**T-SCHOOL**

**PROJECT NAME: LOGISTIKA**

**AUTHOR: MARTA EN**

**DATE: 30.10.2018**

**Version 1.0-SNAPSHOT**

| REVISION HISTORY | | | |
| --- | --- | --- | --- |
| DATE | VERSION | DESCRIPTION | AUTHOR |
| 30.10.2018 | 1.0-SNAPSHOT | Fist version | Marta En |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[DESCRIPTION 3](#_Toc528661009)

[SUMMARY 3](#_Toc528661010)

[PROJECT STRUCTURE 3](#_Toc528661011)

[LOGIWEB module 4](#_Toc528661012)

[TECHNICAL COMPONENTS 4](#_Toc528661013)

[DATA MODEL AND PERSISTENCE 6](#_Toc528661014)

[BUSINESS LOGIC 8](#_Toc528661015)

[FRONTEND 9](#_Toc528661016)

[UI for users with LOGIST role 9](#_Toc528661017)

[UI for users with DRIVER role 12](#_Toc528661018)

[SECURITY 14](#_Toc528661019)

[LOGGING 14](#_Toc528661020)

[TESTING 14](#_Toc528661021)

[DEPLOYMENT 15](#_Toc528661022)

[TABLEAU module 16](#_Toc528661023)

[TECHNICAL COMPONENTS 16](#_Toc528661024)

[BUSINESS LOGIC 16](#_Toc528661025)

[FRONTEND 17](#_Toc528661026)

[LOGGING 17](#_Toc528661027)

[DEPLOYMENT 17](#_Toc528661028)

[POSSIBLE FUTURE DEVELOPMENTS 18](#_Toc528661029)

# DESCRIPTION

## SUMMARY

The application is intended to support logistic company business (groupage freight), and covers following business processes:

* Planning transportation of incoming orders (intended users - logistic managers)
* Executing transportation and delivery (intended users – drivers)
* Business reporting (available for use by all users)

## PROJECT STRUCTURE

Maven-based project (group id = **com.marta**, artifact id = **logistika**) comprising two modules:

* **logiweb**: web application to manage logistic planning and delivery operations
* **tableau**: web application to view business reports

Logiweb module is responsible for data persistence and core business logic. To make it possible for other applications to access data managed by it, it also provides a REST controller delivering certain business data and publishes notifications on business data updates on a messaging server.

Tableau module is a REST client subscribed to messaging server updates. Upon every update notification, it re-requests data from Logiweb module to represent it to the user.

# LOGIWEB module

## TECHNICAL COMPONENTS

External applications needed to run the Logiweb module:

* MySQL 5.7
* Tomcat 9.0.8
* ActiveMQ 5.15.6

Internal components (dependencies):

| COMPONENT (DEPENDENCY) | VERSION | NOTE |
| --- | --- | --- |
| ***Framework*** | | |
| org.springframework.spring-context | 5.1.0.RELEASE |  |
| ***Persistence and validation*** | | |
| org.springframework.spring-orm | 5.1.0.RELEASE |  |
| org.springframework.spring-jdbc | 5.1.0.RELEASE |  |
| mysql.mysql-connector-java | 5.1.46 |  |
| org.hibernate.hibernate-core | 5.3.0.Final |  |
| org.hibernate.hibernate-entitymanager | 5.3.0.Final |  |
| javax.validation.validation-api | 2.0.1.Final |  |
| org.hibernate.hibernate-validator | 6.0.13.Final |  |
| ***Web*** | | |
| org.springframework.spring-web | 5.1.0.RELEASE |  |
| org.springframework.spring-webmvc | 5.1.0.RELEASE |  |
| javax.servlet.javax.servlet-api | 3.1.0 |  |
| javax.servlet.jsp-api | 2.0 |  |
| javax.servlet.jstl | 1.2 |  |
| ***Security*** | | |
| org.springframework.security.spring-security-web | 5.1.0.RELEASE |  |
| org.springframework.security.spring-security-config | 5.1.0.RELEASE |  |
| org.springframework.security.spring-security-taglibs | 5.1.0.RELEASE |  |
| ***Websocket*** | | |
| org.springframework.spring-websocket | 5.1.0.RELEASE | Extra to initial task |
| org.springframework.spring-messaging | 5.1.0.RELEASE | Extra to initial task |
| org.webjars.sockjs-client | 1.1.2 | Extra to initial task |
| org.webjars.stomp-websocket | 2.3.3 | Extra to initial task |
| ***Messaging*** | | |
| org.springframework.spring-jms | 5.1.0.RELEASE |  |
| org.apache.activemq.activemq-spring | 5.13.3 |  |
| ***Logging*** | | |
| log4j.log4j | 1.2.17 |  |
| org.springframework.spring-aop | 5.1.0.RELEASE |  |
| org.aspectj.aspectjtools | 1.9.2 |  |
| ***Testing*** |  |  |
| org.springframework.spring-test | 5.1.0.RELEASE |  |
| junit.junit | 4.12 |  |
| org.mockito.mockito-all | 2.0.2-beta |  |
| ***UI*** | | |
| org.webjars.webjars-locator | 0.30 |  |
| org.webjars.bootstrap | 4.1.3 |  |
| org.webjars.jquery | 3.3.1 |  |
| org.webjars.font-awesome | 5.3.1 |  |
| org.webjars.handelbars | 4.0.11-1 | Extra to initial task |
| ***Other support*** |  |  |
| net.sf.dozer.dozer | 5.4.0 |  |
| io.craftsman.dozer-jdk8-support | 1.0.4 |  |
| com.fasterxml.jackson.core.jackson-databind | 2.9.6 |  |

Comments on extra components:

* websocket is used to arrange update of drivers’ instructions upon their buddies’ actions
* handlebars tool is used in templating the ui screens

## DATA MODEL AND PERSISTENCE

Entity relationship model is shown below in Figure 1.

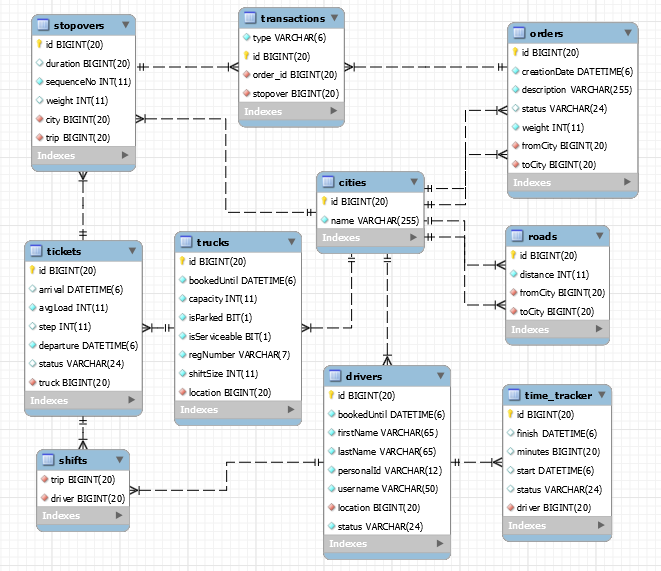


Figure . Entity Relationship Model

Placed orders are stored in **orders** table and show cargo description and weight, as well as departure and destination points. The order entity also has status field (***NEW*,*ASSIGNED*,*READY\_TO\_SHIP*,*SHIPPED*,*DELIVERED***).

As it is assumed that the business is groupage freight, multiple orders will be delivered with single truck. Core entity to manage transportation is trip ticket (**tickets** table).

Trip **ticket** contains following core data:

* planned departure date and time,
* **truck** assigned to perform the delivery (many-to-one relationship with **trucks**)
* list of drivers (**shift**) assigned to drive the truck (many-to-many relationship with **drivers**),
* list of **stopovers** (destinations) (one-to-many relationship with **stopovers**),

and some technical support data:

* ticket id,
* ticket status (***created, approved, running, closed***),
* current step
* estimated arrival date and time
* average load (weighted with distance taken by every cargo item)

Every **stopover**, in its turn, has:

* sequence number (stopover sequence within the trip),
* target city,
* list of load / unload **transactions** to be fulfilled at this stopover,
* total cargo weight at leaving this stopover.

**Transactions** can be of load and unload type (realized in object-relational mapping through inheritance from abstract transaction entity, with single table inheritance strategy), and every transaction is a record containing:

* link to stopover,
* transaction type (load or unload), and
* order that must be loaded or unloaded.

**Drivers** and **trucks** are resources used in transportation, and respective tables contain the fields necessary to plan the transportation: such as location, status, time until which they are booked. Trucks also are marked with shift size, capacity limit and serviceability, and for drivers work time is tracked (**time-tracker** table) to ensure meeting the monthly workload limits.

**Cities** and **roads** entities are used for route search. Cities mark available stopovers (pick-up and delivery points), and roads describe available direct routes between cities.

On the application side, persistence is realized using Spring-ORM framework and Hibernate as JPA provider. Transactions are managed with Entity Manager.

## BUSINESS LOGIC

To provide business logic, following services exist:

**CityService**, **RoadService** - to manage entities of cities and roads;

**DriverService**, **TruckService** – to manage entities of drivers and trucks;

**OrderService** – to register and view customer orders;

**TripTicketService** – to manage operations with trip tickets, including:

1. creating, filling and approving the ticket (logistic manager operations):
   * create ticket (includes assigning a truck),
   * add single or multiple orders to ticket (includes truck capacity limit check),
   * remove order from ticket,
   * approve ticket (includes assigning drivers, with monthly workload limit check);
2. processing ticket by drivers on the road:
   * get instruction for the driver,
   * register truck reaching next stopover,
   * register load and unload operations being completed,
   * register driver getting online or taking/finishing a break;
3. general ticket search and selection operations;

**TimeTrackerService** – to keep drivers’ time records and calculate monthly workload;

**TableauService** – to inform external users on business data updates.

## FRONTEND

Frontend part is based on JSP.

Taglibs used: JSTL, Spring tags.

Templating engine: Handlebars.

Libraries used: Bootstrap, JQuery, Fontawesome.

Screenshots of the main pages come in Figures 2-8 below.

After login, a user with the LOGIST role will be redirected to his core working screen /orders, and user with the DRIVER role – to his core working screen /logiweb.

### **UI for users with LOGIST role**

Navigation pane for logistic manager includes:

 (/orders) core screen to work with incoming orders and to compile trip tickets

 (/orders/view) order list

 (/tickets/view) trip ticket list

 settings drop-down submenu leading to:

 (/destinations) manage cities and roads data

 (/trucks) manage trucks data

 (/drivers) manage drivers’ data

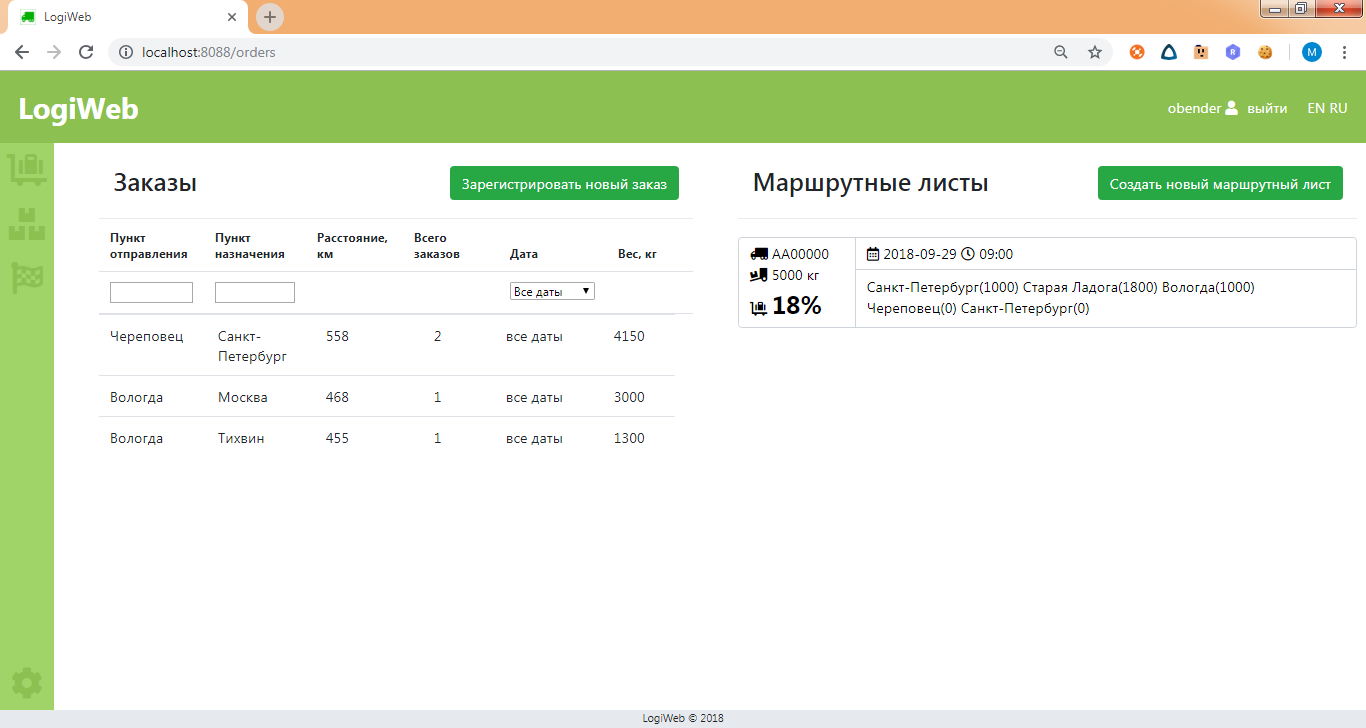


Figure . Core screen for logistic manager

Core working screen for logistic manager is divided in two sections:

* to the left: new orders, yet unassigned to trip tickets,
* to the right: trip tickets created but not yet approved for execution.

Actions available to the user from the core screen include:

* register new incoming order (by clicking the button on the top of orders section);
* create new trip ticket (by clicking the button on the top of tickets section);
* view new orders summed up by destinations, or (by double-clicking the respective line) view order list by selected destination;
* filter order list by departure city, destination city, order date;
* view list of trip tickets being compiled with summary including planned route and average truck load;
* go to full trip ticket info by double-clicking the ticket item;
* add order to ticket by dragging the order and dropping it on the ticket.

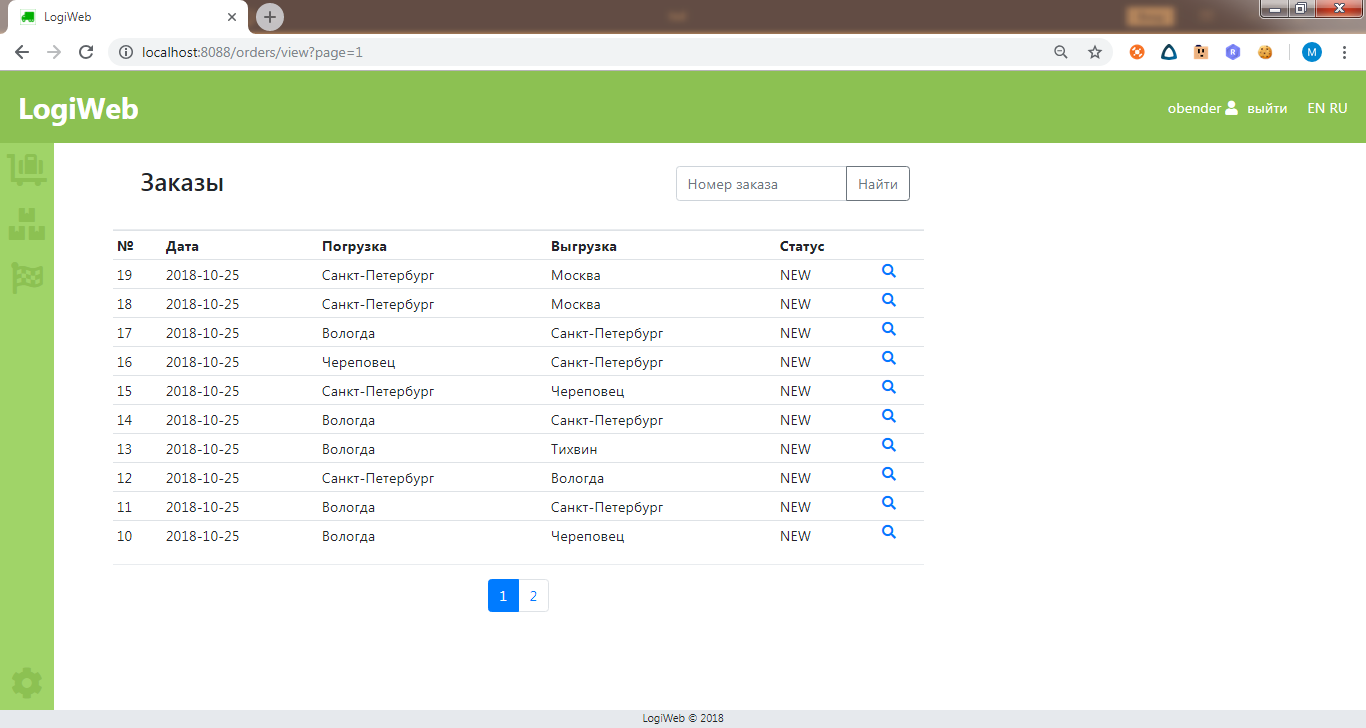


Figure . Sample of past data viewing screen - orders list

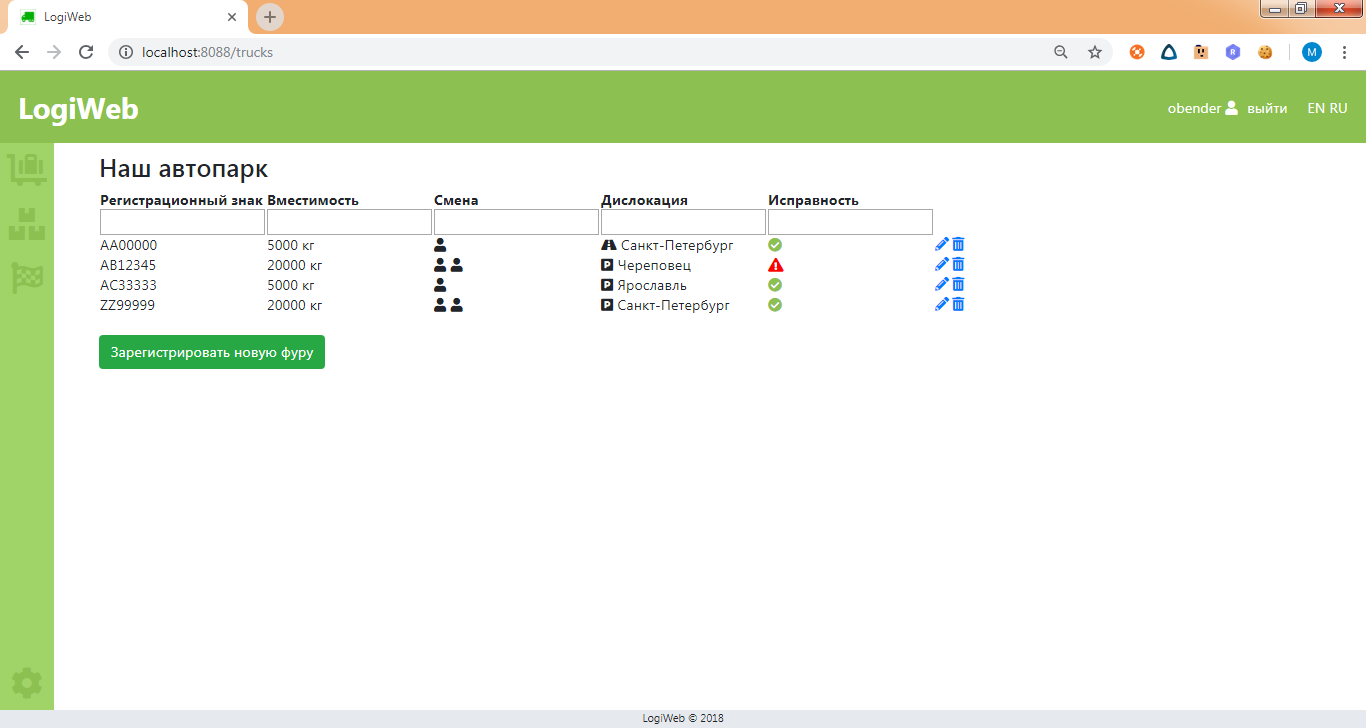


Figure . Sample of settings screens - truck data management

### **UI for users with DRIVER role**

Driver’s part is made as a single-page application.

Navigation pane for a driver includes:

 current instruction

 trip ticket info

 actual status (icon depends on driver’s actual status)

At the instruction screen the driver can read the next requested action and push the button to confirm the action is completed.

Note: The completion of ticket stage (reaching next stopover, loading / unloading) can be confirmed by any driver in the shift and will lead to instruction update for all the drivers in the shift.

At the status screen, the driver gets access to status change button – here he can take a break, or finish the break, or take the place of the first driver if he is seconding.

Note: there are two types of break possible:

\* road break – taken by all drivers simultaneously (any driver can request a break, and it will start for all the drivers in his shift),

\* stopover break – taken by the drivers individually (one driver may take a break, while the other will do the cargo handling).

Moving to the next stopover can’t begin if some of the drivers in the shift is offline or taking a stopover break – in this case all the other drivers will be forced to a waiting status, and the missing driver will receive additional alert that he is waited for.

Figure . Driver's screen - instruction tab – shift opening

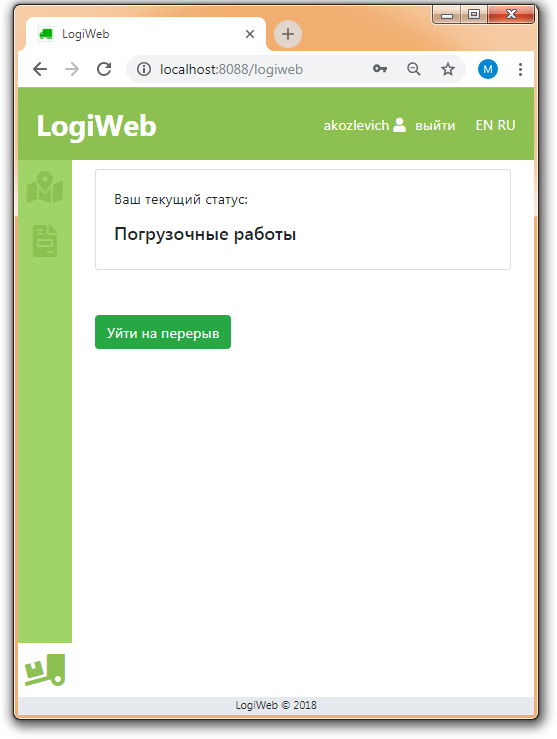


Figure . Driver's screen - status tab

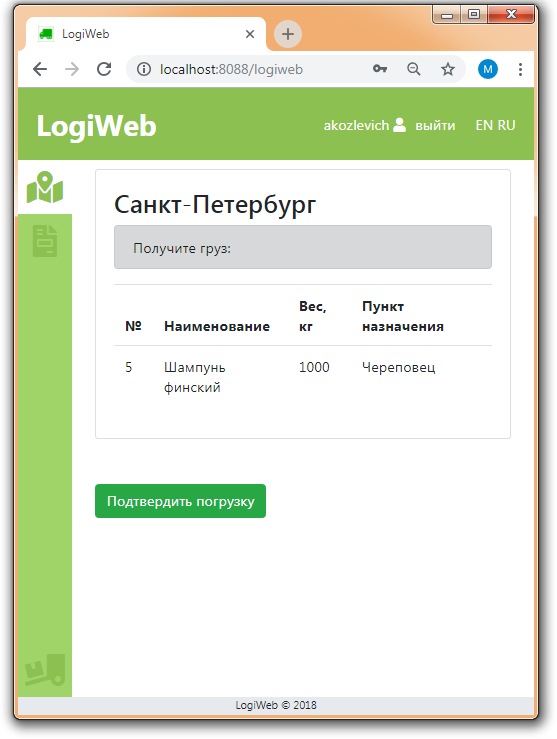


Figure . Driver's screen - instruction tab

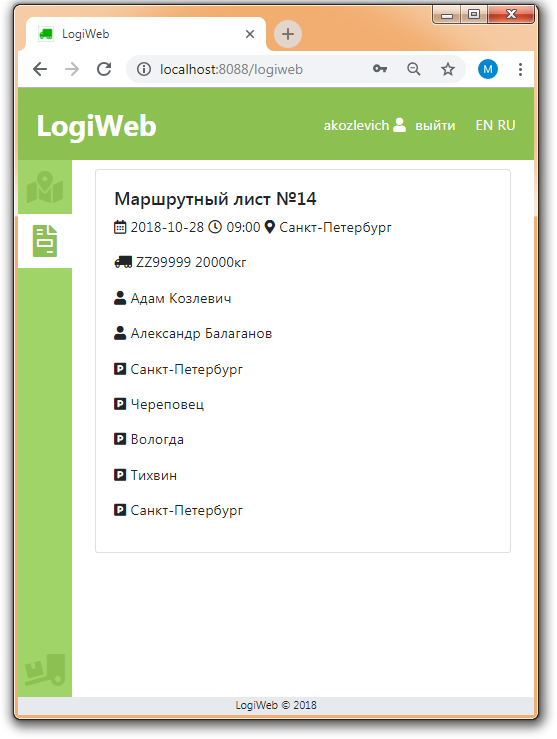


Figure . Driver's screen - ticket info tab

## SECURITY

Security is configured with Spring-security.

There are two user roles defined: **LOGIST** and **DRIVER**.

"/orders/\*\*", "/tickets/\*\*", "/trucks/\*\*", "/drivers/\*\*", "/destinations/\*\*" pages are limited to users with the **LOGIST** role.

“/logiweb/\*\*” page is limited to users with the **DRIVER** role.

"/tableau/\*\*" permits all users.

## LOGGING

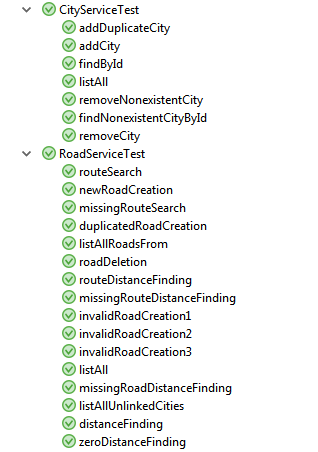
Log4j is used for logging, and core logging is configured with Spring-AOP.

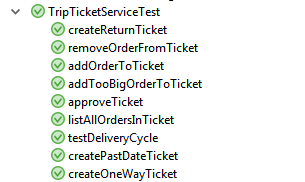
## TESTING

Unit tests (using JUnit and Mockito) are designed for the services.

Tests currently cover 62% of code lines in service layer.

List of tests:





## DEPLOYMENT

1. Set up the database. On MySQL Server, run the scripts from the .scripts directory:
   * 1-shema-and-user.sql: to create user and grant him necessary privileges;
   * 2-security.sql: to set up security database with users and create admin user
   * 3-db-structure.sql: to create business database structure
   * 4-test-data.sql: run this script if you wish to populate the application with sample data
2. Run the ActiveMQ server
3. Run the Tomcat server and deploy logistika.war to it

# TABLEAU module

## TECHNICAL COMPONENTS

External applications needed to run the Logiweb module:

* Wildfly 14.0.1
* ActiveMQ 5.15.6

Internal components (dependencies):

| COMPONENT (DEPENDENCY) | VERSION | SCOPE |
| --- | --- | --- |
| ***Framework*** | | |
| javax.javaee-api | 8.0 |  |
| javax.enterprise.cdi-api | managed (\*) | provided |
| org.jboss.spec.javax.annotation.jboss-annotations-api\_1.3\_spec | managed (\*) | provided |
| org.jboss.spec.javax.ejb. jboss-ejb-api\_3.2\_spec | managed (\*) | provided |
| org.jboss.spec.javax.faces.jboss-jsf-api\_2.3\_spec | managed (\*) | provided |
| org.hibernate.javax.persistence.hibernate-jpa-2.1-api | 1.0.2.Final |  |
| org.primefaces.primefaces | RELEASE | compile |

(\*) org.wildfly.bom.wildfly-javaee8-with-tools (version 14.0.1.Final) is used for version management of the other components

## BUSINESS LOGIC

This single-page web application is a REST client requesting all the necessary data at startup and re-requesting it later upon any next notification from the messaging server. Notification is pushed to the JSF side via websocket, and triggers page reloading.

## FRONTEND

Frontend is made with JSF and Primefaces. Sample screenshot is shown below.

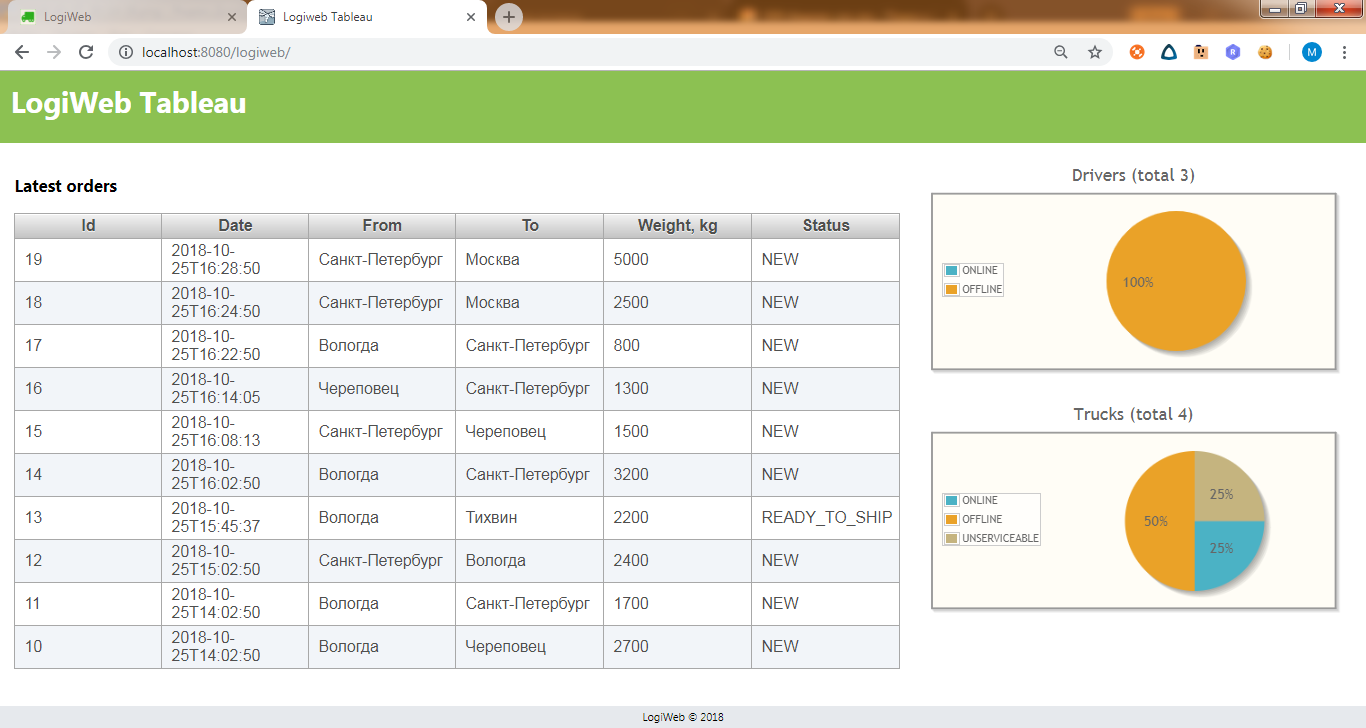


Figure . Tableau sample screenshot

## LOGGING

Log4j is used for logging. Logged events are message receipt and websocket push.

## DEPLOYMENT

1. Have the Logiweb module up and running (including MySQL server and ActiveMQ)
2. Run Wildfly server and deploy logiweb.war from to it

# POSSIBLE FUTURE DEVELOPMENTS

1. Order processing section can be extended by:
   1. Building the customer part of the application
   2. More sophisticated order management process (order editing, deletion, rejection at load time etc.)
2. Route planning can be developed using Google maps api or similar.
3. Driver master data maintaining should be a part of general employee data maintaining process.
4. Drivers’ work time tracking can be extended to provide data for payroll.
5. Tableau can be developed to show balance of requested transportation volumes and truck capacities \ driver availability by city.