**Logiweb**

**Website for cargo transportation**

**Technical Solution Description**

**Table of Contents**

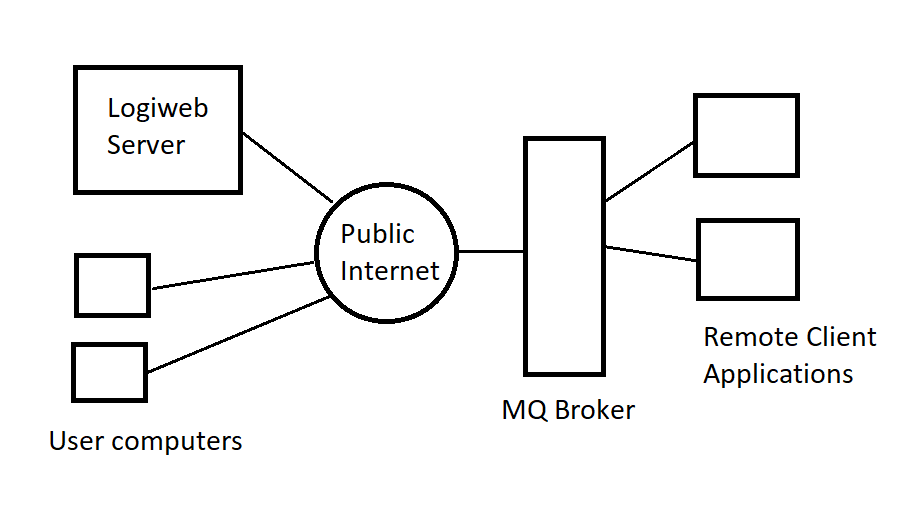
1. Logiweb System Overview
2. Specification of the Logiweb
   1. Technologies
   2. Business cases
   3. Implementations
   4. Data Model
   5. Architecture
      1. Controller layer
      2. Service layer
      3. DAO layer
   6. Views
   7. Database
   8. Tests
3. Logiweb
   1. User levels
   2. Live Monitoring Page
   3. Admin views
      1. Orders view
      2. Orders history view
      3. Trucks view
      4. Drivers view
   4. Driver views
      1. Work details view

**1 Logiweb System Overview**

Logiweb enables managing orders, trucks and drivers, change the status of and order during its execution in real time and also exchange of data in real time between the server and remote client application.

The service is available for logisticians, who create orders and distribute trucks and drivers for them, and for drivers, who delivering cargo. The remote client application available for all users.

The main components are:

* Logiweb Server
* Remote Client Application
* MQ Broker: providing connectivity between Server and remote client application.

**2 Specification of the Logiweb**

**2.1 Technologies**

Message Broker: Active MQ Artemis

**Server Side:**

Core: Java, Spring MVC, Security, Hibernate.

AS: Tomcat

Database: PostgreSQL

View: Thymeleaf, Bootstrap 4

Logging: Log4j2

Testing: Mockito and JUnit 5

Utils: Lombok

**Remote Client Application:**

Core: Java, EJB

AS: WildFly

View: JSF

Logging: Log4j

Testing: Mockito and Junit 5

**2.2 Business cases**

**Server Side:**

1) The logistician needs to create an order for several cargo.

2) Add to this order serviceable truck with capacity greater or equals max capacity of this order and current city of truck be the same as the start point of this order.

3) Add drivers to this order considering that the limit of hours worked in this month not been exceeded during order execution.and current city is the same as the order truck city.

4) Logistician need to view, delete, edit and create the drivers and trucks.

5) Driver needs to see his order, truck and co-drivers.

6) Driver needs to update his status and finish the shift.

7) Driver needs to change cargo status then they load and unload.

**Remote Client Application:**

1. User need to get full information about last 10 orders, drivers and truck numbers.

**2.3 Implementations**

**Server Side:**

1)The employee goes to the Orders tab and can see tables of orders. Orders it’s an entity reflect real order with waypoints and cargo, path, completion status of order, truck of this order and drivers who completed this order. Employee click on button create order and go to page with order creation form. This page contains buttons for add/remove quantity of waypoints, form where employees can choose cities for load and unload and cargo, at bottom button for creating order. If your path is suboptimal, the system will be alert about it and don’t create the order.

2)After Order created in column ‘truck’ employee sees a button to choose the truck for the order. After clicking on this button it gives a list of trucks which have status ‘serviceable’, not completed order, truck city the same as order start point and truck’s capacity no less then max capacity of the order. After in this list employees choose truck and click add. Now in the Order entity will be added to the Truck entity.

3)After adding truck to order, employees can add drivers. In the table appeared a button with available shift size. After clicking on this button it gives a list of drivers which have status ‘REST’, have not completed the order, hours worked in this month not greater then 176(during the execution of this order) and have the same city as a truck. Employees can choose drivers and click add. After that drivers get order details about this order and the Truck entity will be added to the Driver entity.

4)At tabs ‘Drivers’ and ‘Trucks’ employees can view all drivers and trucks. In column table in this tables employees can click on buttons edit or delete, and entity will be deleted or edit. On top of this tables located button for creating a truck or driver entity.

5-7) Drivers can go to the ‘Work details’ button and see order details and shift details. If the driver has not order for completion, he will see ‘You don’t have order for completed yet’ and cannot change his driver status. If a driver has an order for completed he will see his codrivers(it drivers that have the same order for completed), order and truck id, order’s path, remaining time in travel and table with waypoints. If a driver is located in the same city as needed for load or unload cargo, the driver can change status of cargo and after change this status can click on a button near the text ‘Change city to’, at this button will be written the next city point. After clicking on this button, remaining time in travel will decrease and the difference between current time and last time added to worked hours of drivers in shift.

**Remote Client Applicaton:**

1. User can go to main page and see full information that updated every time when server information is updated.

**2.4 Data Model**

**Server Side:**

User - have unique id, email, password and role, need to authenticate and authorize in the security system.

Driver - entity that reflects drivers. Relation: driver related to truck by @ManyToOne, because one truck entity may have many drivers.

Truck - entity that reflects trucks of this company. Relation: truck related to driver by @OneToMany, because drivers of this truck can relate only to it.

Order - entity that reflects order with waypoints, status and truck on which the order will be executed. Relation: order related to truck @OneToOne.

Waypoint - entity that reflects path points where cargo loading or unloading. Relation: waypoint related to order like @ManyToOne because one order has many waypoints. Related to Cargo like @OneToOne unidirection relation.

Cargo - entity that reflects cargo.

City - entity that reflects cities;

Road - entity that connects cities and defines distance between them. Relation: one road related to two cities by @ManyToOne

CountryMap - entity that reflects map for path creation.

PastOrder – entity that reflects completed orders. Have @ManyToMany relation with drivers.

**Remote Client Application:**

Information – summary information about drivers, trucks and orders.

DriverInfo – information about total number of drivers, available and unavailable unmbers.

TruckInfo – information about total, available, in use, faulty numbers of trucks.

OrderInfo – information about last 10 uncompleted orders.

Driver – id, name and surname of drivers.

Order – id, status, truck id, path and drivers.

**2.5 Architecture**

Use **MVC** architecture pattern.For building some entities use pattern **Builder.** For faster data retrieval from the database and data transfer between layers use **DTO** pattern.

**2.5.1 Controller Layer**

HomeController - home page endpoint.

AuthController - endpoint for login

DriverController - represent methods for the view of all drivers, have endpoints for creation, editing and delete drivers.

TruckController - represent methods for the view of all trucks, have endpoints for creation, editing and delete trucks.

OrderController - represent methods for the view of all orders, have endpoints for creation, adding truck and drivers to order, view cargo of order.

ProfileController - represent methods for the view of order and shift details of drivers, have endpoints for changing cargo and driver status, city and finish the shift when order is completed.

**2.5.2 Service Layer**

**Server Side:**

CargoService - view all cargo or cargo of order.

DriverService - provide methods for creating, reading, updating and delete drivers from drivers list from database.

OrderService - provide methods for creating and reading orders, reading and adding drivers and trucks for order.

TruckService - provided methods for creating, reading, updating and delete trucks from database

CountryMapService - provided methods for reading all cities and roads.

PathDetailsService - provided methods for calculating shortest path, max capacity on this path, approximate time it takes to overcome the path.

OrderDetailsService - provided methods for view order details, change cargo status and remaining path.

ShiftDetailsService - provided methods for view shift details, change driver status and shift active status, calculate worked hours and finish shift of completed order.

InformationProducerService – produce message with information about driver, trucks and orders.

InformationService – provide methods for reading information about drivers, trucks and orders.

InitializeListener – receive initialize message from client side.

EmailUniqueValidator, TruckIDUniqueValidator, WaypointsValidator – services for @Validation annotations.

DateTimeService – calculate time until end of month and refresh worked hours of drivers in the end of month.

UserDetailsServiceImpl - implementation for UserDetailsService, load user by email, if user has role EMPLOYEE logging entrance.

**Remote Client Application:**

InformationListener – service for receive message from mq broker and update information service.

InformationService – service for store information and update jsf bean.

InitializeService – service for send initial message to server.

**2.5.3 DAO Layer**

CargoDAO - find all Cargo entities and Cargo entities of order ID or of cargo ID, update Cargo and find cargo weight of cargo ID.

CountryMapDAO - find all City entities and return a list of CityDTO. Find City entity by city code and find all Road entities and return a list of RoadDTO.

DriverDAO - crud operation on Driver entity. Save WorkShift entity.

OrderDAO - crud operation on Order entity. Find Waypoints entities by order and return the WaypointDTO list and save Waypoint entities. Find TruckDTO and DriverDTO lists by parameters for order. Save and find OrderDetails entities.

OrderDetailsDAO - find and update order details entities.

ShiftDetailsDAO - find and update shift details entities.

TruckDAO - crud operation on Truck entity and count drivers which related to this truck.

InformationDAO – read information about trucks, orders and drivers.

ScheduleDAO – refresh drivers worked hours.

UserDAO - find user by email, return UserDTO.

**2.6 Views**

**Server Side:**

For UI page user thymeleaf with bootstrap 4. Have a page for creating, editing and view list of drivers, trucks, orders. User thymeleaf fragments for navbar and script for bootstrap. Profile view for shift and order details of driver. In exception folder have views for exceptions(404,403,500 and etc.).

**Remote Client Application:**

Use JSF pages that connected with ManagedBean and updated with using websocket technology. Have 500 and 404 exceptions pages.

**2.7 Database**

Table cities no relation, pk citi\_code field.

Table roads reference to cities table by fk city\_a/b\_code, pk road\_id field.

Table trucks reference to cities table by fk city\_code, pk id field.

Table orders reference to trucks table by fk truck\_id, pk id field.

Table order\_details reference to orders table by fk id, pk id field.

Table users no reference, pk id field.

Table drivers reference to users, cities, trucks, order\_details tables by fk id, city\_code, truck\_id, order\_details, pk id field.

Table work\_shifts reference to drivers by fk id, pk id field.

Table cargo no reference, pk id field.

Table waypoints reference to cities, orders, cargo tables by fk city\_code, order\_id, cargo\_id, pk waypoint\_id field.

Table orders\_history refelect completed orders, pk id same as completed order id.

Table past\_shifts reference to orders\_history, drivers table by fk id, driver\_id, pk id, driver\_id field.

**2.8 Tests**

**Server Side:**

ShiftDetailsServiceTest:

givenRestDriverStatus\_WhenShiftActiveTrue\_ThenItShouldReturnFalse, givenRestDriverStatus\_WhenShiftActiveFalse\_ThenItShouldReturnTrue,

givenDrivingDriverStatus\_WhenShiftActiveTrue\_ThenItShouldReturnTrue,

givenDrivingDriverStatus\_WhenShiftActiveFalse\_ThenItShouldReturnFalse - test validate method.

thenUpdateShiftDetailsThenreturnUpdateDetailsValues1/2 - test changeShiftDetails method for correct change entity.

OrderDetailsServicetest:

whenAllCargoStatusAreDeliveredReturnTrue, whenNotAllCargoStatusAreDeliveredReturnFalse - test orderIsCompleted method on correct work.

givenDriverId\_whenOrderDetailsExist\_thenReturnedDTOBeEqualsToExpectedDTO – test readOrderDetails method work correct and return correct values.

givenDriverId\_whenOrderDetailsNot\_exist\_thenReturnNull – test what if order details for driver not exist method return null.

givenDriverIDAndCargoIds\_whenOrderCompleted\_thenShiftDetailsStatusAndCargoStatusEqualsToUpdatedValues,

givenDriverIDAndCarogIDs\_whenOrderNot/Completed\_thenShiftDetailsStatusAndCargoStatusNot/EqualsToUpdated(Past)Values - test updateOrderByCargoStatus work correct and set driver status and cargo status properly

givenDriverIdAndOrderID\_thenRemainingPathAndTime(Not)Equals(Past)NewValues – test changeCity method for change values of orderDetails

PathDetailsServiceTest:

returnSamePathOfWaypointAsExpected1/5 - add wapoints to getPath method and result path should equals expected path that has been calculated before.

returnDistanceAsExpected1/5 - calculate distance of path and result should be equals distance that has been calculated before.

returnShiftHoursAsExpected1/5 - calculate shift hours of path and result should be equals shift hours that has been calculated before.

returnMaxCapacityAsExpected1/5 - calculate capacity of path and result should be equals capacity that has been calculated before.

PathParserTest:

whenParseListThenReturnRightString1/2, whenParseListThenReturnWrongString1, whenParseEmptyListThenReturnEmptyString - test parseLongListToString on correct parse.

whenParseSringThenReturnRightPreetyPath1/2 - test toPrettyPath method correct work.

whenParseEmptyStringThenThrowException, whenParseUncorrectStringThenThrowException - test parseStringToLongList throw PathParseException, if string empty or string uncorrect

whenParseStringThenReturnRightList1/2, whenParseStringThenReturnWrongList1 - test parseStringToLongList to correct parse.

DateTimeServiceTest:

whenBeforeAndOfMounthRemainsOneDay\_thenReturnTwentyFour, whenBeforeAndOfMounthRemainsOneHour\_thenReturnOne – test that method calculateTimeUntilEndOfMonth() work correct.

cornInRefreshWorkedHoursMethodWorkAsNeeded – test that cron expression in @Sheduled method is correct.

**Remote Client Application:**

givenMessage\_whenExpectedResultIsSame\_thenReturnedInformationIsEquals – test what listener get message and update InformationService.

givenMessage\_whenExpectedResultIsDifferent\_thenReturnedInformationIsNotEquals() – test is listener get message then this message will be not equals to another values.

whenMessageIsNotCorrect\_thenThrowMessageProcessingException – test if message is uncorrect then listener throw MessageProcessingException.

**3 Logiweb**

Access to the Logiweb application is made through a web browser.

**3.1 User levels**

There are 3 levels of users, each allocated with different authorizations for interaction with the Logiweb system. The three levels of user are the following:

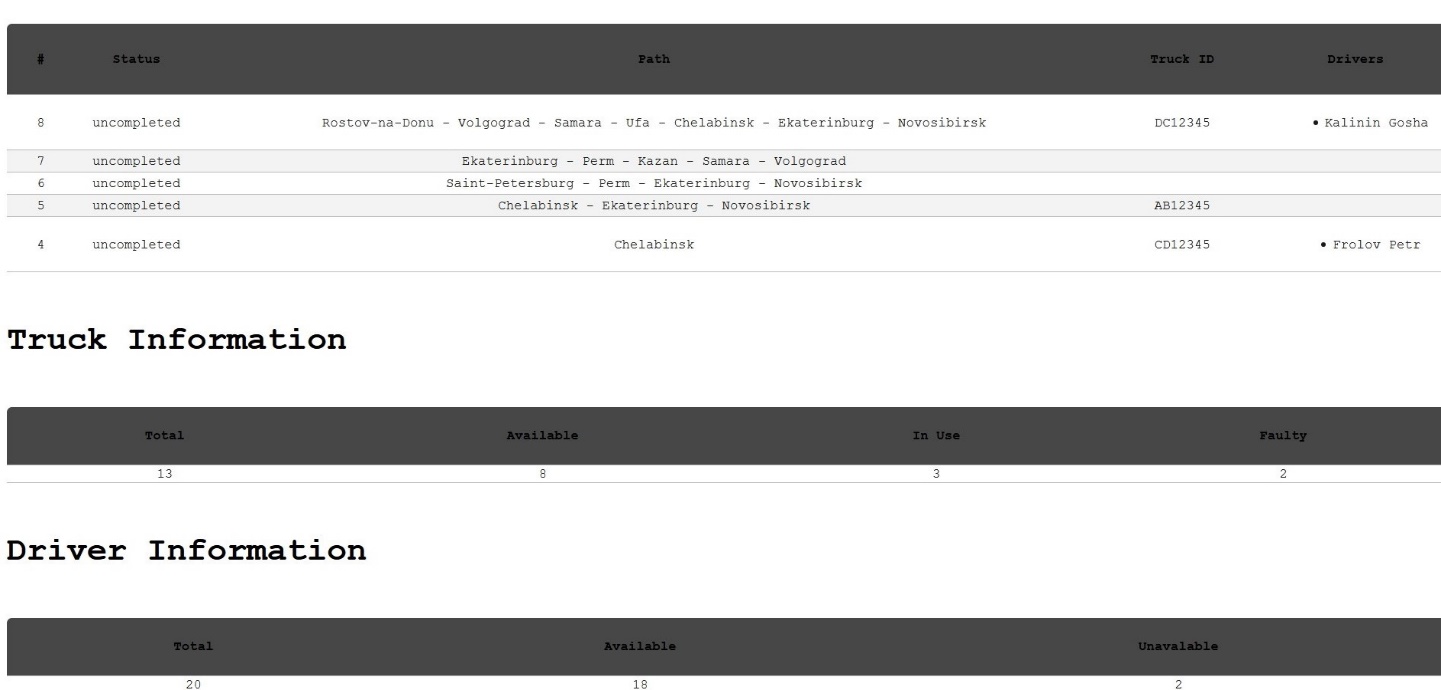
* Admin User (Logistician)
* Active User (Driver)
* Passive User

Admin User have Admin views accessible. Active User have Driver views accessible. Passive User have home page views and Live Monitoring Page access, if that user have remote client application.

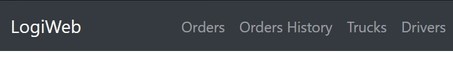
**3.2 Live Monitoring Page**

The live monitoring page is accessible through remote client application.

On this page located information about last 10 uncompleted orders, numbers of drivers and trucks.



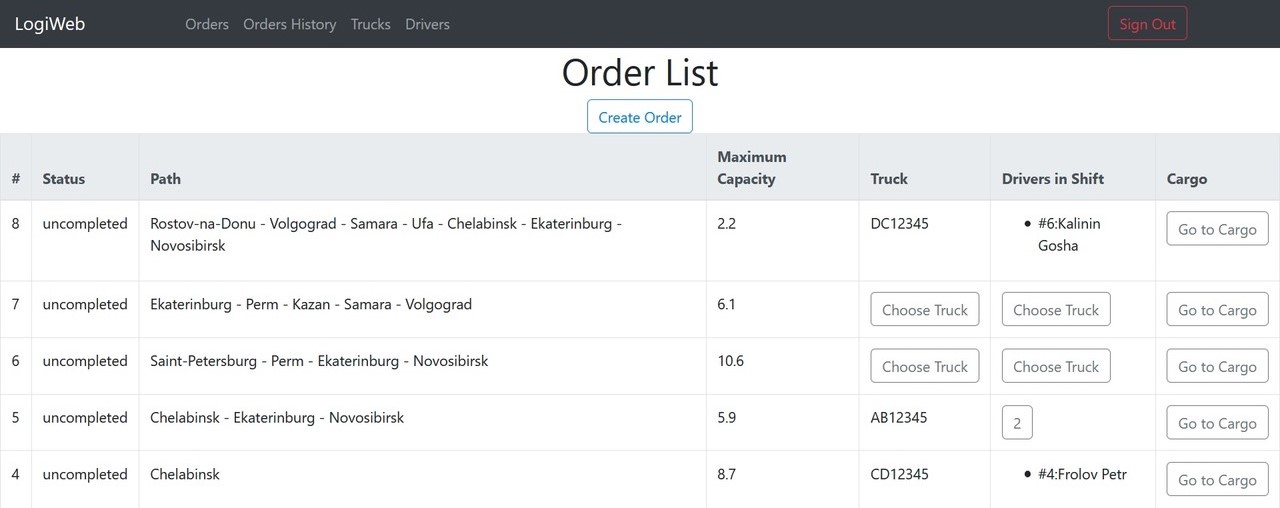
**3.3 Admin views**

Admin have 4 view pages. For crossing over those views on navigation bar located tabs with name of views.

**3.3.1 Orders view**

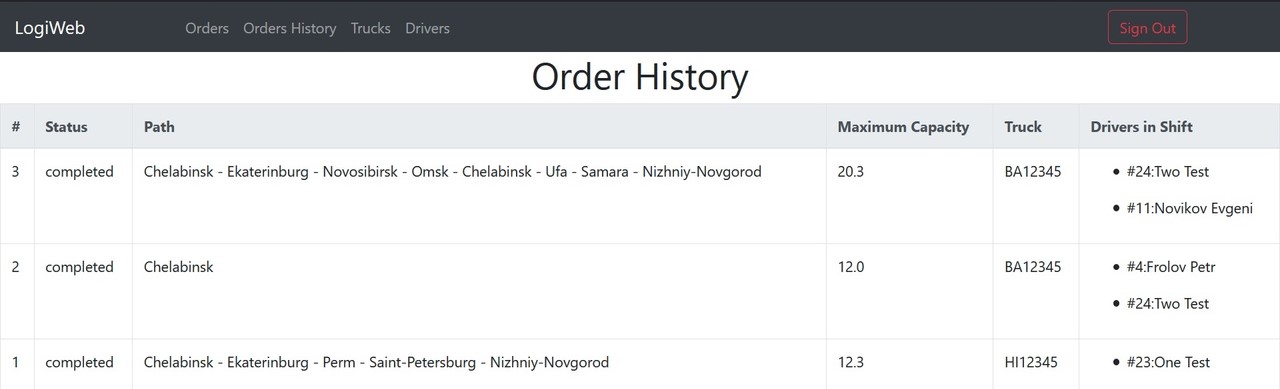
When on the Orders view, the following interactions are possible:

* **Click on button “Create Order”:** display form, where user can choose waypoints and cargo for delivering and create order.
* **Click on button “Choose Truck”:** display table with suitable truck for this order, where user can select truck for order.
* **Click on button at “Drivers in Shift” column:** display table with suitable drivers for this order, where user can select driver(s) for order.
* **Click on “Go to Cargo”:** display table with cargo of current order.



**3.3.2 Orders History view**

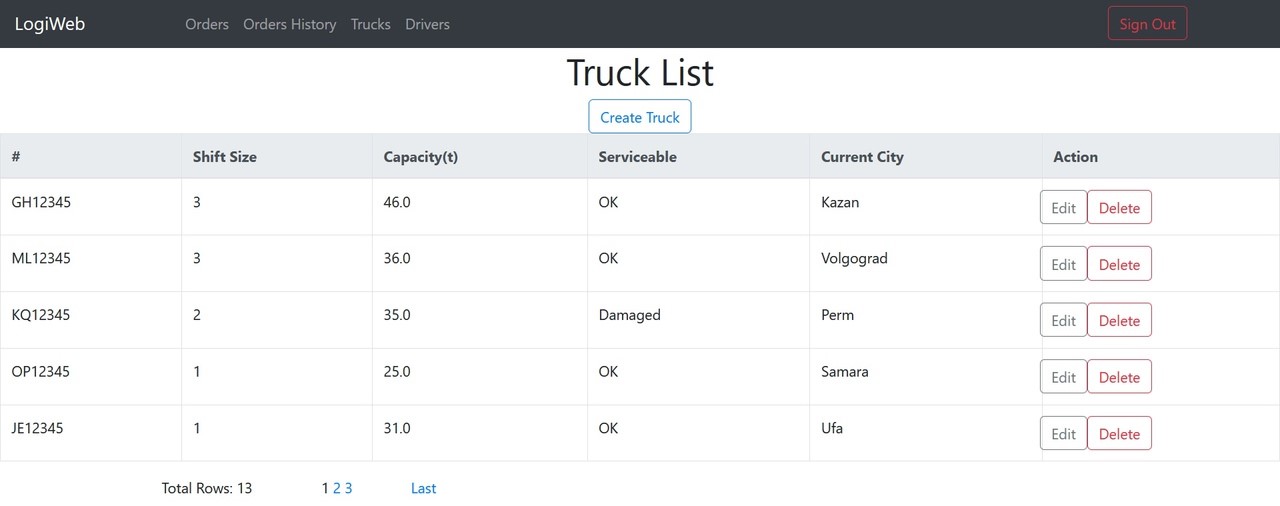
When on this view, the user can look on all completed orders.



**3.3.3 Trucks view**

When on the Trucks view, the following interactions are possible:

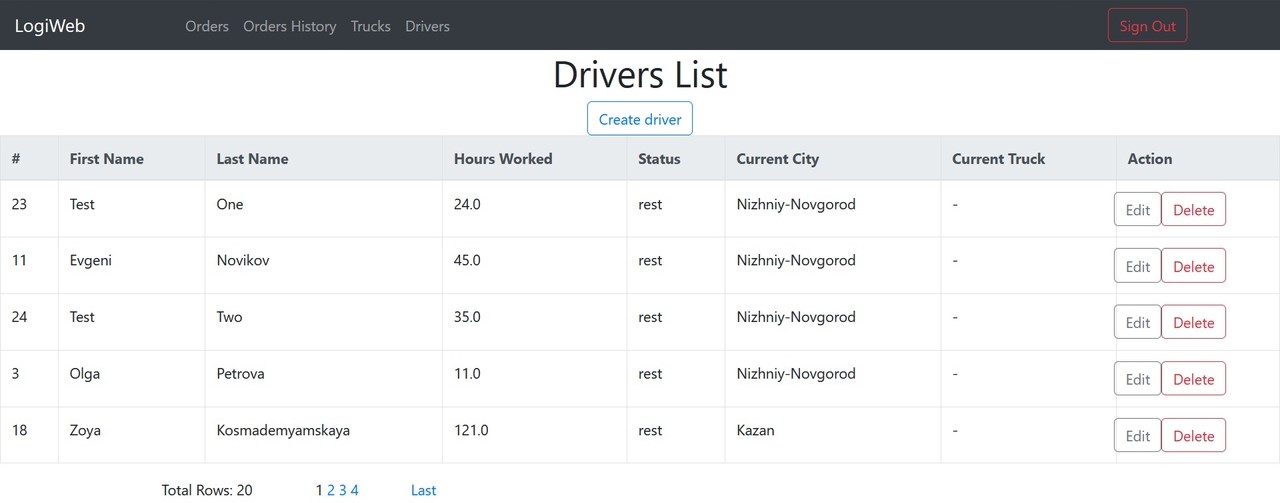
* **Click on button “Create Truck”:** display form, where user can set truck id, capacity, shift size and serviceable status and create new truck.
* **Click on button “Edit”:** display form, where user can edit capacity, shift size and serviceable status.
* **Click on button “Delete”:** delete truck.

****

**3.3.4 Drivers view**

When on the Drivers view, the following interactions are possible:

* **Click on button “Create Driver”:** display form, where user can set email, password, name and surname and create new driver.
* **Click on button “Edit”:** display form, where user can edit name, surname, worked hours.
* **Click on button “Delete”:** delete driver.

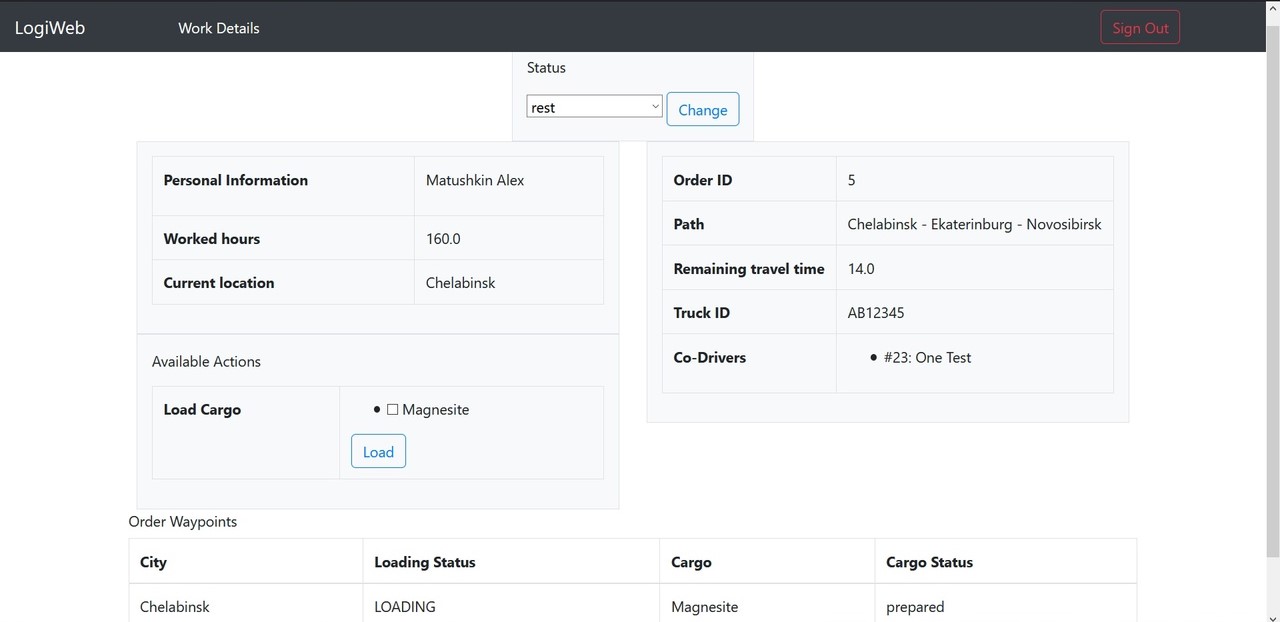
****

**3.4 Driver views**

**3.4.1 Work details view**

When on the Work Details view, the following interactions are possible:

* **Choose status and click on button “Change”:** change driver status.
* **Click on button in “Available Action” panel:** Change city to next, load/unload cargo by select them and click on button “Load/Unload”.

****

On this view driver can look at his personal information (Name and Surname, worked hours and current location), order information (order ID, path, remaining travel time, truck ID and co-drivers) and waypoints table.