Technical solution description

LogiWeb

Author: Nazukin Dmitry

1. **Content**

During this work an application that emulates the information system of some truck company was developed. Managers are able to manage company staff, map, trucks and orders (create, read, update, delete etc.). Drivers are able to get information about their appointment (order’s number, order’s items, co-drivers information, truck’s information) and manage order’s state (load and deliver cargos, start and finish order). Generally, all functionality from technical task was implemented.

Besides, additional features was implemented:

* Registration, autentification and authorization. There are 3 employee’s type: NEW (newly registered user, they doesn’t have any functionality), DRIVER (driver’s functionality), MANAGER (manager’s functionality).
* Map creation: managers are able to create and edit cities, add interval’s information.
* Order editing and deleting: managers are able to edit order, add points, delete points, change drivers.

## 2. Used technologies

Used instruments:

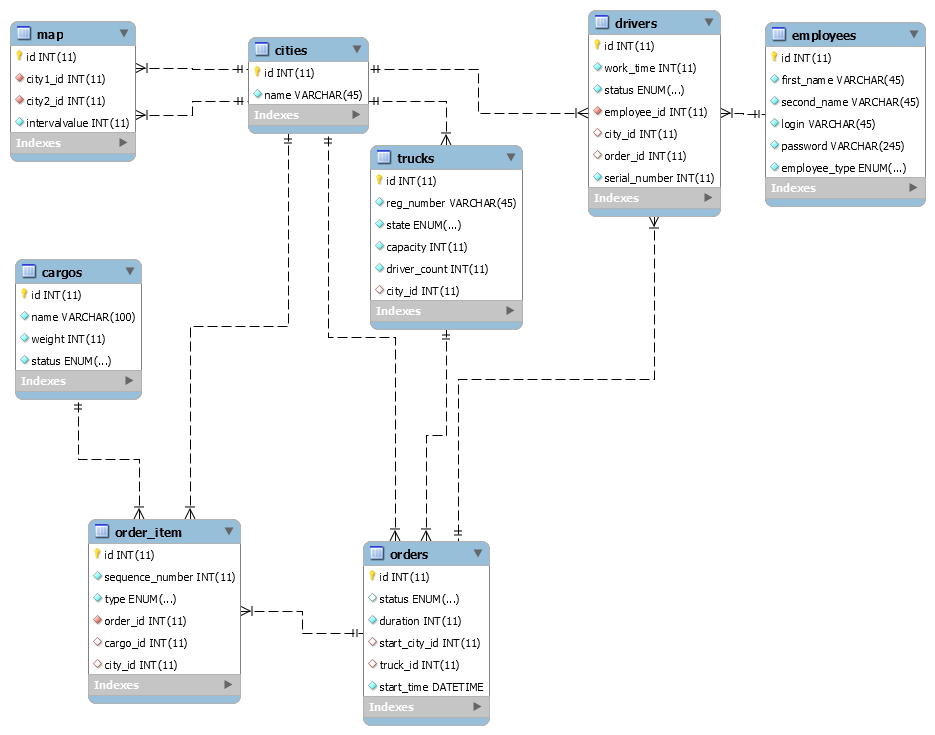
* Intellij IDEA Ultimate v15.0.3
* Apache Maven 3.3.3
* Wildfly 10.0.0 Final
* MySQL 5.6
* MySQL WorkBench
* SonarQube 5.4

Technologies:

* Spring 4.2.5
* JPA 2.2
* EJB 3
* JSF 2.1
* Jax-rs
* Log4j 1.2.17
* JUnit 4
* Mockito
* Bootstrap 3
* JQuery

# Implementation

* 1. **Database scheme:**



## Project structure

Project consists 3 modules:

* mainapp. Main application;
* rest-service. Rest api for drivers systems;
* rest-client. Rest client, which uses rest api.

## Architecture

The application has 4 layers defining its architecture. Each layer uses only the functionality of the previous layer.

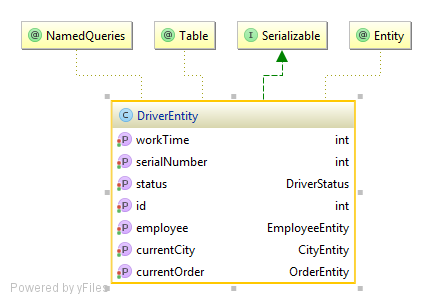
1. Model
   * Entities.
   * DAO.
2. Services, which include business logic.
3. Controllers process requests and parameters from UI and dispatch queries.
4. View layer consists of JSP-pages for the main application and JSF-pages for rest-client.

## Model layer

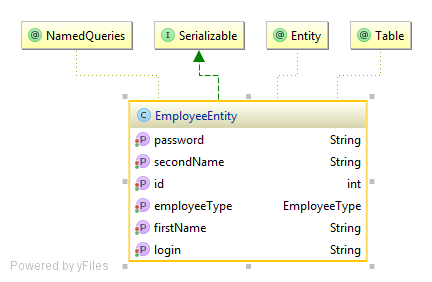
## The entities

My application has 8 entities, defined in the corresponding classes:

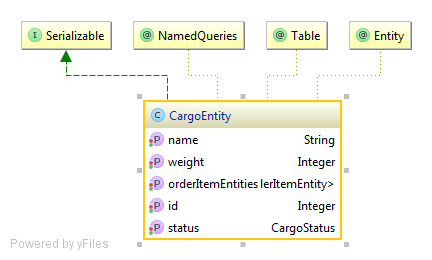
* DriverEntity



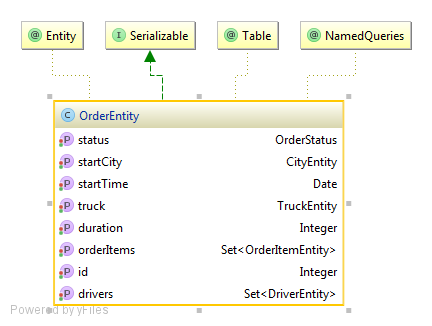
* EmployeeEntity



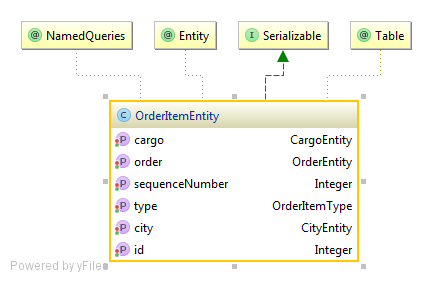
* CargoEntity



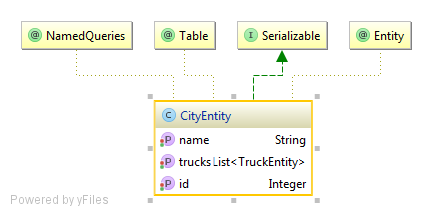
* OrderEntity



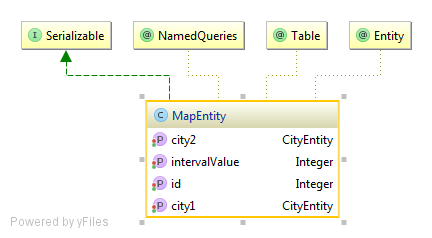
* OrderItemEntity



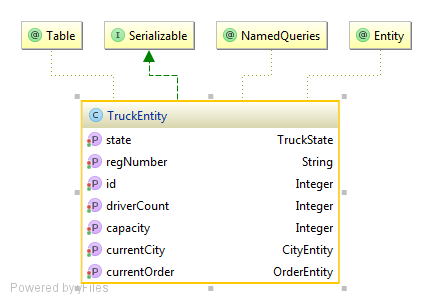
* CityEntity



* MapEntity



* TruckEntity

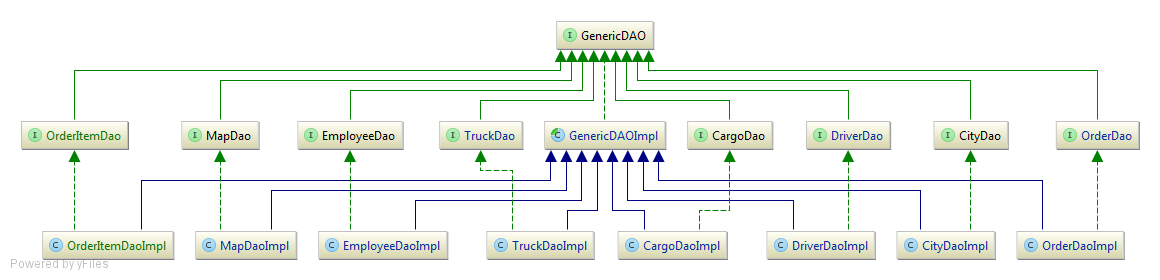


* + 1. **DAO**

DAO layer consists interface and implementation packages.

There is an implementation with the realization of the methods for each interface. The realization of the CRUD methods is done in the GenericDAOImpl class. Other methods are realized in the corresponding implementation classes. Entitymanager is being injected by @PersistenceContext annotation using Spring DI. Each of the DAO implementation classes is annotated with a @Repository annotation.

Dao layer’s diagram:



## Service layer

Service layer consists of API package and its implementation. Each of the service implementation classes is annotated with a @Service annotation. Transactions are declared at this level, each method of the service is performed in a transaction. The methods are annotated with @Transactional annotation, so Spring handles transactions.

Services:

* *CityService*. It does the functionality for cities management. CRUD.
* *MapService*. It does the functionality for map management (intervals between cities).
* *DriverService*. It provides functionality for working with driver entities. CRUD operations, specified selection, etc.
* *EmployeeService*. It saves new users, finds new users, change user’s role.
* *OrderService*. All functionality related to order management.
* *TruckService*. All functionality related to truck management.
* *UserDetailsService*. This service is used by Spring Security to validate users of the application. Implements “org.springframework.security.core.userdetails.UserDetailsService”.
  1. **Controller layer**

This layer holds the classes which are responsible for the interaction between UI and service classes. Controllers are based on the Spring MVC framework, have @Controller annotation.

Controllers:

* DriversController
* NewUsersController
* HomeController
* MapController
* OrdersController
* RegistrationController
* TrucksController
* DriverPageController
* GlobalExceptionController

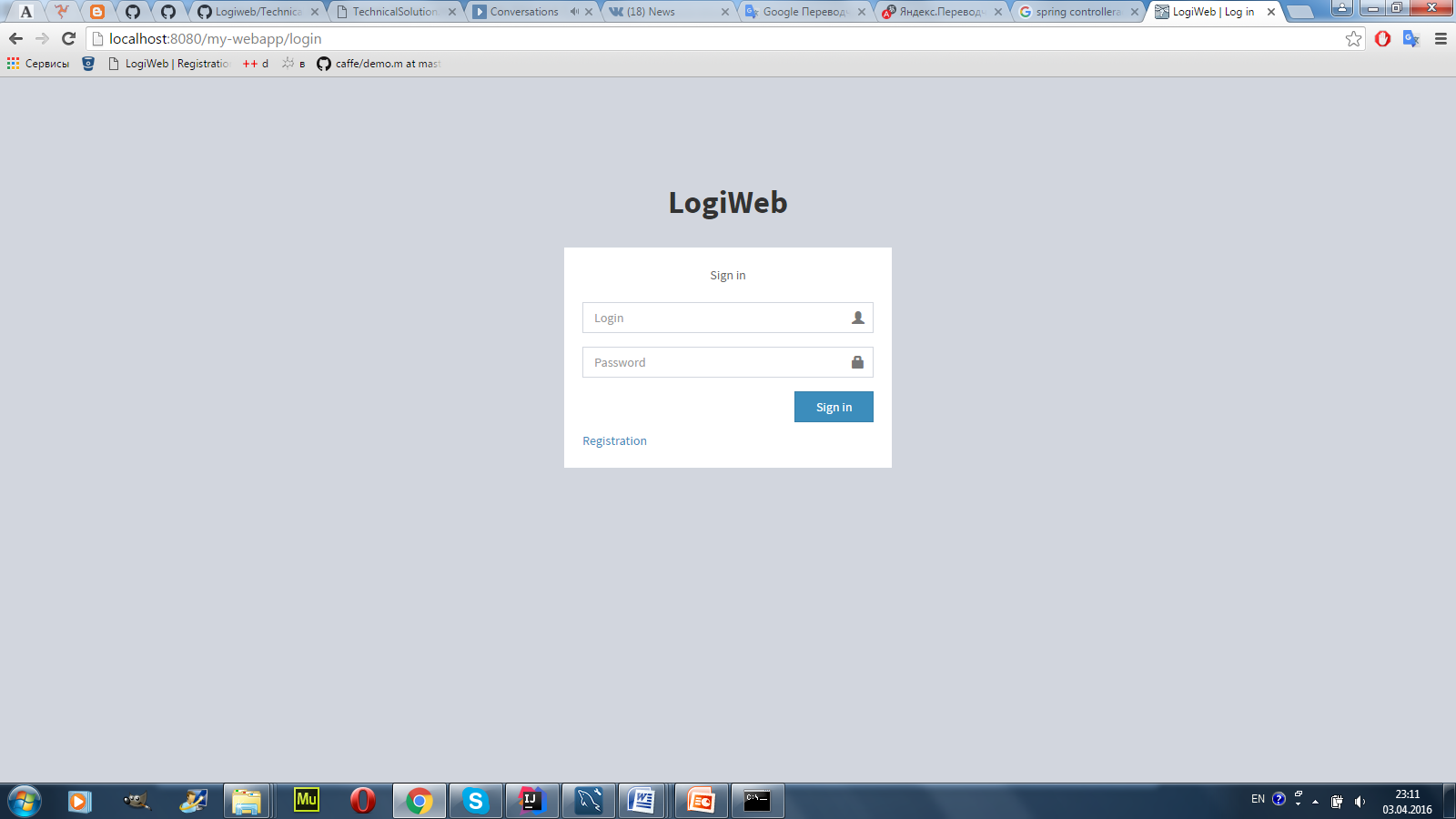
GlobalExceptionController handles all exceptions from controller layer. It is annotated @ControllerAdvice.

* 1. **View layer**

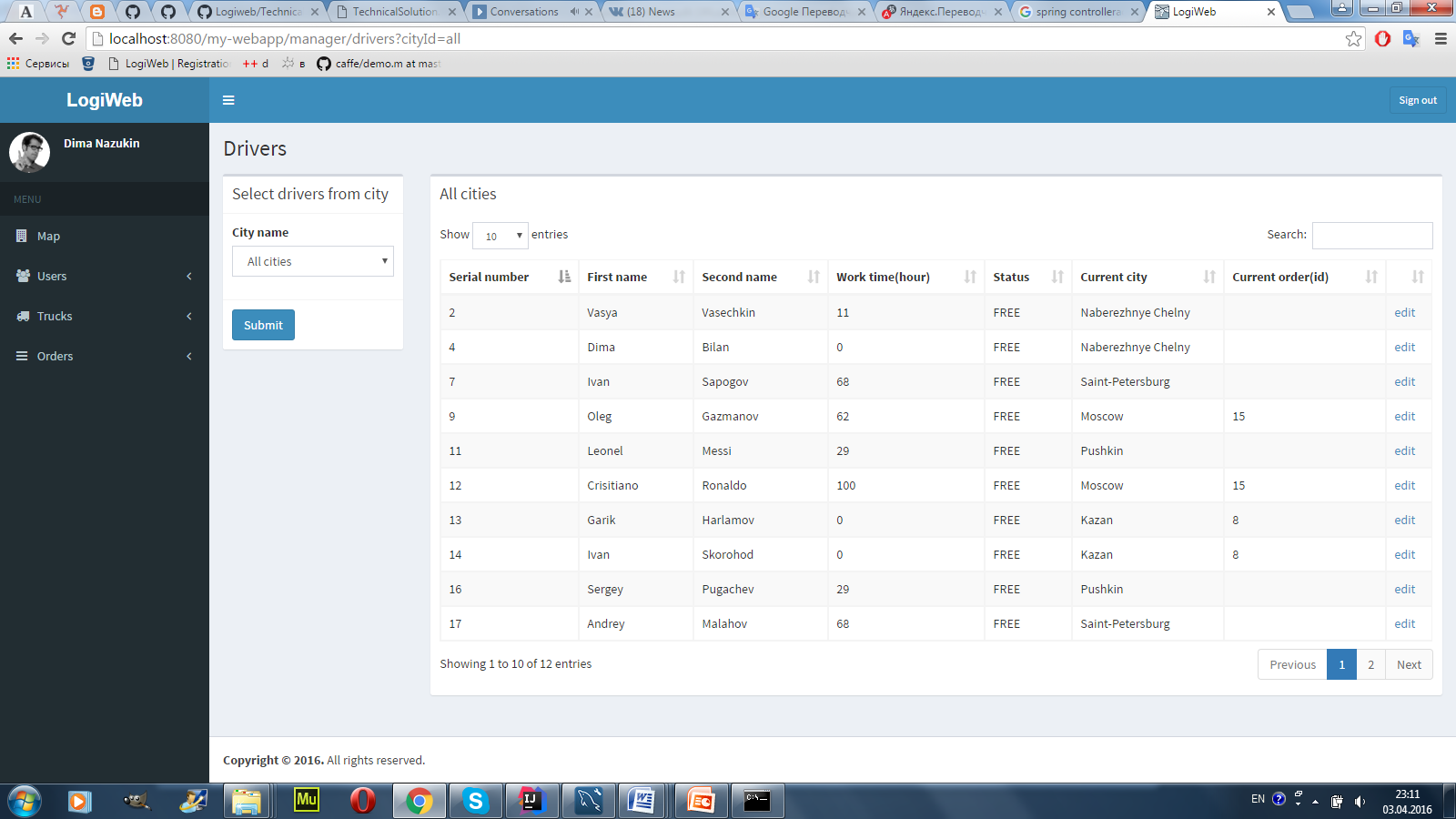
This layer is responsible for UI. Most part of design in main application is made using Bootstrap framework. All jsp, js, css are in webapp package. Rest-client is designed using JSF and Bootstrap. AdminLTE is used as a template.

Some pages:

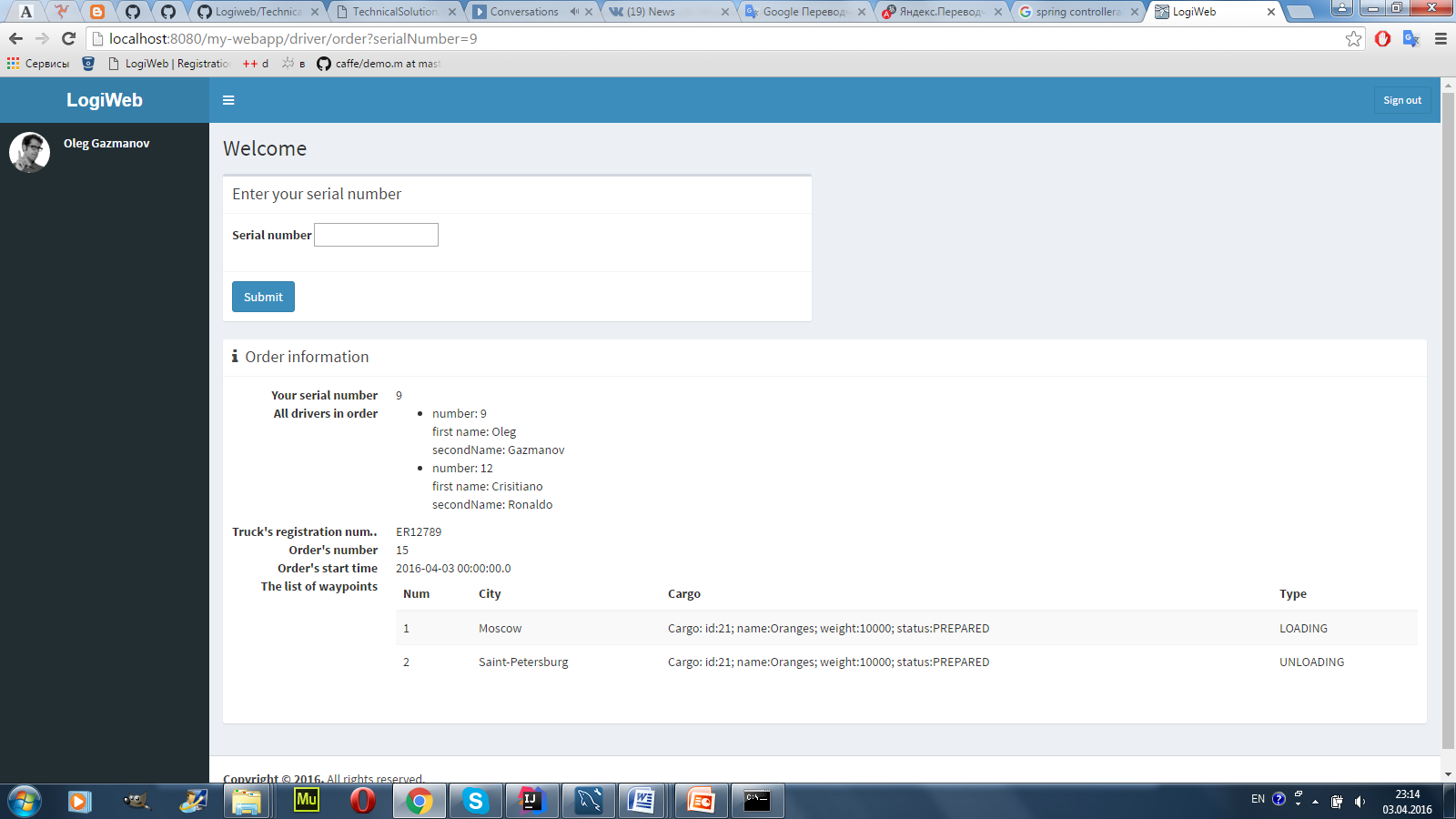
Login page



One of manager’s pages



Driver’s page



1. **Security**

Application security provided by Spring Security. Spring security is configured in SecurityConfig.class. Login and password are stored in database. Password is bqrypt encoded. So drivers and usual users can’t access to manager’s pages without correct login and password.

1. **Logging**

Log4j is used for logging. It is configured to make FATAL, ERROR, WARN  logs to file and console. Each module has own log file. Log files are stored on wildfly server.

Example:

2016-04-03 23:13:38 WARN DriverPageController:51 - Wrong serial number!!!

com.tsystems.nazukin.logiweb.customexceptions.WrongSerialNumberException

at com.tsystems.nazukin.logiweb.service.DriverService.findByEmployeeIdAndSerial(DriverService.java:184)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:497)

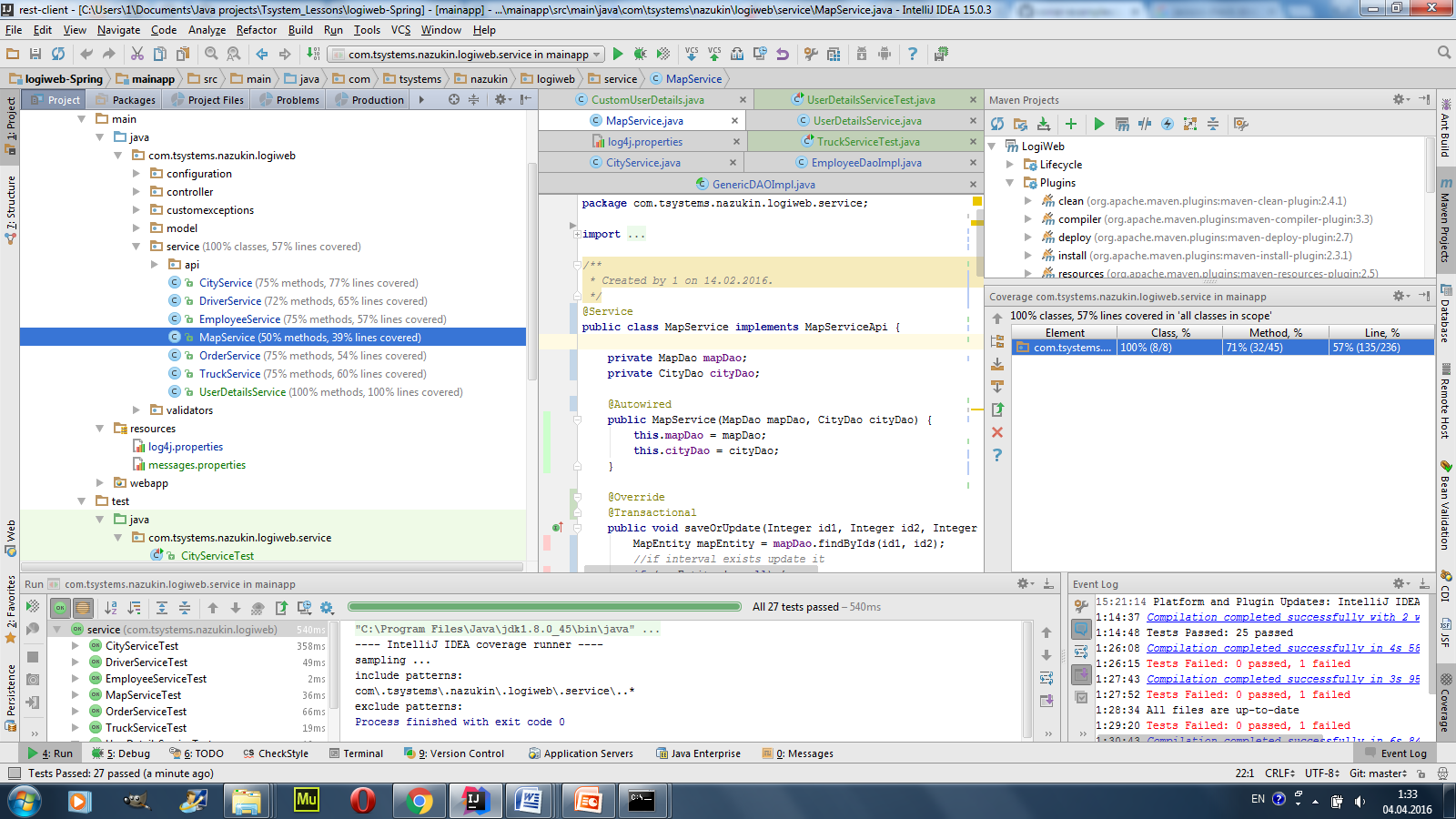
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:302)

at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:190)

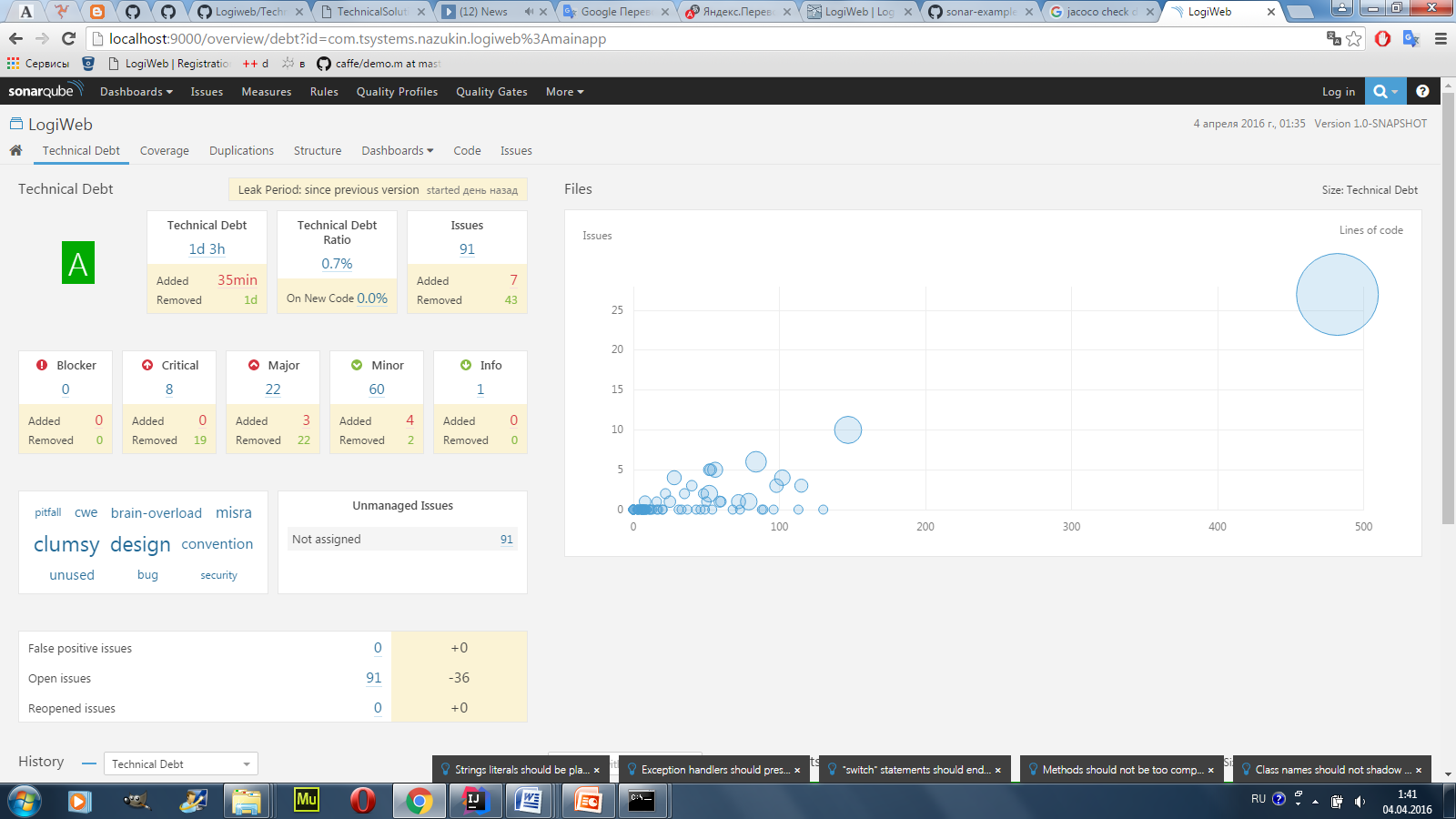
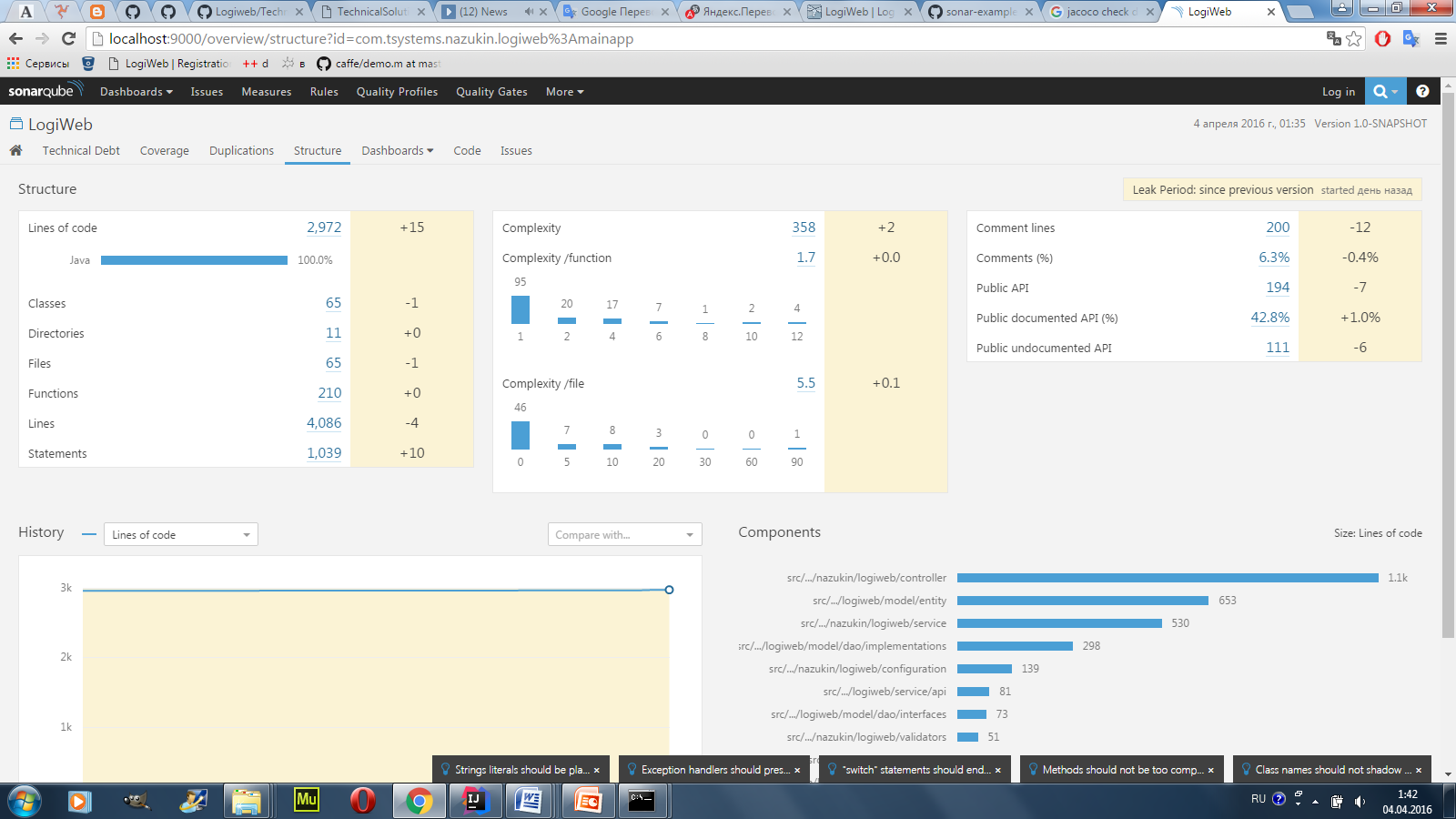
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:157)

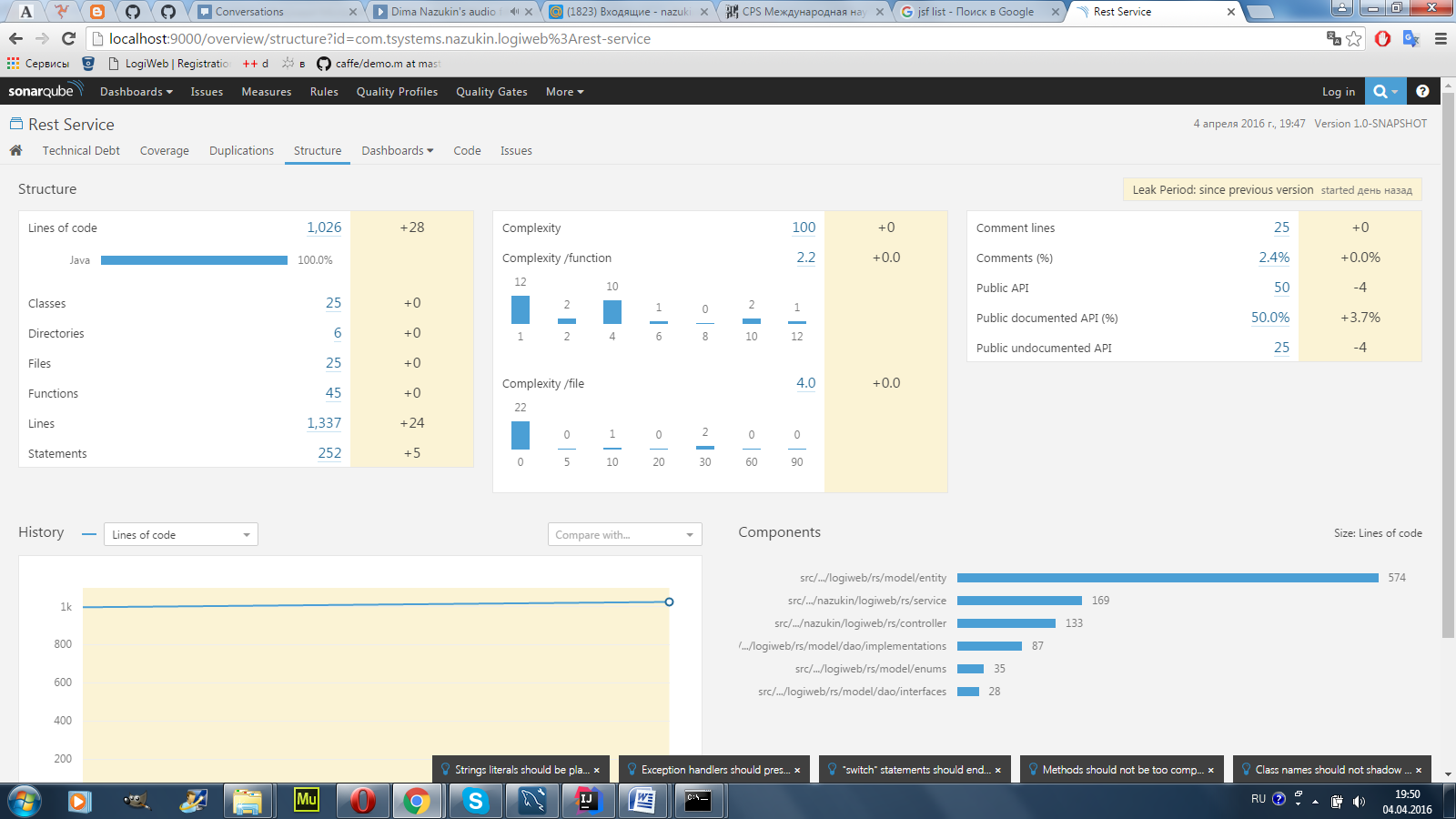
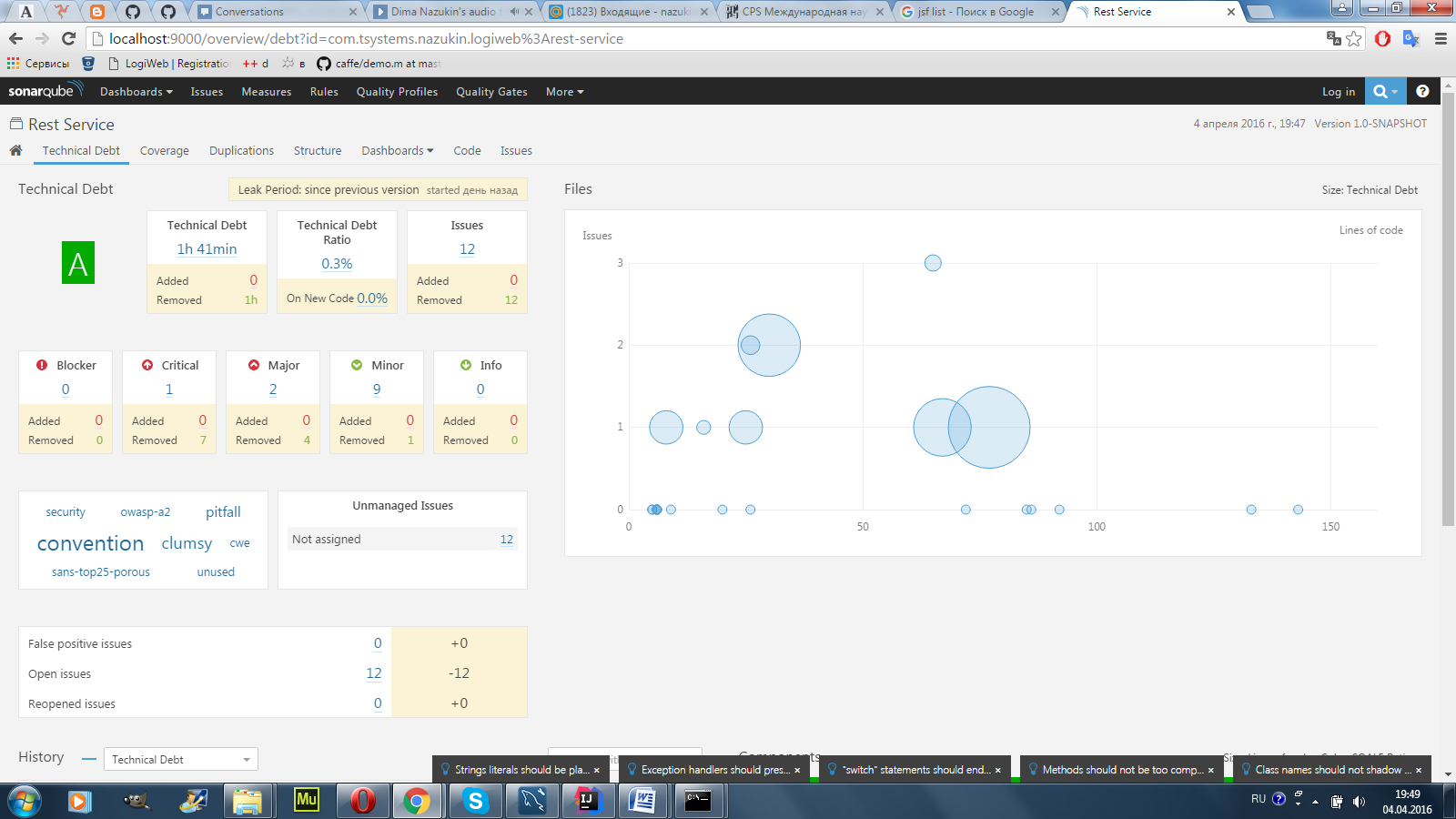
1. **Sonar and Unit tests.**

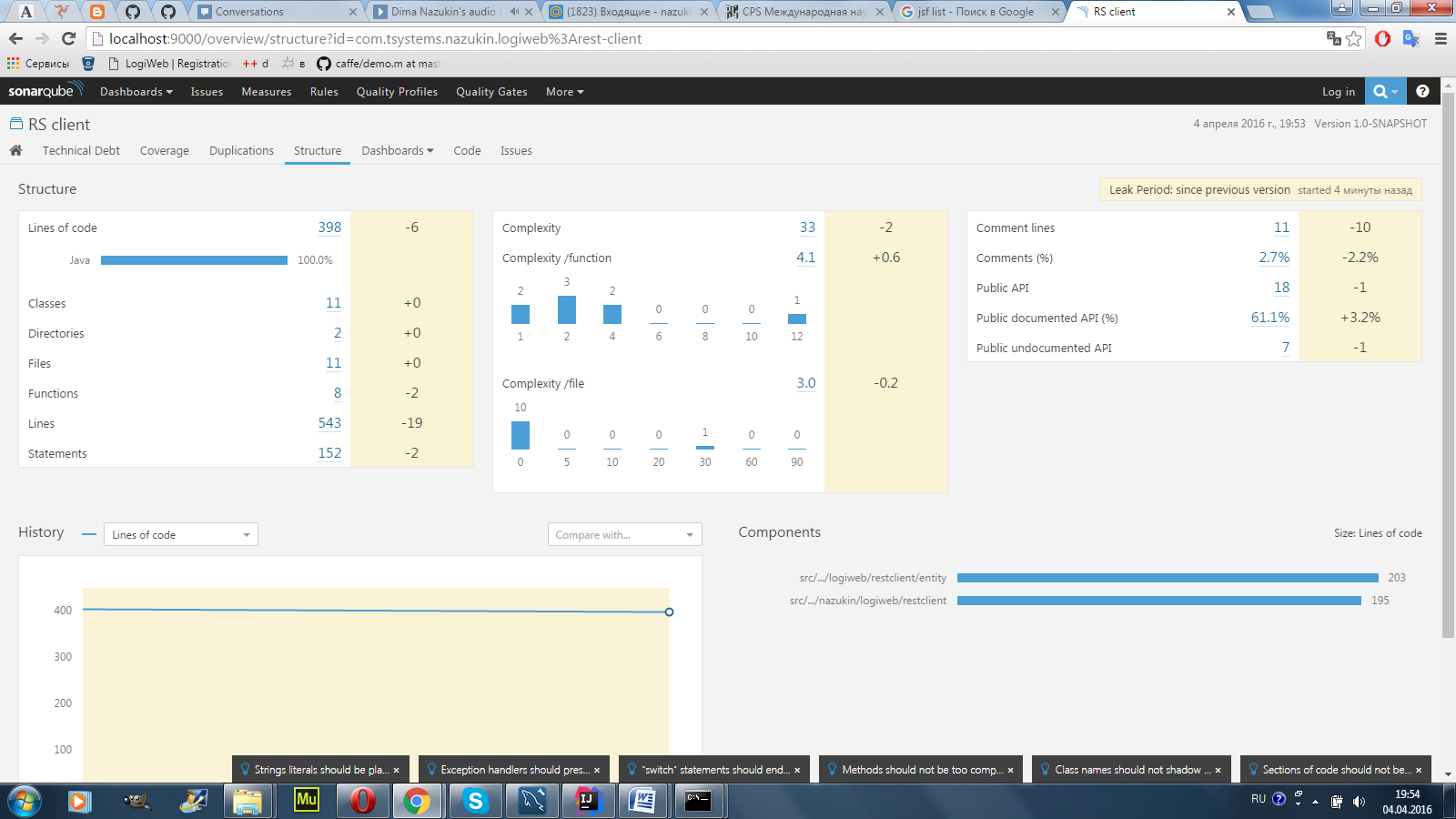
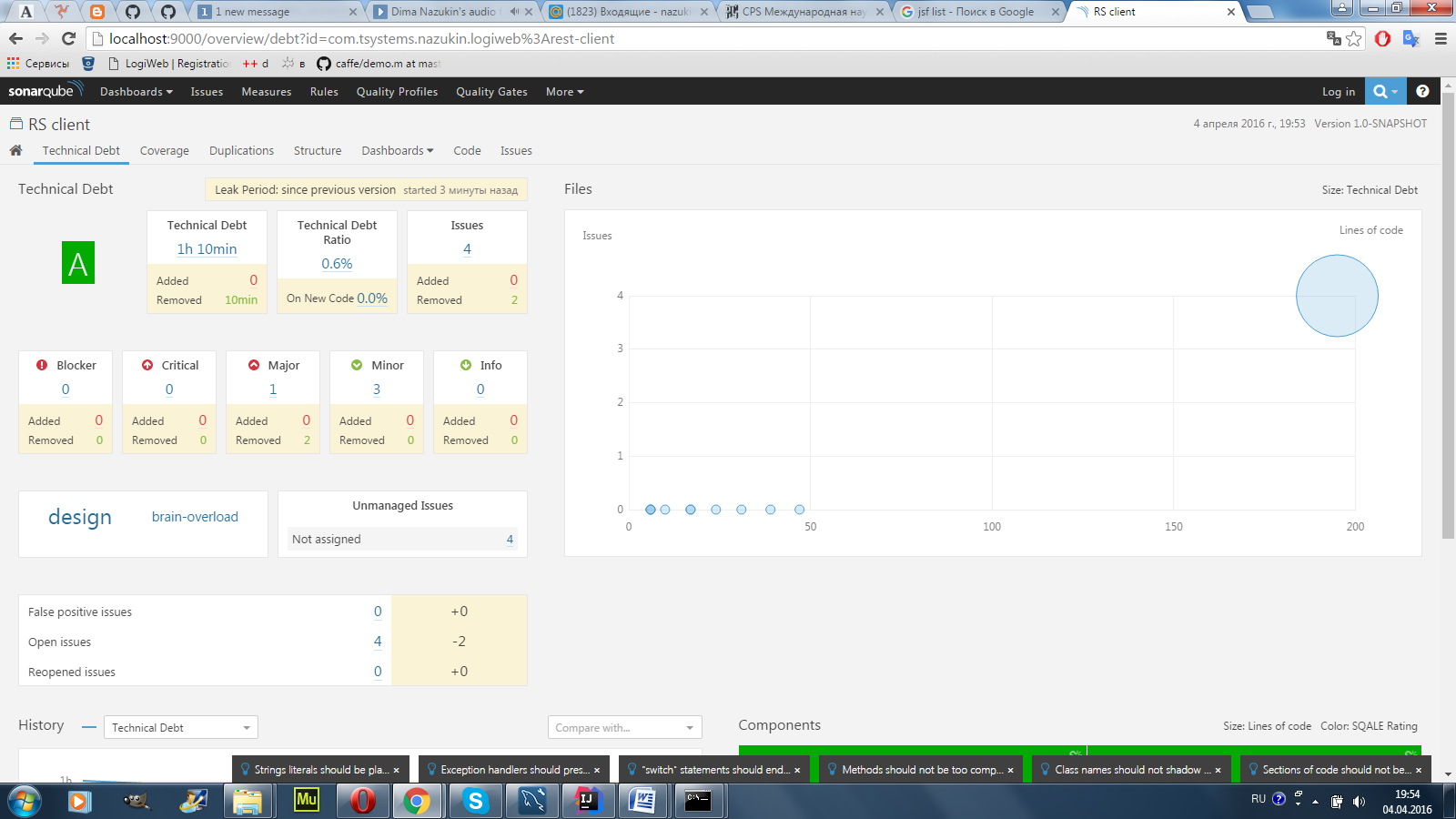
Unit tests was written only for service layer. Dao layer was mocked by Mockito. There are 27 tests and 57% lines coverage.



The application was analysed by SonarQube ­ a static code quality analyzer. Results are below.





There are some critical errors, but they are not errors at all.

1. **Build and deploy**

It is possible to build project whole project(all modules) by one command from root folder:

*mvn clean install;*

Also It is possible to deploy all modules to wildfly by one command from root folder:

*mvn wildfly:deploy;*