Technical solution description

Logiweb project

MIKHAIL MAZURKEVICH

Table of contents

Project Goal

Develop software that represents information system for freight management. Application should be able to manage Trucks, Drivers, Orders. And implements remote control system for driver's delivering

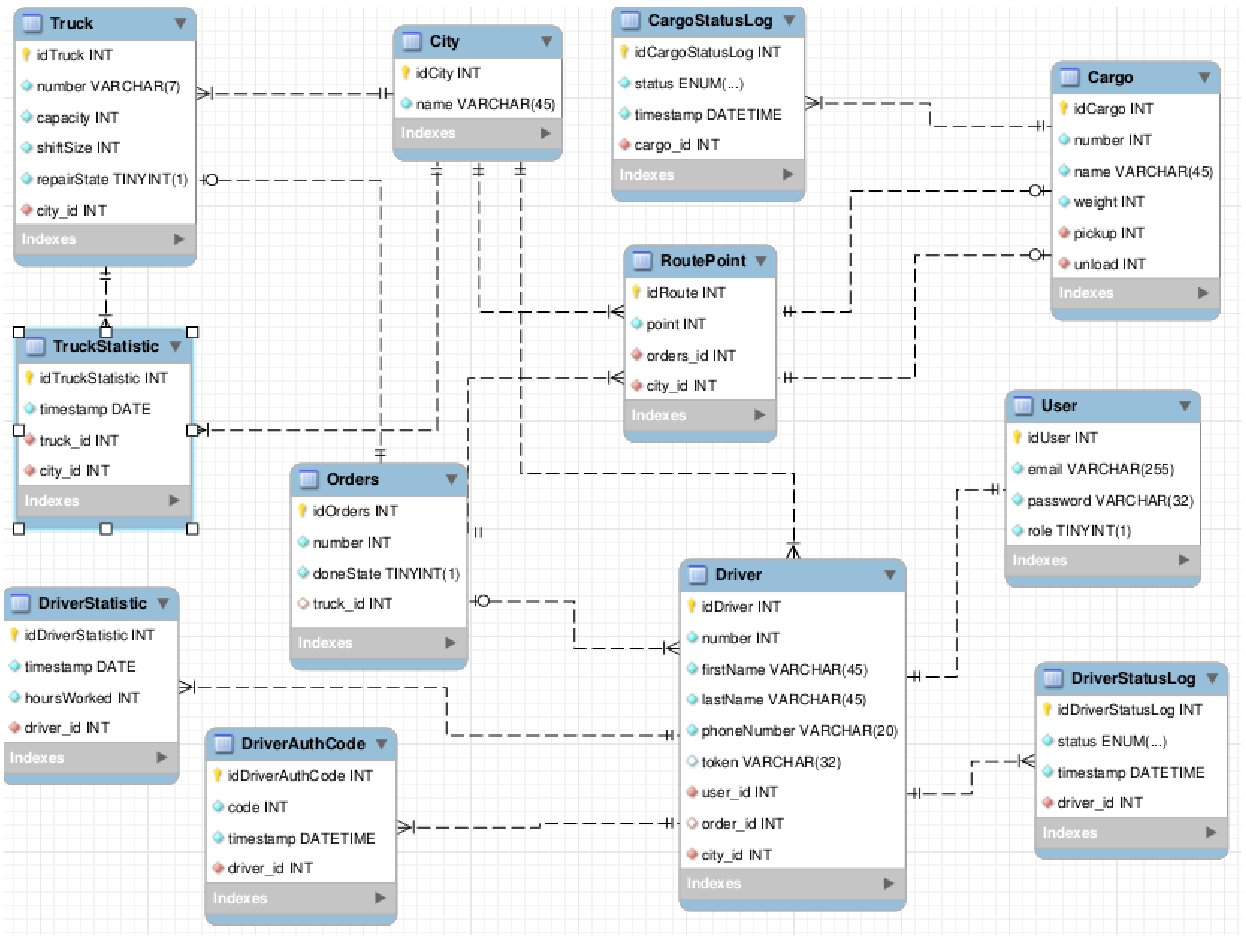
List of used tecnologies and framevorks

1. Tomcat 8.*
2. Wildfly 10.*
3. Maven 3.*
4. Servlet
5. JSP/JSTL
6. JPA 2.1
7. Spring
8. MySQL
9. JSF
10. EJB
11. Herocu
12. JavaMail
13. Junit / Mokito
14. Log4j

Additional features

1. Sending mail to driver when he added to system and assing to order with direct link to order description(you can watch order information without login becouse this link is secure).
2. Sending sms massage when driver assign to odred and for driver logining to system, this look like you enter your number, when get sms with verification code and succesfully login.
3. To provide flexible creating order there is Yandex map for bilding route map and get hours of works which will spend on order's delivering.
4. Autocomplite fields with city name.
5. Project deploing to herocu platform for remote access.
6. Some statistics by one month for trucks and drivers.

Database scheme



This image is database visualisation.

1. User consist of email, password and role, this table describes managers of system and part of driver's information.
2. Driver table has full information about persone which can delivere cargoes. Phone number used for sending sms, tocken need for generate and create link for order description. Relationship 1:1 with User table. Contains FK link to order.
3. DriverAuthCode contains verification codes for login driver to system. Relationship is 1:n because 1 driver be able has many codes.
4. DriverStatusLog describe then and which status driver set. Relationship same as DriverAuthCode table.
5. DriverStatistic consist of driver's hours worked.
6. Truck, TruckStatistic, Cargo, CargoStatusLog this tables same as describeing before they describe entity Truck and Cargo and some statistics.
7. Order describe entity which contains number, done state and link to track which assign to order.
8. RoutePoint table consist of route points, there orderliness and link to city.

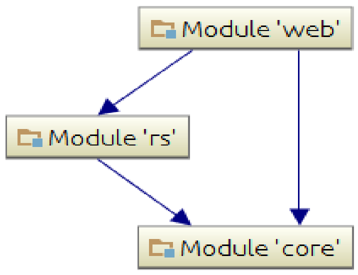
Application modules

Architecture of this app consists of 3 modules.

1. Core — consists of 3 packages entity, dao and service layer
2. Web module describe controllers, verification classes and user UI
3. RS describe all application api using for autocomplete and remote driver access.

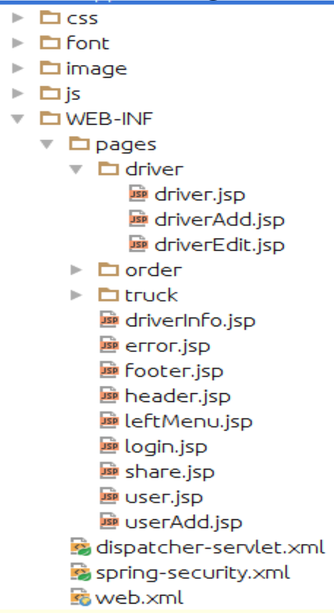
Web module get request form servlet dispatcher and address it to corresponding controller which invoke service from core module and service contacts with dao app layer.

RS module comunicate directly with core module by invoking corresponding methods in services.



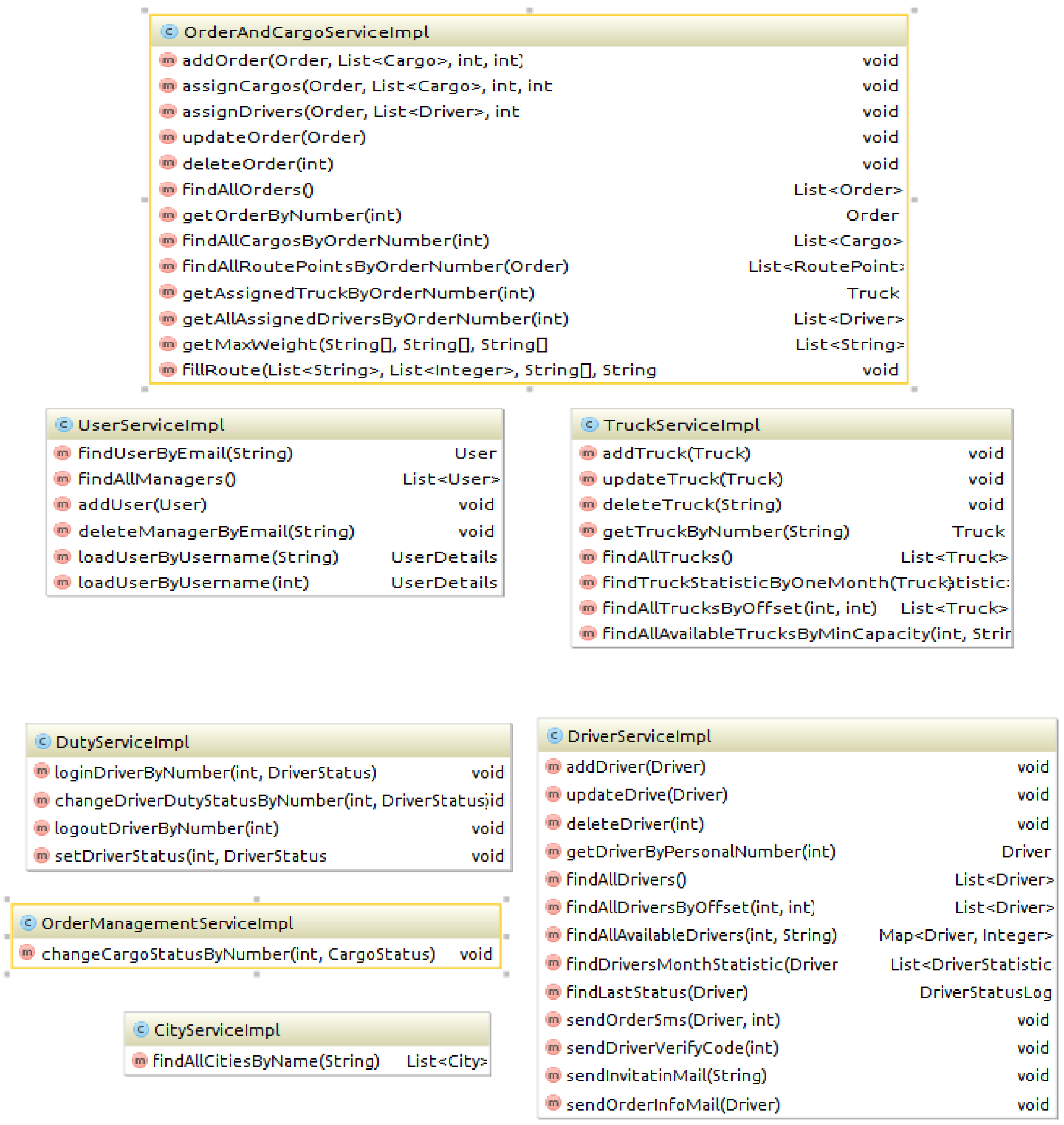
User interface

All users pages use JSP tecnology for page creating. Main things which includs in all pages are footer, header, left menu. Header consists of css including files and js scripts. For creating users pages was used materiallaz css frameworks, jQuery, smart wizard component (wich was fully castomyse). Pages folder consists of driver, order and truck folders which implements full logic binding with this entity.



```
▸ css
▸ font
▸ image
▸ js
▾ WEB-INF
  ▾ pages
    ▾ driver
        driver.jsp
        driverAdd.jsp
        driverEdit.jsp
    ▸ order
    ▸ truck
      driverInfo.jsp
      error.jsp
      footer.jsp
      header.jsp
      leftMenu.jsp
      login.jsp
      share.jsp
      user.jsp
      userAdd.jsp
    dispatcher-servlet.xml
    spring-security.xml
    web.xml
```

Business logic



```
OrderAndCargoServiceImpl
addOrder(Order, List<Cargo>, int, int)                void
assignCargos(Order, List<Cargo>, int, int             void
assignDrivers(Order, List<Driver>, int                void
updateOrder(Order)                                    void
deleteOrder(int)                                      void
findAllOrders()                                       List<Order>
getOrderByNumber(int)                                 Order
findAllCargosByOrderNumber(int)                       List<Cargo>
findAllRoutePointsByOrderNumber(Order)                List<RoutePoint
getAssignedTruckByOrderNumber(int)                    Truck
getAllAssignedDriversByOrderNumber(int)               List<Driver>
getMaxWeight(String[], String[], String[]             List<String>
fillRoute(List<String>, List<Integer>, String[], String   void
```

```
UserServiceImpl
findUserByEmail(String)               User
findAllManagers()                     List<User>
addUser(User)                         void
deleteManagerByEmail(String)          void
loadUserByUsername(String)            UserDetails
loadUserByUsername(int)               UserDetails
```

```
TruckServiceImpl
addTruck(Truck)                                   void
updateTruck(Truck)                                void
deleteTruck(String)                               void
getTruckByNumber(String)                          Truck
findAllTrucks()                                   List<Truck>
findTruckStatisticByOneMonth(Truck)tistic
findAllTrucksByOffset(int, int)                   List<Truck>
findAllAvailableTrucksByMinCapacity(int, Strir
```

```
DutyServiceImpl
loginDriverByNumber(int, DriverStatus)                void
changeDriverDutyStatusByNumber(int, DriverStatus)id
logoutDriverByNumber(int)                             void
setDriverStatus(int, DriverStatus                     void
```

```
OrderManagementServiceImpl
changeCargoStatusByNumber(int, CargoStatus)    void
```

```
CityServiceImpl
findAllCitiesByName(String)    List<City>
```

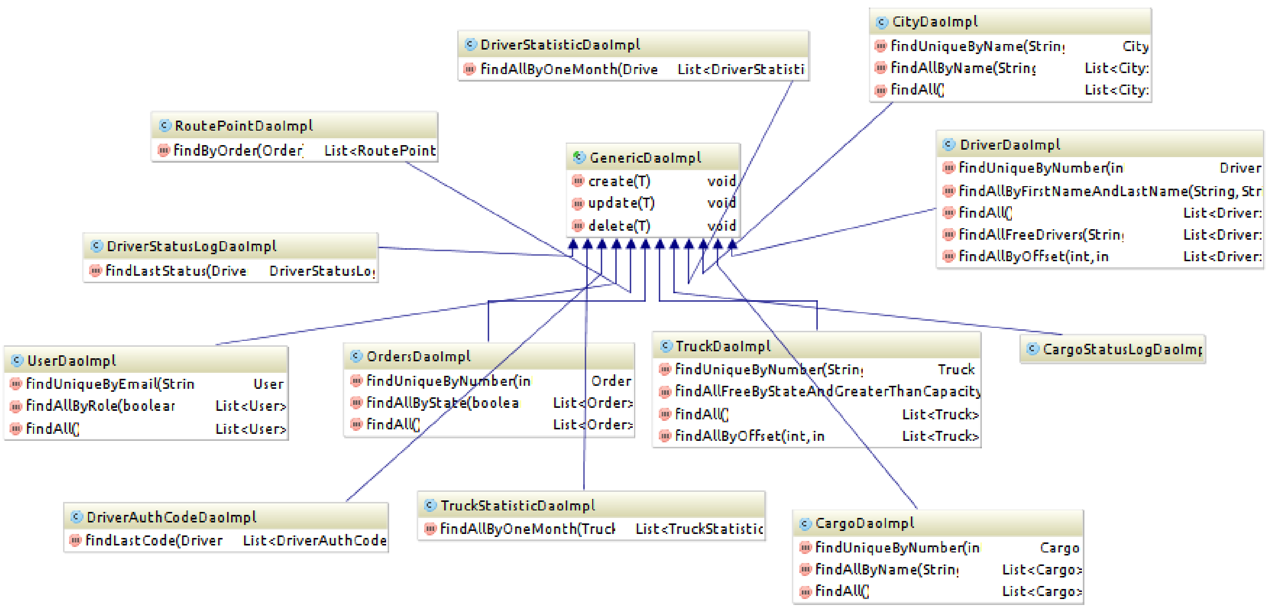
```
DriverServiceImpl
addDriver(Driver)                          void
updateDrive(Driver)                        void
deleteDriver(int)                          void
getDriverByPersonalNumber(int)             Driver
findAllDrivers()                           List<Driver>
findAllDriversByOffset(int, int)           List<Driver>
findAllAvailableDrivers(int, String)       Map<Driver, Integer>
findDriversMonthStatistic(Driver           List<DriverStatistic
findLastStatus(Driver)                     DriverStatusLog
sendOrderSms(Driver, int)                  void
sendDriverVerifyCode(int)                  void
sendInvitatinMail(String)                  void
sendOrderInfoMail(Driver)                  void
```

This image describe service layer, this layer contains transaction managment.

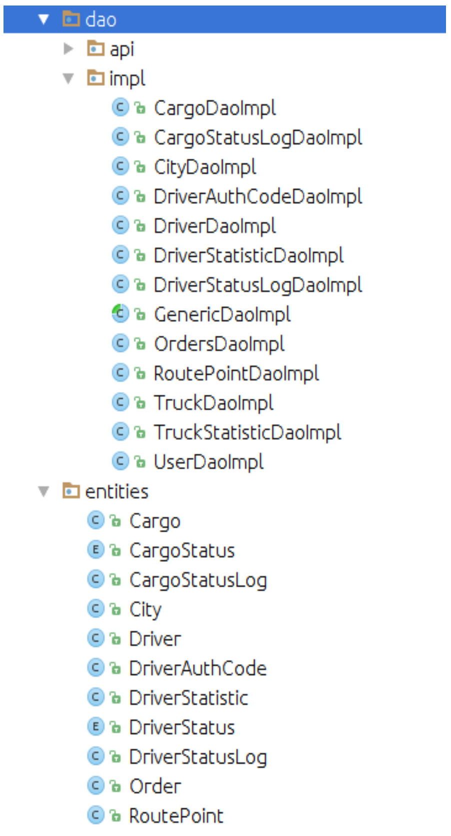
1. CityService — using for autocompleate it comunicate with CityDao and get all cities like name.
2. DriverService — using for create, delete, update drivers, find drivers, sending sms, email and generate verify code, get driver statistic and find all drivers suitable for order.

3. TruckService — using for create, update, delete trucks, find all avaliable trucks, find all trucks and get truck statistic by one month.
4. UserService — help us to create, delete manager and to get data for custome authentification provider.
5. DutyService — use for changing driver status, if previous status was driving when count hours of work.
6. OrderManagerService — set cargo status and check if order's cargoes delivered than chenge order done status and free all assign resources.
7. OrderAndCargoService — used for creating order and assign with it cargoes, route points, assign truck and driver also this class used for creating route map and count max weight.

Entities, Dao layer description


CityDaoImpl
findUniqueByName(Strin; City
findAllByName(String List<City:
findAll() List<City:
DriverStatisticDaoImpl
findAllByOneMonth(Drive List<DriverStatisti
RoutePointDaoImpl
findByOrder(Order List<RoutePoint
GenericDaoImpl
create(T) void
update(T) void
delete(T) void
DriverDaoImpl
findUniqueByNumber(in Driver
findAllByFirstNameAndLastName(String, Str
findAll() List<Driver:
findAllFreeDrivers(Strin; List<Driver:
findAllByOffset(int, in List<Driver:
DriverStatusLogDaoImpl
findLastStatus(Drive DriverStatusLo;
UserDaoImpl
findUniqueByEmail(Strin User
findAllByRole(boolear List<User>
findAll() List<User>
OrdersDaoImpl
findUniqueByNumber(in Order
findAllByState(boolea List<Order>
findAll() List<Order>
TruckDaoImpl
findUniqueByNumber(Strin; Truck
findAllFreeByStateAndGreaterThanCapacity
findAll() List<Truck>
findAllByOffset(int, in List<Truck>
CargoStatusLogDaoImp
DriverAuthCodeDaoImpl
findLastCode(Driver List<DriverAuthCode
TruckStatisticDaoImpl
findAllByOneMonth(Truck List<TruckStatistic
CargoDaoImpl
findUniqueByNumber(in Cargo
findAllByName(Strin; List<Cargo>
findAll() List<Cargo>

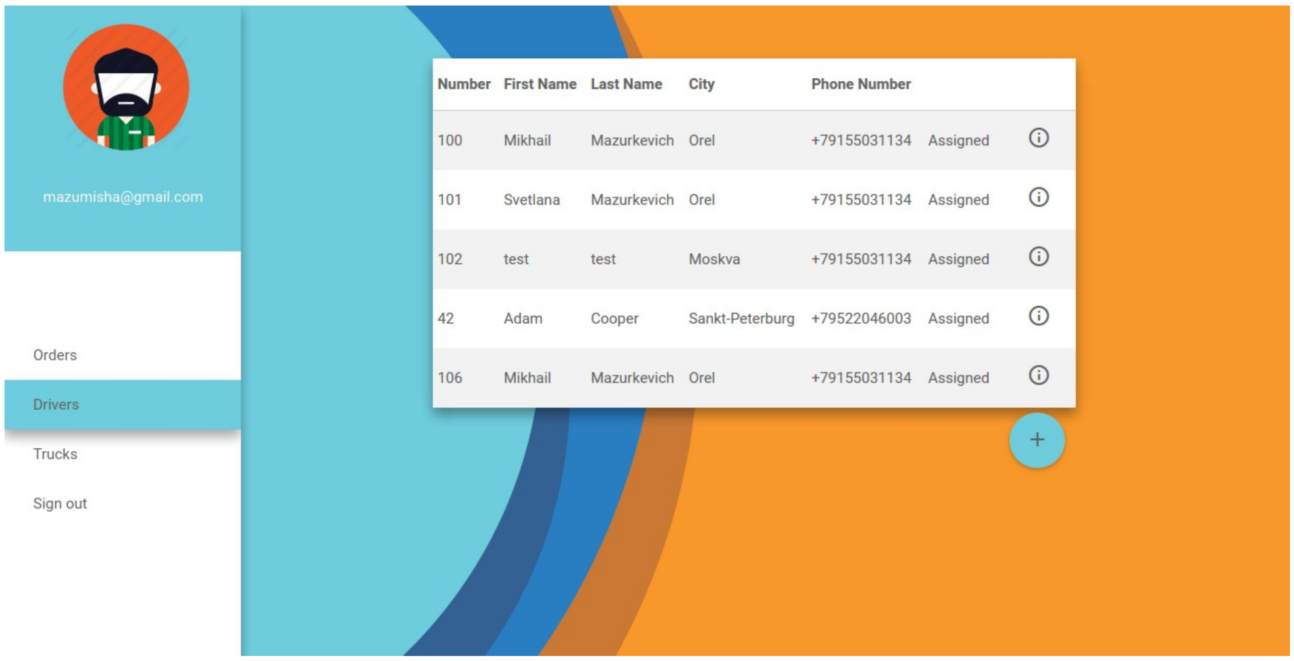
Entities layer describe mapping database tables on objects, fields in table same in classes. The tables structure you can see on image above. Transactions handel by spring and annotation @Transactional use on methods. Dao layer use for commone db operations at objects such CRUD and some select queries. More info you can see in my github project.



```
▼ dao
    ▶ api
    ▼ impl
        CargoDaoImpl
        CargoStatusLogDaoImpl
        CityDaoImpl
        DriverAuthCodeDaoImpl
        DriverDaoImpl
        DriverStatisticDaoImpl
        DriverStatusLogDaoImpl
        GenericDaoImpl
        OrdersDaoImpl
        RoutePointDaoImpl
        TruckDaoImpl
        TruckStatisticDaoImpl
        UserDaoImpl
▼ entities
    Cargo
    CargoStatus
    CargoStatusLog
    City
    Driver
    DriverAuthCode
    DriverStatistic
    DriverStatus
    DriverStatusLog
    Order
    RoutePoint
```

Screenshots

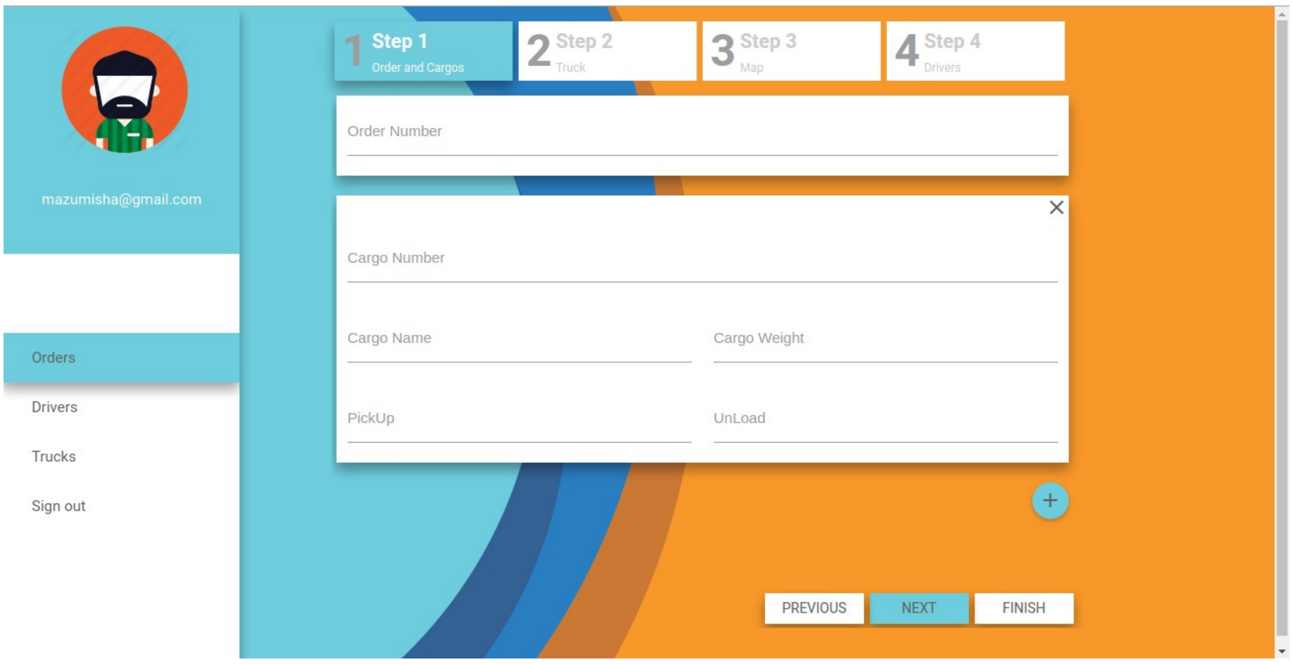
Main pages of app look like images below



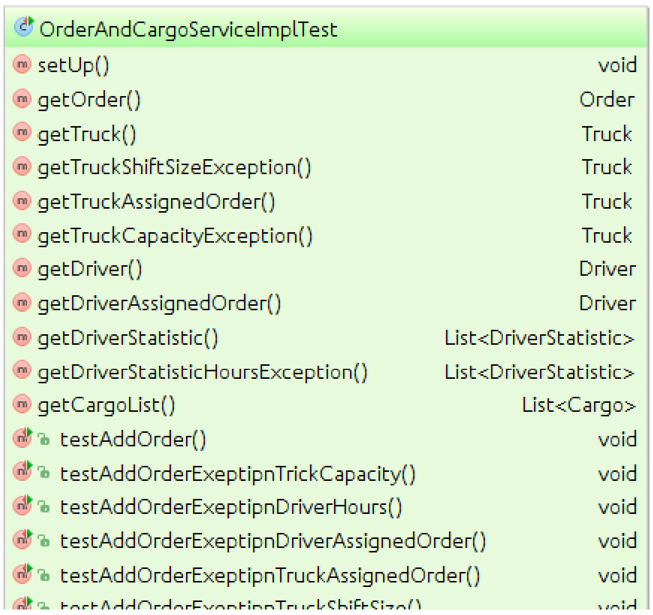
mazumisha@gmail.com

Number	First Name	Last Name	City	Phone Number		
100	Mikhail	Mazurkevich	Orel	+79155031134	Assigned	ⓘ
101	Svetlana	Mazurkevich	Orel	+79155031134	Assigned	ⓘ
102	test	test	Moskva	+79155031134	Assigned	ⓘ
42	Adam	Cooper	Sankt-Peterburg	+79522046003	Assigned	ⓘ
106	Mikhail	Mazurkevich	Orel	+79155031134	Assigned	ⓘ

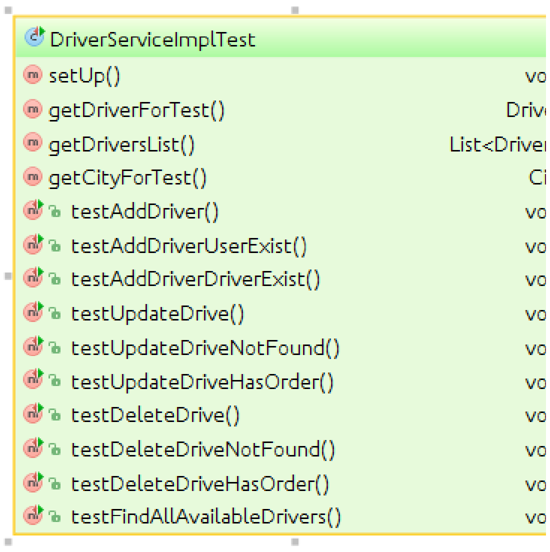
Page shows drivers



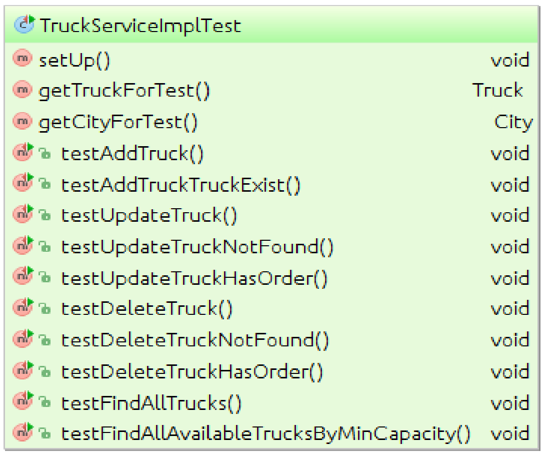
JUnit tests



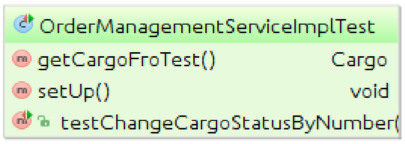
OrderAndCargoServiceImplTest	
setUp()	void
getOrder()	Order
getTruck()	Truck
getTruckShiftSizeException()	Truck
getTruckAssignedOrder()	Truck
getTruckCapacityException()	Truck
getDriver()	Driver
getDriverAssignedOrder()	Driver
getDriverStatistic()	List<DriverStatistic>
getDriverStatisticHoursException()	List<DriverStatistic>
getCargoList()	List<Cargo>
testAddOrder()	void
testAddOrderExeptipnTrickCapacity()	void
testAddOrderExeptipnDriverHours()	void
testAddOrderExeptipnDriverAssignedOrder()	void
testAddOrderExeptipnTruckAssignedOrder()	void
testAddOrderExeptipnTruckShiftSize()	void



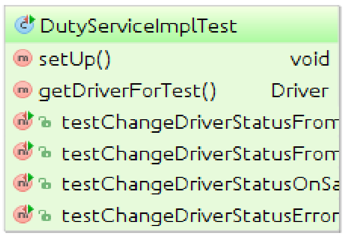
DriverServiceImplTest	
setUp()	vo
getDriverForTest()	Driv
getDriversList()	List<Drive
getCityForTest()	Ci
testAddDriver()	vo
testAddDriverUserExist()	vo
testAddDriverDriverExist()	vo
testUpdateDrive()	vo
testUpdateDriveNotFound()	vo
testUpdateDriveHasOrder()	vo
testDeleteDrive()	vo
testDeleteDriveNotFound()	vo
testDeleteDriveHasOrder()	vo
testFindAllAvailableDrivers()	vo



TruckServiceImplTest	
setUp()	void
getTruckForTest()	Truck
getCityForTest()	City
testAddTruck()	void
testAddTruckTruckExist()	void
testUpdateTruck()	void
testUpdateTruckNotFound()	void
testUpdateTruckHasOrder()	void
testDeleteTruck()	void
testDeleteTruckNotFound()	void
testDeleteTruckHasOrder()	void
testFindAllTrucks()	void
testFindAllAvailableTrucksByMinCapacity()	void



OrderManagementServiceImplTest	
getCargoFroTest()	Cargo
setUp()	void
testChangeCargoStatusByNumber(	



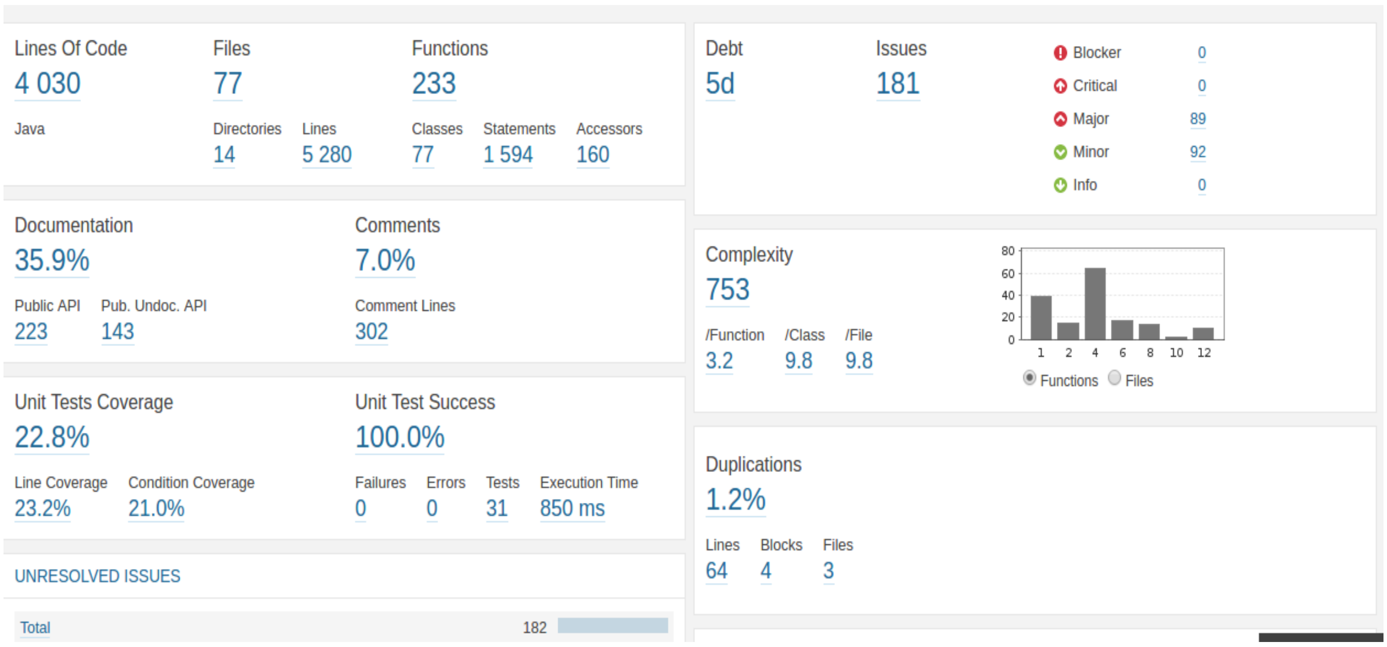
DutyServiceImplTest	
setUp()	void
getDriverForTest()	Driver
testChangeDriverStatusFrom	
testChangeDriverStatusFrom	
testChangeDriverStatusOnSa	
testChangeDriverStatusError	

Methods describe in classes testing service logic and test some exceptions which appear when parameters are wrong. They named like services layer classes with Test suffix.

Application built and deployment

For building and deploy app you need to clone git repository with project. Then 'cd' to workflow directory and execute '*mvn liquibase:update*' db will be created at db4free.org repository, you can create your own login password and put this information into pom.xml file or use default login 'logiweb' password 'secret'. Then you need to packege and deploy app on you server or use other platform like Herocu for deployment. For Herocu you need only install it in you wirkflow project directory and init it '*herocu create*' then execute '*git push herocu master*' and '*heroku open*' for visiting deployed web site. For deploeing your app at tomcat you need to describe user with username 'admin' and password 'admin' and set then role 'roles=manager-script,manager-gui' in tomcat user config file. And then execute maven comant in project folder 'mvn clean install' and 'mvn tomcat7:deploy' or 'mvn tomcat7:redeploy' if we got some exceptions. Then you can visit http://localhost:8080/ and enter admin, admin add new manager and use the system functionality.

Sonar project statistic



Lines Of Code
4 030
Java

Files
77
Directories 14
Lines 5 280

Functions
233
Classes 77
Statements 1 594
Accessors 160

Debt
5d

Issues
181

Blocker	0
Critical	0
Major	89
Minor	92
Info	0

Documentation
35.9%
Public API 223
Pub. Undoc. API 143

Comments
7.0%
Comment Lines 302

Complexity
753
/Function 3.2
/Class 9.8
/File 9.8

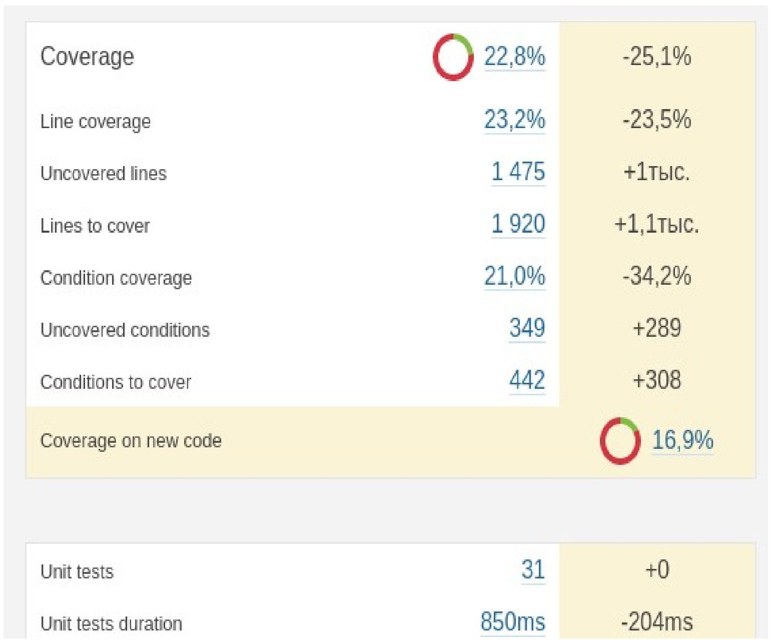
Unit Tests Coverage
22.8%
Line Coverage 23.2%
Condition Coverage 21.0%

Unit Test Success
100.0%
Failures 0
Errors 0
Tests 31
Execution Time 850 ms

Duplications
1.2%
Lines 64
Blocks 4
Files 3

UNRESOLVED ISSUES

Total 182



Coverage	22,8%	-25,1%
Line coverage	23,2%	-23,5%
Uncovered lines	1 475	+1тыс.
Lines to cover	1 920	+1,1тыс.
Condition coverage	21,0%	-34,2%
Uncovered conditions	349	+289
Conditions to cover	442	+308
Coverage on new code		16,9%

Unit tests	31	+0
Unit tests duration	850ms	-204ms

Another information

In first part of my app i used some patterns like: singletone, factory, frontcontroller. Singletone was creating for injection classes to other layers, factroy controller return implementation of interfase to execute comming request. FrontController handles all income requests and filter them.

Also was created some addition classes like SMS and Email utils for sending notifications.

I create small algorithm for analysing route for driver. If there is some cargoies to same city, then i can combine them. For security check was used spring security and rewriting all authentification classes.

All infarmation about exceptions getting from properties where you can get exception description by exception code.