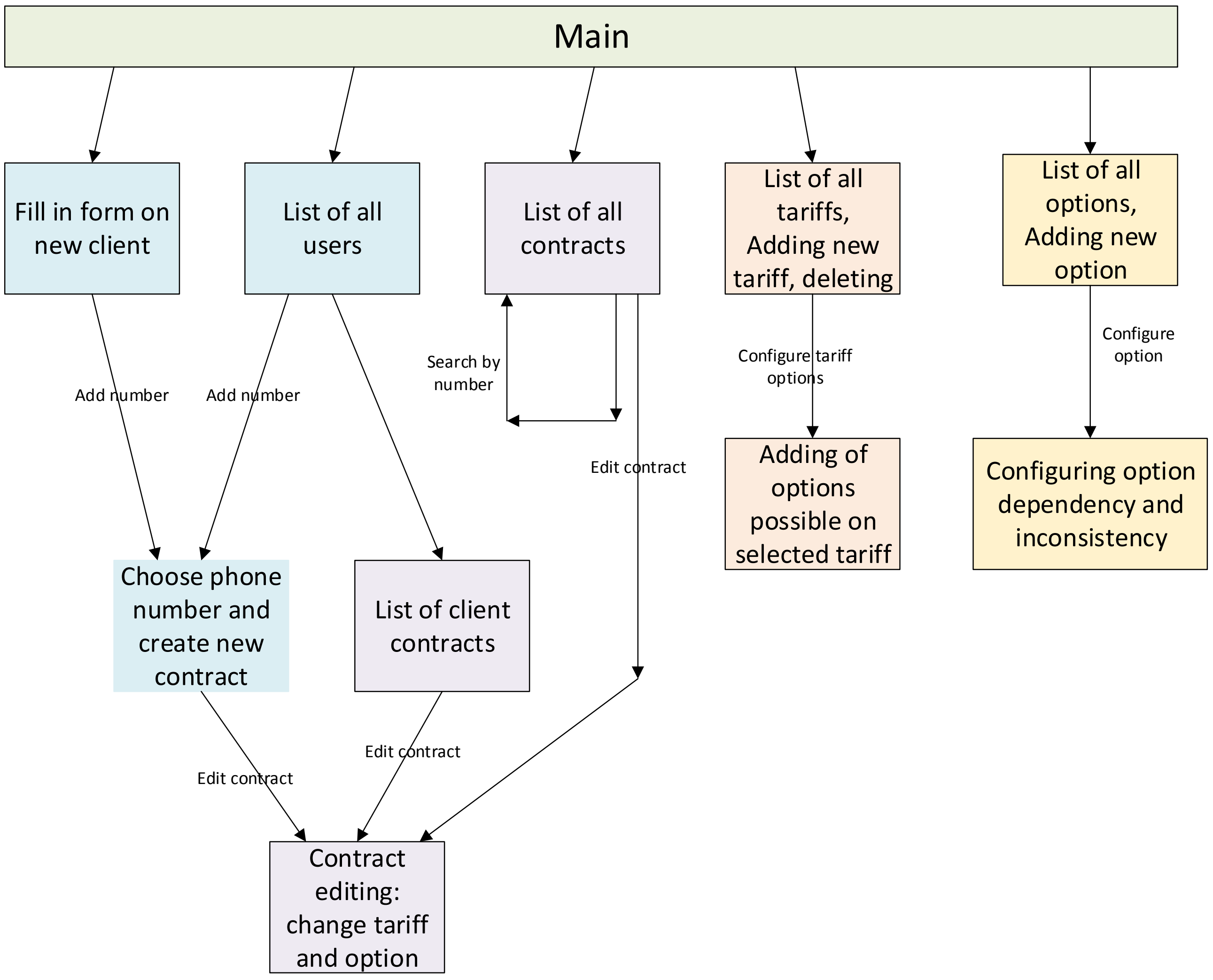
| Contract | |
|---|---|
| balance | Integer |
| phoneNumber | Long |
| tariff | Tariff |
| chosenOption | Set<Option> |
| blocked | Boolean |
| contractId | Integer |
| user | User |
| blockedByStaff | Boolean |

## Common interaction between application layers

The diagram below shows a common example of interactions between layers. This example is based on tariff removal.



## UI forms


Main
Fill in form on new client
List of all users
List of all contracts
List of all tariffs, Adding new tariff, deleting
List of all options, Adding new option
Add number
Add number
Search by number
Configure tariff options
Configure option
Edit contract
Adding of options possible on selected tariff
Configuring option dependency and inconsistency
Choose phone number and create new contract
List of client contracts
Edit contract
Edit contract
Contract editing: change tariff and option

## Input data validation

For the purpose of validation the user input data I used Java Validation API in Spring MVC. It requires a class for binding it to form’s inputs. But my existing classes (like UserDTO) were not what I need because Java Validation API does not provide @Pattern annotation for Integer fields. Consequently I created a bunch of classes for form validation. All those classes located in «*controllers_mvc.validationFormClasses*» package.

# Options | Option relationship

## Overview

By the conditions of the task an option may have a rule that it must work along with the specific set of other options. Some option may be inconsistent each other.

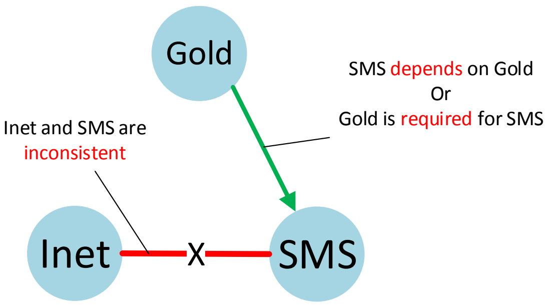
So let's review operations on option relationship.

- Deleting of inconsistency between two option
- Deleting option dependency of other option
- Adding option dependency of other option
- Adding inconsistency between two option

The first and the second points are quite simple and lead to deleting of appropriate records of tables in database (accordingly inconsistent_option and dependent_option – but actually with the help of JPA it goes under cover). The last two points are more complex. There is much attention are devoted in application to save overall option relationship correct after adding new dependency or inconsistency between options. The simplest example of incorrectness: option A depends on B and at the same time A and B are inconsistent.

Below I will review the adding of dependency or inconsistency in more details.

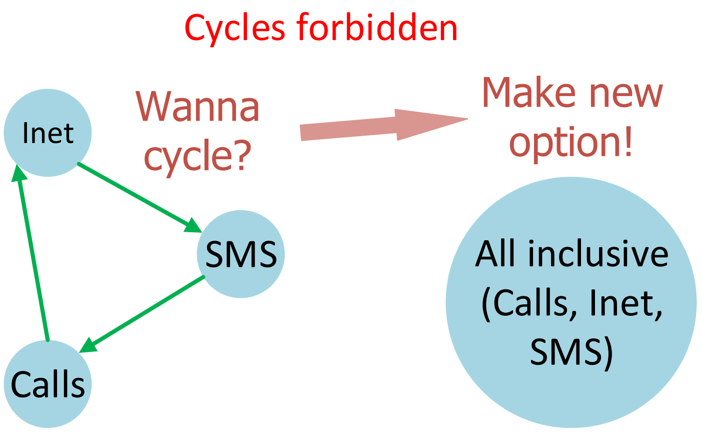
Later I will use A->B notation when saying that option B depends on option A (or in other words B is required A)



## Criteria of valid option relationship

First of all an important note here:

Option dependencies does not allow cycles. For example, if there is an existing option relationship A->B and B->C, the application should not give a user an ability to create a cycle like that: C->A.



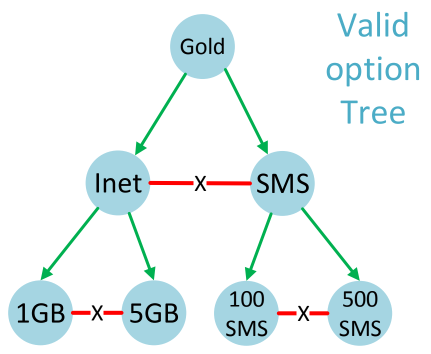
Why? This looks to be a sensible approach because cycle option dependencies would become an entire structure. When adding (to contract) an option from such a structure we would have to add and all the rest options from the cycle. The same behavior would be when deleting. The cycle actually would become a new «big» opting. That is why it is forbidden to create a cycle from option dependencies. As consequence, option relationship forms tree data structure.

The «main principle» of correctness of option relationship is following:

For any option «A» we may form a tree «T» containing «A» and all options which are required for «A». Every option from T should be consistent to each other from «T».

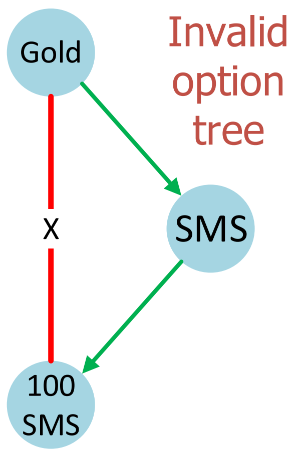
It would be clearer with a help of example. Let's review the Diagram 1 and assume that we want to add 100 SMS to a contract. We see that 100 SMS depends on SMS, which depends on Gold. All of them should be added to contract to. So we've got a tree {100 SMS, SMS, GOLD} and every option in it is consistent to each other. All correct.

Diag.1 Valid option tree

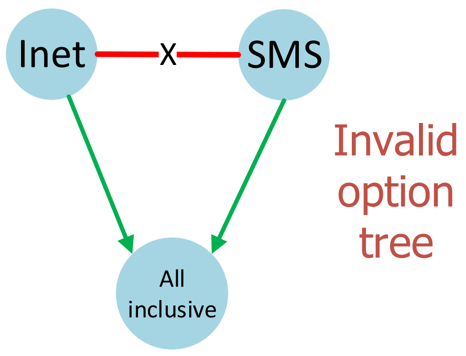


Let's review a couple of incorrect examples starting from Diagram2. We will add 100SMS to contract again. But it is inconsistent with Gold now. As in previous example, SMS and GOLD should be added to contract as well as 100SMS. But 100SMS and Gold are inconsistent. That's why options relationship on Diagram 2 is invalid. The similar thing is on Diagram 3 which also has an invalid option tree.

Diag.2 Invalid option tree



Diag.3 Invalid option tree



## When is it allowed to add option dependency?

It would be much useful to have criteria that if it would be correct to add a new dependency having existing option relationship.

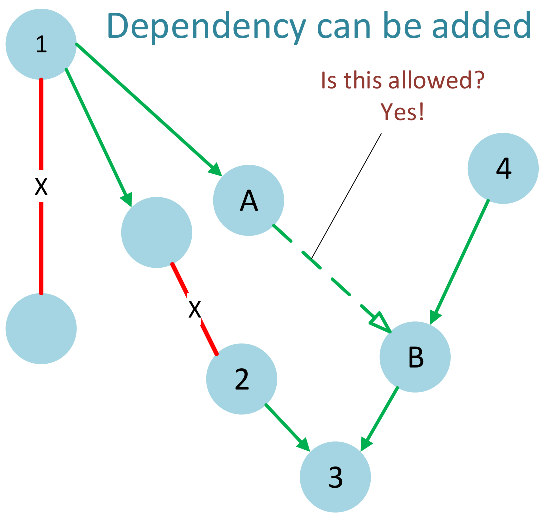
Let's assume we have to check a correctness of adding the dependency A->B (B depends on A). For check correctness we need to do the following:

1. Form the first set consisting of A and required option tree for A.
2. Form the second set consisting of B, required option tree for B and dependent option tree for B.
3. Check if there is no option from first set which are inconsistent to any option from second set. If such an inconsistency does not exist, adding dependency would be correct.

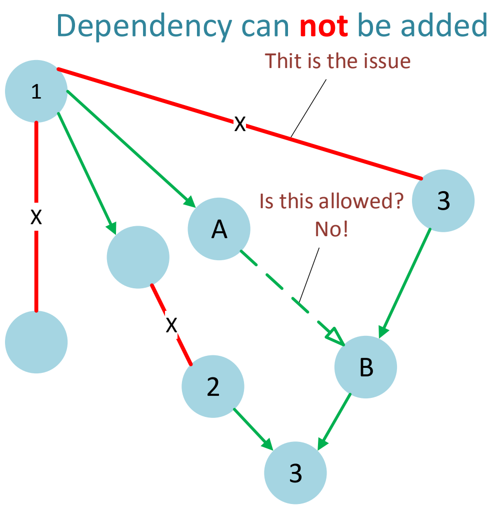
Let's see how it works on example (Diagram 4). First set would be {A, {1}}. Second set would be {B, {4}, {3}}. We see that there is no option from first set which is inconsistent to any option from second set. So the dependency A->B adding is correct.

Diagram 5 shows quite similar case but... First and second sets are the same. But option 1 from the first set is inconsistent to option 3 from second set. So the dependency A->B adding is incorrect.

Diag.4



Diag.5

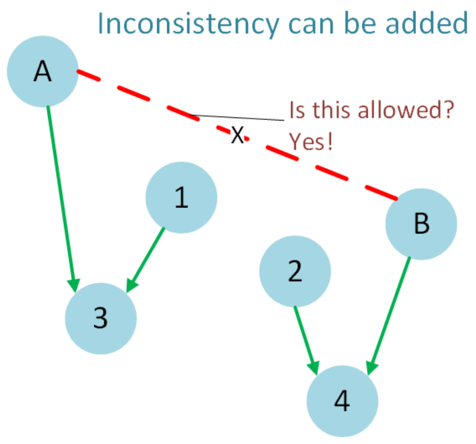


## When is it allowed to add option inconsistency?

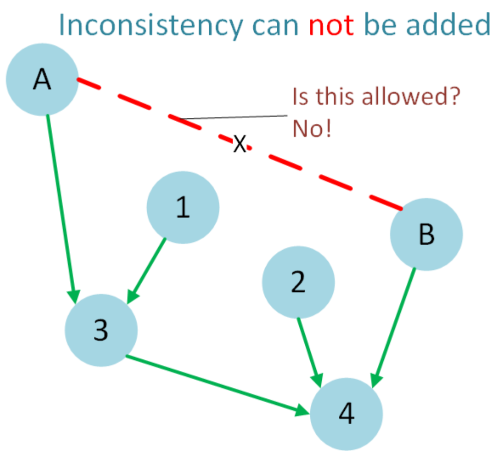
Let's assume we have to check a correctness of adding the inconsistency between A and B. For check correctness we need to do the following:

1. Form the first set consisting of A and dependent option tree for A.
2. Form the first set consisting of B and dependent option tree for B.
3. Check for the intersection of both sets. If it is not empty, adding the inconsistency would be incorrect.

Diag.6



Diag.7

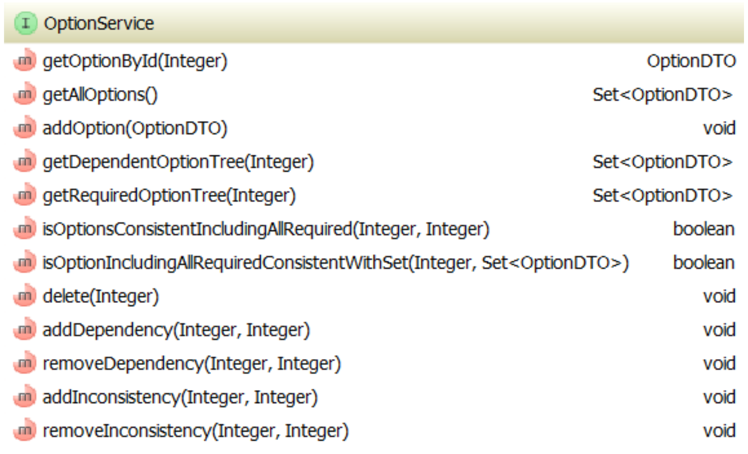


Let's see again how it works on example (Diagram 6). Would it be correct to add the inconsistency between A and B? We a going to follow algorithm described above. First set would be {A, {3}}. Second set would be {B, {4}}. We see that there is no intersection between both sets. That's why inconsistency between A and B can be added.

Diagram 7 is quite different. First set would be {A, {3,4}}. Second set would be {B, {4}}. Option 4 is the intersection of both sets. It is not empty so adding the inconsistency between A and B would not be correct.

## Implementation

As you can see in most cases algorithms on this section are based on routine of finding dependent option tree and finding required option tree for given option. In both cases uses the same recursive walk through a tree. All the required methods located it OptionService class.



| OptionService | |
|---|---|
| getOptionById(Integer) | OptionDTO |
| getAllOptions() | Set<OptionDTO> |
| addOption(OptionDTO) | void |
| getDependentOptionTree(Integer) | Set<OptionDTO> |
| getRequiredOptionTree(Integer) | Set<OptionDTO> |
| isOptionsConsistentIncludingAllRequired(Integer, Integer) | boolean |
| isOptionIncludingAllRequiredConsistentWithSet(Integer, Set<OptionDTO>) | boolean |
| delete(Integer) | void |
| addDependency(Integer, Integer) | void |
| removeDependency(Integer, Integer) | void |
| addInconsistency(Integer, Integer) | void |
| removeInconsistency(Integer, Integer) | void |

## Tariffs

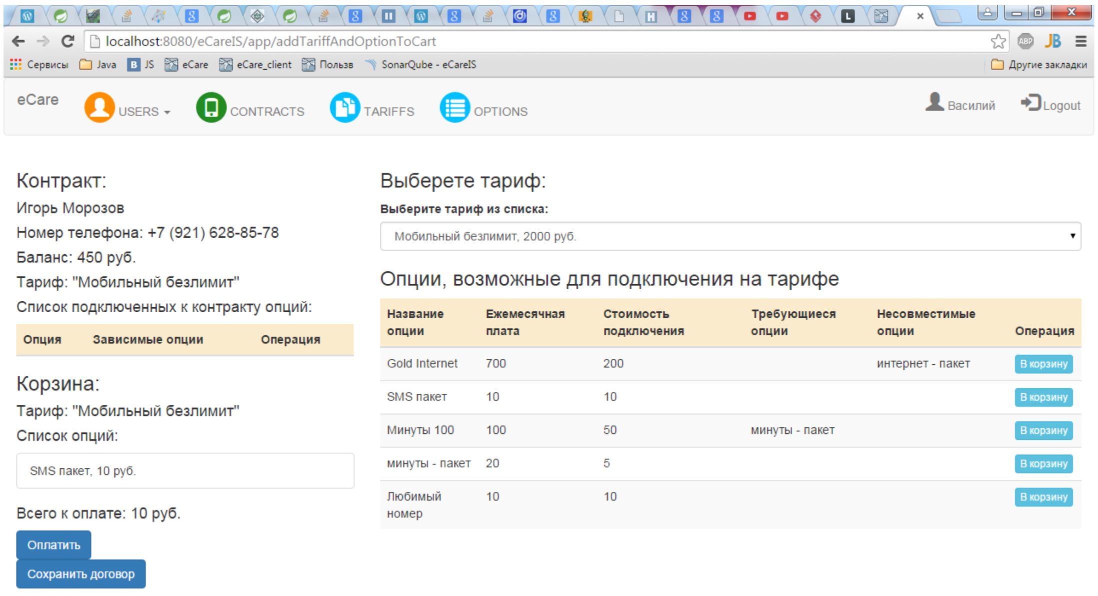
Creating of a new tariff is just a form validation (tariff title, price) and invocation of TariffDao.create() method. Removal of a tariff is more complex routine. It requires caring of contracts which are connected to the tariff. Those entire contracts are being connected to so-called «Base tariff» which cannot be removed. This process was shown in details on diagram some pages ago.

By the condition of the task each tariff can work with a specific set of options. So the application let an ability to configure option set for an each tariff. Note that at that moment it doesn't need to care of inconsistency between options. For instance, we are free to add any two inconsistent options to tariff (not to contract!). (We will have to worry about it when adding options to contracts, not to a tariff).

Having developed the getting of all required option tree for some selected option the application helps with connecting of all of them when we adding some option. For example, when we want to connect 100 SMS (recall diagram 1) to some tariff, the application will connect  SMS and GOLD automatically.

The similar things happen when disabling an option from a tariff. Application automatically disable all dependent options for chosen option. For example, when disabling opting Gold (see diagram 1 again) the application will automatically disable Inet, 1Gb, 5Gb, 100 SMS, SMS, GOLD.

## Contract editing


localhost:8080/eCareIS/app/addTariffAndOptionToCart

eCare USERS CONTRACTS TARIFFS OPTIONS Василий Logout

Контракт:
Игорь Морозов
Номер телефона: +7 (921) 628-85-78
Баланс: 450 руб.
Тариф: "Мобильный безлимит"
Список подключенных к контракту опций:

| Опция | Зависимые опции | Операция |
|---|---|---|

Корзина:
Тариф: "Мобильный безлимит"
Список опций:
SMS пакет, 10 руб.

Всего к оплате: 10 руб.
Оплатить
Сохранить договор

Выберете тариф:
Выберите тариф из списка:
Мобильный безлимит, 2000 руб.

Опции, возможные для подключения на тарифе

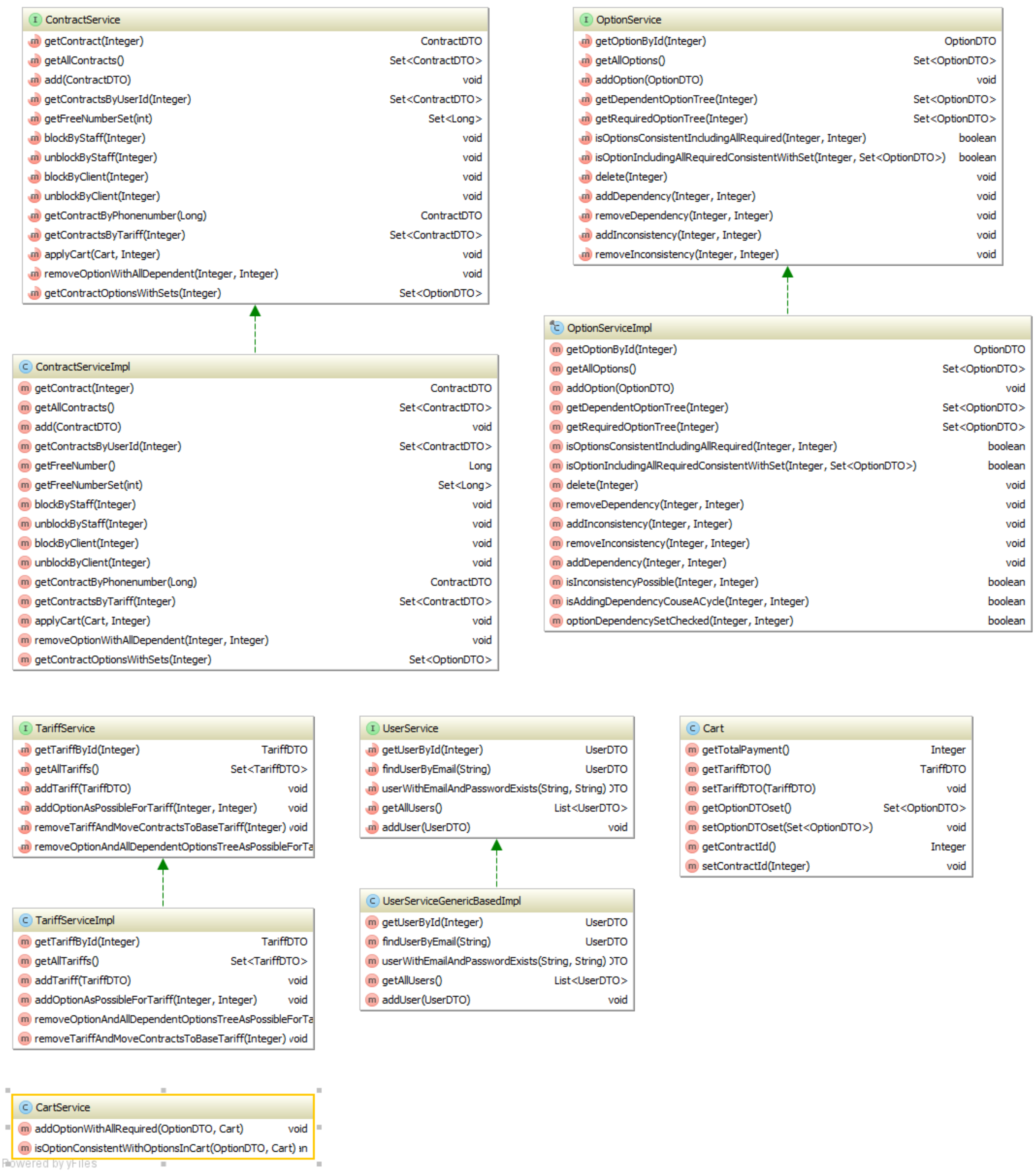
| Название опции | Ежемесячная плата | Стоимость подключения | Требующиеся опции | Несовместимые опции | Операция |
|---|---|---|---|---|---|
| Gold Internet | 700 | 200 | | интернет - пакет | В корзину |
| SMS пакет | 10 | 10 | | | В корзину |
| Минуты 100 | 100 | 50 | минуты - пакет | | В корзину |
| минуты - пакет | 20 | 5 | | | В корзину |
| Любимый номер | 10 | 10 | | | В корзину |

This from is intended both for clients and employees. A user may view his current contract configuration, view existing tariffs, view tariff options. Unblocked user can change tariff and options. Firstly new tariff and options get to the shopping cart. Tariff and options are changed in the contract only after applying the shopping cart. The shopping cart is implemented by Cart class. The instance of Cart is stored in the HttpSession.

Unlike tariff when editing contacts option inconsistency matters. Application doesn't allow adding options which are inconsistent to existing options in shopping cart.

# Services class diagram

All service classes as well as DAO classes have interfaces and implementations.



| ContractService | |
|---|---|
| getContract(Integer) | ContractDTO |
| getAllContracts() | Set<ContractDTO> |
| add(ContractDTO) | void |
| getContractsByUserId(Integer) | Set<ContractDTO> |
| getFreeNumberSet(int) | Set<Long> |
| blockByStaff(Integer) | void |
| unblockByStaff(Integer) | void |
| blockByClient(Integer) | void |
| unblockByClient(Integer) | void |
| getContractByPhonenumber(Long) | ContractDTO |
| getContractsByTariff(Integer) | Set<ContractDTO> |
| applyCart(Cart, Integer) | void |
| removeOptionWithAllDependent(Integer, Integer) | void |
| getContractOptionsWithSets(Integer) | Set<OptionDTO> |

| ContractServiceImpl | |
|---|---|
| getContract(Integer) | ContractDTO |
| getAllContracts() | Set<ContractDTO> |
| add(ContractDTO) | void |
| getContractsByUserId(Integer) | Set<ContractDTO> |
| getFreeNumber() | Long |
| getFreeNumberSet(int) | Set<Long> |
| blockByStaff(Integer) | void |
| unblockByStaff(Integer) | void |
| blockByClient(Integer) | void |
| unblockByClient(Integer) | void |
| getContractByPhonenumber(Long) | ContractDTO |
| getContractsByTariff(Integer) | Set<ContractDTO> |
| applyCart(Cart, Integer) | void |
| removeOptionWithAllDependent(Integer, Integer) | void |
| getContractOptionsWithSets(Integer) | Set<OptionDTO> |

| OptionService | |
|---|---|
| getOptionById(Integer) | OptionDTO |
| getAllOptions() | Set<OptionDTO> |
| addOption(OptionDTO) | void |
| getDependentOptionTree(Integer) | Set<OptionDTO> |
| getRequiredOptionTree(Integer) | Set<OptionDTO> |
| isOptionsConsistentIncludingAllRequired(Integer, Integer) | boolean |
| isOptionIncludingAllRequiredConsistentWithSet(Integer, Set<OptionDTO>) | boolean |
| delete(Integer) | void |
| addDependency(Integer, Integer) | void |
| removeDependency(Integer, Integer) | void |
| addInconsistency(Integer, Integer) | void |
| removeInconsistency(Integer, Integer) | void |

| OptionServiceImpl | |
|---|---|
| getOptionById(Integer) | OptionDTO |
| getAllOptions() | Set<OptionDTO> |
| addOption(OptionDTO) | void |
| getDependentOptionTree(Integer) | Set<OptionDTO> |
| getRequiredOptionTree(Integer) | Set<OptionDTO> |
| isOptionsConsistentIncludingAllRequired(Integer, Integer) | boolean |
| isOptionIncludingAllRequiredConsistentWithSet(Integer, Set<OptionDTO>) | boolean |
| delete(Integer) | void |
| removeDependency(Integer, Integer) | void |
| addInconsistency(Integer, Integer) | void |
| removeInconsistency(Integer, Integer) | void |
| addDependency(Integer, Integer) | void |
| isInconsistencyPossible(Integer, Integer) | boolean |
| isAddingDependencyCouseACycle(Integer, Integer) | boolean |
| optionDependencySetChecked(Integer, Integer) | boolean |

| TariffService | |
|---|---|
| getTariffById(Integer) | TariffDTO |
| getAllTariffs() | Set<TariffDTO> |
| addTariff(TariffDTO) | void |
| addOptionAsPossibleForTariff(Integer, Integer) | void |
| removeTariffAndMoveContractsToBaseTariff(Integer) | void |
| removeOptionAndAllDependentOptionsTreeAsPossibleForTa | |

| TariffServiceImpl | |
|---|---|
| getTariffById(Integer) | TariffDTO |
| getAllTariffs() | Set<TariffDTO> |
| addTariff(TariffDTO) | void |
| addOptionAsPossibleForTariff(Integer, Integer) | void |
| removeOptionAndAllDependentOptionsTreeAsPossibleForTa | |
| removeTariffAndMoveContractsToBaseTariff(Integer) | void |

| UserService | |
|---|---|
| getUserById(Integer) | UserDTO |
| findUserByEmail(String) | UserDTO |
| userWithEmailAndPasswordExists(String, String) | )TO |
| getAllUsers() | List<UserDTO> |
| addUser(UserDTO) | void |

| UserServiceGenericBasedImpl | |
|---|---|
| getUserById(Integer) | UserDTO |
| findUserByEmail(String) | UserDTO |
| userWithEmailAndPasswordExists(String, String) | )TO |
| getAllUsers() | List<UserDTO> |
| addUser(UserDTO) | void |

| Cart | |
|---|---|
| getTotalPayment() | Integer |
| getTariffDTO() | TariffDTO |
| setTariffDTO(TariffDTO) | void |
| getOptionDTOset() | Set<OptionDTO> |
| setOptionDTOset(Set<OptionDTO>) | void |
| getContractId() | Integer |
| setContractId(Integer) | void |

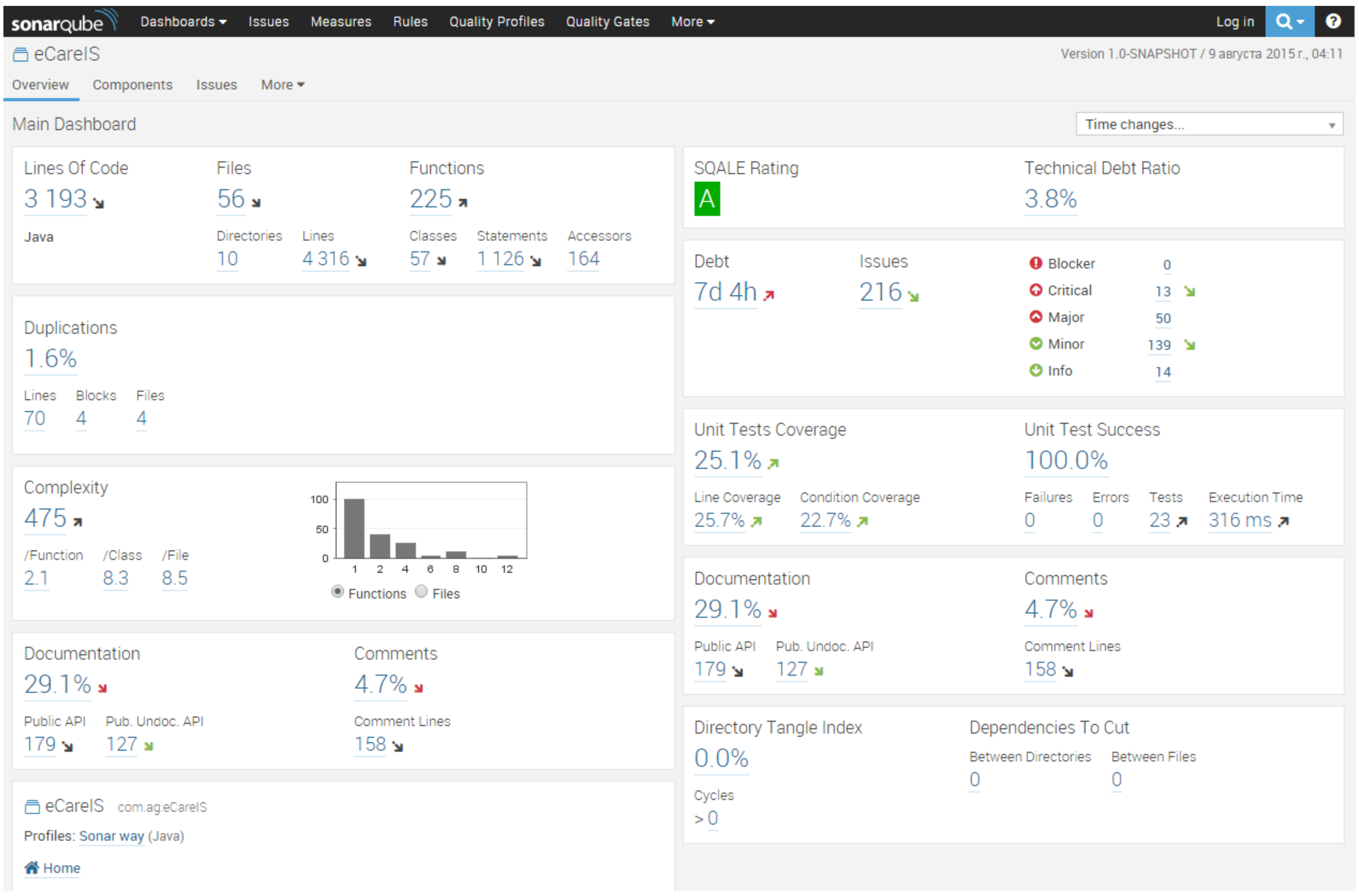
| CartService | |
|---|---|
| addOptionWithAllRequired(OptionDTO, Cart) | void |
| isOptionConsistentWithOptionsInCart(OptionDTO, Cart) | ın |

## Authorization and authentication

During the logging user information data (UserId, ContractID (if a user was a client)) are stored into the HttpSession. Authentication is implemented on HttpFilter, which does all the necessary checks before passing a request to the web-controller.

Beside that filter is also responsible for the authorization. It checks for user can not have an access to the editing of tariffs, options or somebody else's contract. It also doesn’t allow a client to change a contract when his number is locked.

## Sonar report



eCareIS — Version 1.0-SNAPSHOT / 9 августа 2015 г., 04:11

Main Dashboard

Lines Of Code 3 193 — Java
Files 56 — Directories 10, Lines 4 316
Functions 225 — Classes 57, Statements 1 126, Accessors 164

Duplications 1.6% — Lines 70, Blocks 4, Files 4

Complexity 475 — /Function 2.1, /Class 8.3, /File 8.5

Documentation 29.1% — Public API 179, Pub. Undoc. API 127
Comments 4.7% — Comment Lines 158

eCareIS com.ag:eCareIS
Profiles: Sonar way (Java)

SQALE Rating A
Technical Debt Ratio 3.8%

Debt 7d 4h
Issues 216

| | |
|---|---|
| Blocker | 0 |
| Critical | 13 |
| Major | 50 |
| Minor | 139 |
| Info | 14 |

Unit Tests Coverage 25.1% — Line Coverage 25.7%, Condition Coverage 22.7%
Unit Test Success 100.0% — Failures 0, Errors 0, Tests 23, Execution Time 316 ms

Documentation 29.1% — Public API 179, Pub. Undoc. API 127
Comments 4.7% — Comment Lines 158

Directory Tangle Index 0.0% — Cycles > 0
Dependencies To Cut — Between Directories 0, Between Files 0