**Software Documentation for Web application**

**“TruckGistic”**

Produced By:

Piotr Myszkowski

6500 25th Ave NE

Seattle, WA 98115

Email: [myszon01@gmail.com](mailto:myszon01@gmail.com)



Ministry of Education of the Republic of Belarus

Educational Institution

Belarusian State University of Informatics and Radioelectronics

Computer Systems Institute

\_\_\_\_\_\_\_\_\_\_\_\_\_ Dr.-Ing Vitaly V. Khoroshko Head of the Department of Information and Computer Systems Design BSUIR

**Technical Assignment for Graduation Project**

**Piotr Myszkowski**

1. **Project Name**

Web Application for Freight Transportation Management

Approved by Order No.581-c from 03/26/2018.

1. **Date Terms**

05/01/2018 – 06/19/2018.

1. **Initial data for application development** 
   1. Purpose of the application – freight transportation management.
   2. A brief description of the application functions. The application should provide:
      * users accounts with diverse roles (client, carrier, etc.) and their administration;
      * the database with information about products, time and points of departure and delivery, etc.;
      * a set of tools for ordering, order managing and logistics activities;
      * connecting online maps to display routes;
      * the model for calculating the conditional order price.
   3. Requirements for the programming language – Java.
   4. Requirements for information storage – relational database MySQL.
   5. Requirements for client-side software are not specified.
   6. If necessary, allowed to use JavaScript and CSS code in html-pages.
   7. Other requirements will be specified during project implementation, if needed.
2. **Project Content Requirements**
   1. Title page. Technical Assignment. Content. Introduction.
   2. Subject Area Description.
   3. Hardware Requirements (if needed).
   4. Description of Technologies Used.
   5. Database Design.
   6. Application Structure and UI Design
   7. Algorithms Design
   8. Testing
   9. User Manual (if needed).
   10. Conclusion. References. Attachments.
3. **Graphics Requirements**
   1. ER-Model Database Diagram.
   2. UML Class Diagram.
   3. UML Activity Diagram
   4. UML Deployment Diagram
   5. UML State Machine Diagram.
   6. Screenshots of Application UI.
4. **Development Schedule**

|  |  |  |
| --- | --- | --- |
| Stage Name (requirements part) | Terms | Notes |
| Meeting with graduate supervisor |  |  |
| 1-st Checkpoint |  | 40% |
| 2-nd Checkpoint | 11/15/2018 – 11/16/2018 | 70% |
| 3-rd Checkpoint | 11/29/2018 – 11/30/2018 | 100% |
| Checking of Graduation Project | 12/03/2018 –  12/05/2018 | in agreement with instructor |
| Reviewing of Graduation Project | 12/06/2018 – 12/15/2018 |  |
| Presentation of Graduation Project | 12/17/2018 – 12/22/2018 | May be December 18, 2018 |

Instructors of Graduation Project \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Andrei V. Pyatosin

Assistant of Department of Information

and Computer Systems Design BSUIR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pavel Protasevich

Instructor of Computer Systems Institute

Student \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Piotr Myszkowski

Contents

Introduction 5

1 Subject Area Description 6

1.1 Freight Transportation 6

1.2 Fleet management 6

1.3 CRM – Customer relationship management 7

2 Description of Technologies Used 7

2.1 Back-end 7

2.2 Front-end 9

2.3 Host 11

3 Database Design 11

3.1 User Table 12

3.2 Customer Table 12

3.3 Facilities Table 13

3.4 Loads table 14

3.5 Pick\_drops Table 14

4 Application Structure and UI Design 15

4.1 Application structure 15

4.2 Server side - backend 16

4.3 Client side - frontend 18

4.4 User Interface Design 21

5 Testing 29

5.1 Functional Testing 31

6 Economics 33

Conclusion 38

Conclusion 39

References 40

Appendix A 42

# Introduction

The purpose of this document is to describe the implementation of the TruckGistic web application. The TruckGistic web application software is designed to help freight management businesses focused in USA territory to track theirs loads, drivers and customers.

This document describes the implementation details of the TruckGistic Web Application. The software will consist of multiple functions. That include create/delete/update loads, customers, drivers, facilities. We will also outline and describe specific components of the project.

When designing this software it was critical to know and deeply understand logistics problems that freight companies have to face every day. This application was consultant with group of people that work in the industry to help understand the best design and solution for the problems.

This application is design for multiple companies/users. Because of the design and lightweight framework that were used it can handle thousands of users making requests with no downtime. Thanks to newest technologies and design techniques used in this project it will me much easier to maintain, troubleshoot, add new features and scale this web application.

All technologies, tools that were used are free besides server that will host database and/or web application. That is really important point since I want this application to be free for all user and deliver to them the best quality of the freight managing web application available on the market for free.

# 1 Subject Area Description

## 1.1 Freight Transportation

**S**imply defined, freight transportation is the movement of goods from one area to another. Freight Transpiration allows production and consumption to occur at different locations. Transportation is necessary for economic specialization. Freight transportation allows companies to specialize in producing the products for which they are best suited and trade with other companies to obtain products that can be made more efficiently by others.

Our nation’s economic strength and competitiveness depends on a safe, efficient, sustainable and secure freight transportation system. Whether it is moving coal from the Powder River Basin in Wyoming to Atlanta, GA or tennis shoes from China to Rochester, New York, the system must provide for the reliable, flexible and economic movement of goods – bulk and consumer – from a diverse array of sources.

Customer relationship management (CRM) is a model for managing a company’s interactions with current and future customers. It involves using technology to organize, automate, and synchronize sales, marketing, customer service, and technical support.

Although for many enterprises it takes time for it to sink in, the result is inevitable. Marketing led the way with the concept of tailoring products and services to the requirements of specific groups of customers. It named its approach ‘segmentation’ and asked customers what they wanted, so that the company could produce different offerings that matched varying needs and wants.

## 1.2 Fleet management

Fleet management is the function that oversees, coordinates and facilitates various transport and transport related activities. For the purpose of this document it will cover vehicles involved in the movement of goods; the management of light vehicle fleets used in the transportation of people and light cargo; possibly motorbikes and other equipment such as generators and warehouse handling equipment. Fleet management underpins and supports transport related activities through the management of the assets that are used.

Effective fleet management aims at reducing and minimizing overall costs through maximum, cost effective utilization of resources such as vehicles, fuel, spare parts, etc.

The administration and financial management of fleet is very organizational specific. It largely depends on donor requirements and organizational policies. For example, in some organizations vehicles are restricted to specific projects and others utilize vehicle pools to serve multiple projects. Driving policies can vary from a strict reliance on a dedicated driver from the organization, to using staff to drive the vehicles. The administrative policies of the individual organization will dictate which approach will be utilized. This results in the custodian of the fleet management function to be very much dependent on organizational policies and structures.

## 1.3 CRM – Customer relationship management

CRM is a technology for managing all your company’s relationships and interactions with customers and potential customers. The goal is simple: Improve business relationships. A CRM system helps companies stay connected to customers, streamline processes, and improve profitability.

When people talk about CRM, they are usually referring to a CRM system, a tool that helps with contact management, sales management, productivity, and more.

A CRM solution helps you focus on your organization’s relationships with individual people — including customers, service users, colleagues, or suppliers — throughout your lifecycle with them, including finding new customers, winning their business, and providing support and additional services throughout the relationship.

A CRM system gives everyone — from sales, customer service, business development, recruiting, marketing, or any other line of business — a better way to manage the external interactions and relationships that drive success. A CRM tool lets you store customer and prospect contact information, identify sales opportunities, record service issues, and manage marketing campaigns, all in one central location — and make information about every customer interaction available to anyone at your company who might need it.

A CRM system can give you a clear overview of your customers. You can see everything in one place — a simple, customizable dashboard that can tell you a customer’s previous history with you, the status of their orders, any outstanding customer service issues, and more.

# 2 Description of Technologies Used

## 2.1 Back-end

For the core language I choose **Java** programing language. The reason Java is core language for this web app is that Java has his own JVM and same code can be run on different machines. Another reason for Java is that it has very powerful Spring framework that helps build scalable, structure enterprise projects.

Every good application needs to be scalable and have lightweight technologies that work great with each other. That was the reason I choose **Spring Boot** **framework** as a backend. This framework of Java is great if it comes to large, enterprise applications and have many API, Documentation, support that helps implement different technologies to it. Spring Boot has no XML based configurations at all. Very much simplified properties. The beans are initialized, configured and wired automatically. Spring is also lightweight container, can deployed on tomcat or jetty, easy to implement new features if your application getting bigger and bigger.

For database connectivity I choose **Hibernate** **ORM** over JDBC because JDBC is complex when is used in large projects, it is harder to implement MVC concept, there is no encapsulation. Hibernate on other side has transparent persistence that ensures automatic connection between the application’s object with the database tables. This feature prevents developers from writing lines of connection code. Transparent persistence enables hibernate to reduce the development time and maintenance cost. Hibernate supports both first level and second level caching mechanism. The first level caching is associated with Session object which is used by default. The second level caching is associated with Session Factory Object. Through caching concept, Hibernate retains the objects in cache so as to reduce repeated hits to the database. This feature makes Hibernate highly scalable and optimizes the application’s performance. Hibernate supports optimistic locking through its version property feature. This functionality supports multiple transactions without affecting one another. For example, when two or more users try to alter a database entity at the same time, the version field avoids the conflict and gives preference to the user who commits the changes first. The other user will be prompted with an error message and will be asked to restart the process.

I integrate Hibernate with **Spring Data** so I will have repositories CRUD out of box, nice and easy ways to extend your repositories with query methods, to navigate between entities.

For authentication and authorization, I used **Spring Security.** It is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

As a database I choose **MySQL** because is globally renowned for being the most secure and reliable database management system used in popular web applications. MySQL offers unmatched scalability to facilitate the management of deeply embedded apps using a smaller footprint even in massive warehouses that stack terabytes of data. On-demand flexibility is the star feature of MySQL. This open source solution allows complete customization to eCommerce businesses with unique database server requirements. MySQL features a distinct storage-engine framework that facilitates system administrators to configure the MySQL database server for a flawless performance. Whether it is an eCommerce website that receives a million queries every single day or a high-speed transactional processing system, MySQL is designed to meet even the most demanding applications while ensuring optimum speed, full-text indexes and unique memory caches for enhanced performance. All the fears and worries that arise in an open source solution can be brought to an end with My SQL’s round-the-clock support and enterprise indemnification. The secure processing and trusted software of MySQL combine to provide effective transactions for large volume projects. It makes maintenance, debugging and upgrades fast and easy while enhancing the end-user experience. MySQL is considered a very fast database program. This speed has been backed up by a large number of benchmark tests (though such tests -- regardless of the source -- should be considered with a good dose of skepticism). MySQL is really easy to use with hibernate. There is nothing too complicated to have those two technologies configure and up and running in less than 1 hour. MySQL also provides with very powerful tool – MySQL Workbench – that makes your development of database super easy and enjoyable. On top of that this tool has many other useful tools like data migration, database backup, UML creator, schema generator etc.

**Webpack** is an open-source JavaScript module bundler. Webpack takes modules with dependencies and generates static assets representing those modules. It takes the dependencies and generates a dependency graph allowing web developers to use a modular approach for their web application development purposes.

**Node.js** is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code server-side. Historically, JavaScript was used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser. Node.js lets developers use JavaScript for server-side scripting – running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server side and client side scripts

## 2.2 Front-end

**HTML** is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page.

**JavaScript** is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else. (Okay, not everything, but it is amazing what you can achieve with a few lines of JavaScript code.)

For front end part I have choose **ReactJS**. It is pretty new JavaScript framework developed and released by Facebook. With React, it is possible to develop large web-based applications that change data while reloading pages. You can also use React together with other forms of frameworks or JavaScript libraries like Angular in MVC. React is an open source JavaScript (JS) library used for developing user interfaces, basically for the single-page applications. When dealing with ReactJS, the basic programming knowledge that you need to possess is HTML and CSS. Angular, on the other hand, is more difficult to learn. If you have not familiarized yourself with typescript, then do so before learning Angular. Typescript on its own is not easy to learn, so be prepared for a long learning curve. One major advantage Facebook brought with ReactJS is the possibility of reusing code components. Webmasters can now use codes of different levels anytime, and this saves time. Usually, system upgrades are a headache to webmasters, as other components can be affected by any changes made. with ReactJS, it is easier to manage updates. The reason is that all the components of the system are completely isolated. As a result of this, changes initiated in any of the components will not affect the others. It makes it possible to reuse components that didn’t produce changes, and this makes programming more comfortable and precise. Another advantage with ReactJS is code stability. It allows programmers to work directly with the components and also make use of the [data-binding technique](https://www.techopedia.com/definition/15652/data-binding) to make sure changes that what took place in child structures doesn’t affect the parents.

**CSS** is used for presentation layer. CSS stands for Cascading Style sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media.CSS saves a lot of work. It can control the layout of multiple web pages all at once.

**Bootstrap** framework is responsible for presentation layer. Bootstrap is responsive, mobile-first, prevailing, and front-end framework, which is developed along with CSS, JavaScript, and HTML. It is extremely an easy and speedy procedure to begin with Bootstrap. Bootstrap is very adaptable too.One of the main benefits of utilizing Bootstrap happens to be the speed of the development. While driving out a new, fresh website or application swiftly, you should certainly reflect upon utilizing Bootstrap. Instead of coding from scrape, Bootstrap lets you to use ready-made coding blocks in order to assist you in setting up. You can blend that along with CSS-Less functionality and cross-browser compatibility that can give way to saving of ample hours of coding. You can even buy ready-made Bootstrap themes and alter them to fit your requirements, for gaining the quickest potential route. The Bootstrap can be customized as per the designs of your project. Bootstrap approaches with a pack of JavaScript components for including the functionality that crafts it in simple way for operating things, such as tooltips, modal windows, alerts, etc. You can even leave out the writing scripts completely.

## 2.3 Host

This application is hosted online on **Apache Tomcat**. Apache Tomcat is an open source Web server tool developed by the Apache Software Foundation (ASF). It is one of many Apache-related open source products used by IT professionals for various tasks and objectives. Apache tomcat is an incredibly lightweight. If offers only the most basic functionality necessary to run a server, meaning it provides relatively quick load and redeploy times compared to many of its peers, which are bogged down with far too many bells and whistles. This lightweight nature also allows it to enjoy a significantly faster development cycle. It is open-source always counts as a win. Tomcat’s free, and the source code for the server is readily available to anyone who’d care to download it. Tomcat is an extremely stable platform to build on – and using it to run your applications will contribute to your server’s stability, as well. This is because Tomcat runs independently of your Apache installation – even if a significant failure in Tomcat caused it to stop working, the rest of your server would run just fine.

Apache Tomcat web server is hosted on **Centos**. Centos is a [Linux distribution](https://en.wikipedia.org/wiki/Linux_distribution) that provides a free, [enterprise-class](https://en.wikipedia.org/wiki/Enterprise_software), community-supported computing platform functionally compatible with its upstream source, Red Hat Enterprise Linux (RHEL). When CentOS 7 is properly configured, and running on quality hardware, it is a very stable server operating system, with very few (if any) problems. There is reduced risk of crashes and errors, as it runs only stable versions of packaged software.

For PaaS (**P**latfrom **a**s **a** **S**ervice) I used **DigitalOcean.** It is an [American](https://en.wikipedia.org/wiki/Americas) [cloud](https://en.wikipedia.org/wiki/Cloud_computing) infrastructure provider headquartered in [New York City](https://en.wikipedia.org/wiki/New_York_City) with data centers worldwide. DigitalOcean provides developers cloud services that help to deploy and scale applications that run simultaneously on multiple computers. As of January 2018, DigitalOcean was the third-largest [hosting](https://en.wikipedia.org/wiki/Web_hosting_service) company in the world in terms of web-facing computers.

# 3 Database Design

The first step in the database design was to analyze the data that would be collected and determine the expected uses of the data.

Customer place an order through freight company representative. Then representative have to enter load information to the system, assign it to the driver and to customer. In this database we have total of 5 tables. They are:

## 3.1 User Table

User table hold information about users that logging in to the application. They are drivers and dispatchers (representative that agree with customer on rate and load). This is the most important table since it connects to rest 4 tables.

Table 3.1

|  |  |  |
| --- | --- | --- |
| **Users** | | |
| **Column Name** | **Datatype** | **Description** |
| user\_ID | INT(11) | ID number |
| username | VARCHAR(45) | Username |
| password | CHAR(60) | Password |
| role | VARCHAR(45) | Role of the user |
| email | VARCHAR(255) | E-Mail address |
| confirmation\_token | VARCHAR(255) | Confirmation token for new password |
| enabled | BIT(1) | Enabled field says if user is active  in the system |
| organization | VARCHAR(255) | Organization name |
| first\_name | VARCHAR(45) | First Name |
| last\_name | VARCHAR(45) | Last Name |
| address | VARCHAR(45) | Address |
| city | VARCHAR(45) | City |
| state | VARCHAR(2) | State |
| phone\_number | VARCHAR(45) | Phone Number |
| zipcode | VARCHAR(45) | Zip code |
| version | VARCHAR(45) | Version NR. of this user it is used for locking mechanism |

## 3.2 Customer Table

Customers table is responsible for keeping information about customer.

Table 3.2

|  |  |  |
| --- | --- | --- |
| **Customers** | | |
| **Column Name** | **Datatype** | **Description** |
| customer\_ID | INT(11) | ID number |
| name | VARCHAR(45) | name |
| mc\_number | INT(11) | MC number |
| dot\_number | VARCHAR(45) | DOT number |
| address | VARCHAR(45) | Address |
| city | VARCHAR(45) | City |
| state | VARCHAR(45) | State |
| zip\_code | VARCHAR(45) | Zip Code |
| phone\_number | VARCHAR(45) | Phone Number |
| email | VARCHAR(45) | E-Mail |
| fax | VARCHAR(45) | fax number |
| description | VARCHAR(45) | Description |
| version | INT(11) | Version NR. of this customer it is used  for locking mechanism |
| users\_user\_ID | VARCHAR(45) | User ID to which this customer belongs |

## 3.3 Facilities Table

Facilities table contain information about facility that load is picking up from.

Table 3.3

|  |  |  |
| --- | --- | --- |
| **Facilities** | | |
| **Column Name** | **Datatype** | **Description** |
| facility\_ID | INT(11) | ID number |
| name | VARCHAR(45) | Name |
| address | VARCHAR(45) | Address |
| city | VARCHAR(45) | City |
| zip\_code | VARCHAR(10) | State |
| state | VARCHAR(2) | Zip Code |
| phone\_number | VARCHAR(45) | Phone Number |
| email | VARCHAR(45) | E-Mail |
| version | VARCHAR(45) | Version NR. of the facility it is used for locking mechanism |
| users\_user\_ID | INT(11) | User ID to which this facility belongs |

## 3.4 Loads table

Loads table contain information about driver that is assign for this load, rate, extra cost, instruction

Table 3.4

|  |  |  |
| --- | --- | --- |
| **Loads** | | |
| **Column Name** | **Datatype** | **Description** |
| load\_ID | INT(11) | ID number |
| rate | VARCHAR(45) | Name |
| extra\_cost | VARCHAR(45) | Address |
| instruction | VARCHAR(45) | City |
| version | VARCHAR(45) | Version NR. of the facility it is used for locking mechanism |
| users\_user\_ID | INT(11) | User ID to which this load belongs |
| driver\_ID | VARCHAR(45) | User ID with role driver to which this load is assign to |
| customers\_customer\_ID | VARCHAR(45) | customer id to which this load belongs |

## 3.5 Pick\_drops Table

Pick\_drops table connect load table to facilities table and contain extra information about each pickup or drop like time, date, type (pick or drop), and instruction for this specific pickup.

Table 3.5 –

|  |  |  |
| --- | --- | --- |
| **picks\_drops** | | |
| **Column Name** | **Datatype** | **Description** |
| picks\_drops\_ID | INT(11) | ID number |
| facility\_ID | INT(11) | facility ID number to which this pickup/drop is assaign to |
| load\_ID | INT(11) | Load ID numbet to which this pickup/drop belongs |
| work\_type | VARCHAR(45) | City |
| time | TIME | time of pickup/drop |
| date | DATE | date of pickup/drop |
| instruction | VARCHAR(45) | instruction |
| version | VARCHAR(45) | Version NR. of the pickup/drop it is used for locking mechanism |
| users\_user\_ID | INT(11) | User ID to which this pickup/drop belongs |

Also, all of those table (besides table user) contain reference to table user because we have multiples users. When user creates account, he creates root account for his company. When user is creating driver it automatically creates new user in database. There are two roles of user available. One is role ROLE\_MANAGER which is responsible for creating/deleting/updating customers, loads, facilities, pick\_drops, users tables (for users table ROLE\_MANAGER can only manipulate users with role ROLE\_DRIVER.

Second role is ROLE\_DRIVER and he can only view data in customers, loads, facilities, pick\_drops tables assign only for his manager account.

# 4 Application Structure and UI Design

## 4.1 Application structure

The model view controller (MVC) divides the structural components into three parts; these parts are as follows:

1. **M**odel: Houses application logic
2. **V**iew: User interface
3. **C**ontroller: The glue or binding entity between the view and the model

The model is the core of the application. The business logic of the application, or, in simpler terms, the capabilities of the application on a whole can be termed as the model component of an application. The logic within the model layer might deal with storing and managing data to maintain the state of the application. All the network communications that your application makes, the parsing and complex calculations that it performs on the received data, all comes under the model layer.

The view is the interface to the application. State of the application is represented by the view. In order to enhance decoupling in the application architecture, views should not have any logical component in their implementation other than the ones required to take inputs or display data on a screen.

The controller manages communication between the view and the model. It also encapsulates the functionality that is specific just to governing the interaction between various logical components, as a result of which model and view subsystems remain decoupled. The controller can be termed as the binding element between the model and views.

The software virtue that MVC exhibits is modularity, which helps developers to manage the complexity of developing an application. It allows the model, view, and controller modules to focus on a limited range of functionality. In short, it allows them to specialize.

The TuckGistic application is broken up into three parts: client-side application, server side application and MySQL database (database is described in previous chapter).

## 4.2 Server side - backend

The **server-side** is based on Java framework Spring. The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. It is separated into 6 different packages:

*– com.freight.crm* – contain three classes that are responsible for configuration of spring framework, WebSocket and main Spring class responsible for running application;

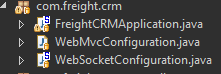


Figure 4.1

*– com.freight.crm.controller* – contain controllers responsible for mapping and process custom request from client-side application;

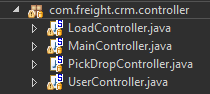


Figure 4.2

*– com.freight.crm.entity* – contains classes that represents tables and relationship in MySQL database and contain setters and getters methods for each class;

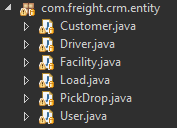


Figure 4.3

*– com.freight.crm.event.handlers.ws* – this package contains classes that are responsible for handling before/after create delete update operation on entity classes. Also it is responsible to send message to connected users in WebSocket endpoint to refresh their content in web;

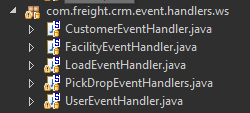


Figure 4.4

*– com.freight.crm.security* – is responsible for Spring security context. Here are all configuration to what quest has access and registered users. It also contain class to search users by username;



Figure 4.5

*– com.freight.crm.service* – this last package is responsible business logic. Most of the interfaces extands PagingAndSortingRepository interface which enables simple REST endpoint for clinet. It also contain class that is responsible for sending confirmation emails to new users.

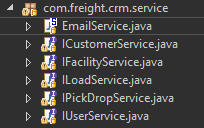


Figure 4.6

As we can see that it is really simple and straight forward application structure where intuitively you can tell what role is responsible what package, class or interface.

## 4.3 Client side - frontend

The **client side** is based on JavaScript framework ReactJS. React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. We use webpack to compile JavaScript components into a single, loadable bundle. The whole client side is hosted on node.js server. When user access application first time it first check with back-end engine to check for authentication and Spring returns view for gest with help of Thymeleaf framework. When user logging in and pass authentication part he is then logged in to ReactJS application.

Per above front end side is divided in two parts. One before user is logged in and all request are handled by Spring framework and second when user is already logged in and Spring will return page where ReactJS has his start point. There are 6 html pages that are return by Spring framework. It is:

*– confirm.html* – this page is displayed when user click on link with his confirmation token to confirm that his email is correct and to enter his new password;

*– index.html* – this page is displayed when user is already logged in and return start point for ReactJS application and then ReactJS makes REST call to back end engine Spring;

*– login.html* - this page is displayed when user is trying to logging in to the application;

*– register.html* – this page is displayed whenever new user wants to register;

*– restPassword.html* – this page is displayed whenever user wants to reset his password;

*– welcome.html* – this page is displayed for quest users when they first time enter the website.

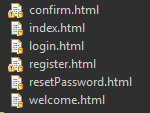


Figure 4.7

Each of this page contain two fragments:

* *footer.html* – contains footer part for web pages
* *header.html* – contains header part for web pages



Figure 4.8

The second part of client side is ReactJS. This part is available only for authorized users after they pass authentication level. We use webpack to define entry point, sourcemaps, compile all JavaScripts bits and it hooks the babel engine using both es2016 and react presets in order to compile ES6 React code into format able to run in any standard browser. All this configuration are located in webpack.config.js file. ReactJS app contain:

1. *api* folder – it contains .js file responsible supports URI Templates.



Figure 4.9

1. *pages* Folder – contains .js file for each page. This folder have .js file that generates web pages for facilities, customers, loads, users page. This folder has following structure:
   1. *components* Folder – contain global components that are used in forms for page in pages Folder.
   2. *CustomerComponents* Folder – contain .js files for create, updated and view customers
   3. *FacilityComponents* Folder - contain .js files for create, updated and view facilities
   4. *LoadComponents* Folder - contain .js files for create, updated and view loads
   5. *UserComponents* Folder - contain .js files for create, updated and view drivers assigned for Manager user
   6. *Customers.js* – is main .js file for customer page that is responsible to make REST call to spring, render view using components from *CustomerComponents* Folder by passing data to those components
   7. *Facility.js* - is main .js file for facility page that is responsible to make REST call to spring, render view using components from FacilityComponents folder by passing data to those components
   8. *Loads.js* - is main .js file for loads page that is responsible to make REST call to spring, render view using components from LoadComponents folder by passing data to those components
   9. *Users.js* - is main .js file for users page that is responsible to make REST call to spring, render view using components from UserComponents folder by passing data to those components

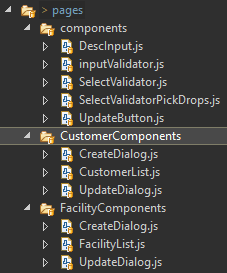
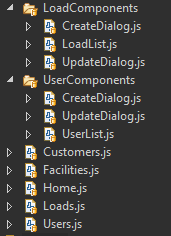
 

Figure 4.9

1. *app.js* – is entry point and it is responsible to route user to whichever component he is trying to access
2. *client.js* - is custom code that configures rest.js to include support for HAL, URI Templates, and other things. It also sets the default Accept request header to application/hal+json.
3. *follow.js* – contain follow function that is responsible to navigate between relationships return by JSON object from the root level
4. *websocket-listener.js* - is responsible for registering user to websocket endpoint in Spring framework server.

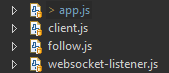


Figure 4.10

React is designed around the concept of reusable components. You define small components and you put them together to form bigger components. All components small or big are reusable, even across different projects.

## 4.4 User Interface Design

User Interface Design is simple and straight forward. When you enter the web application from you PC or mobile device you will see friendly home page.

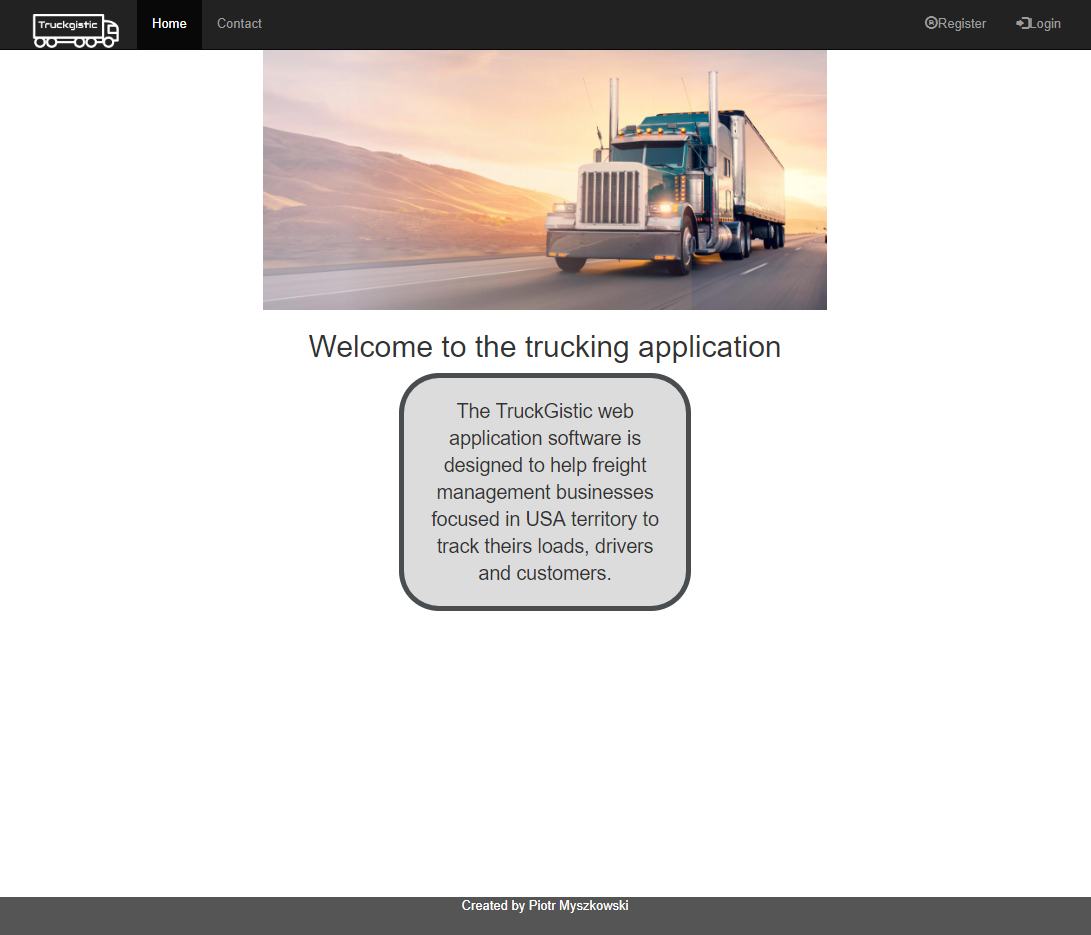


Figure 4.11 – Home page

If the user is a new user then he first have to go through the register process by clicking register button located on top right corner.

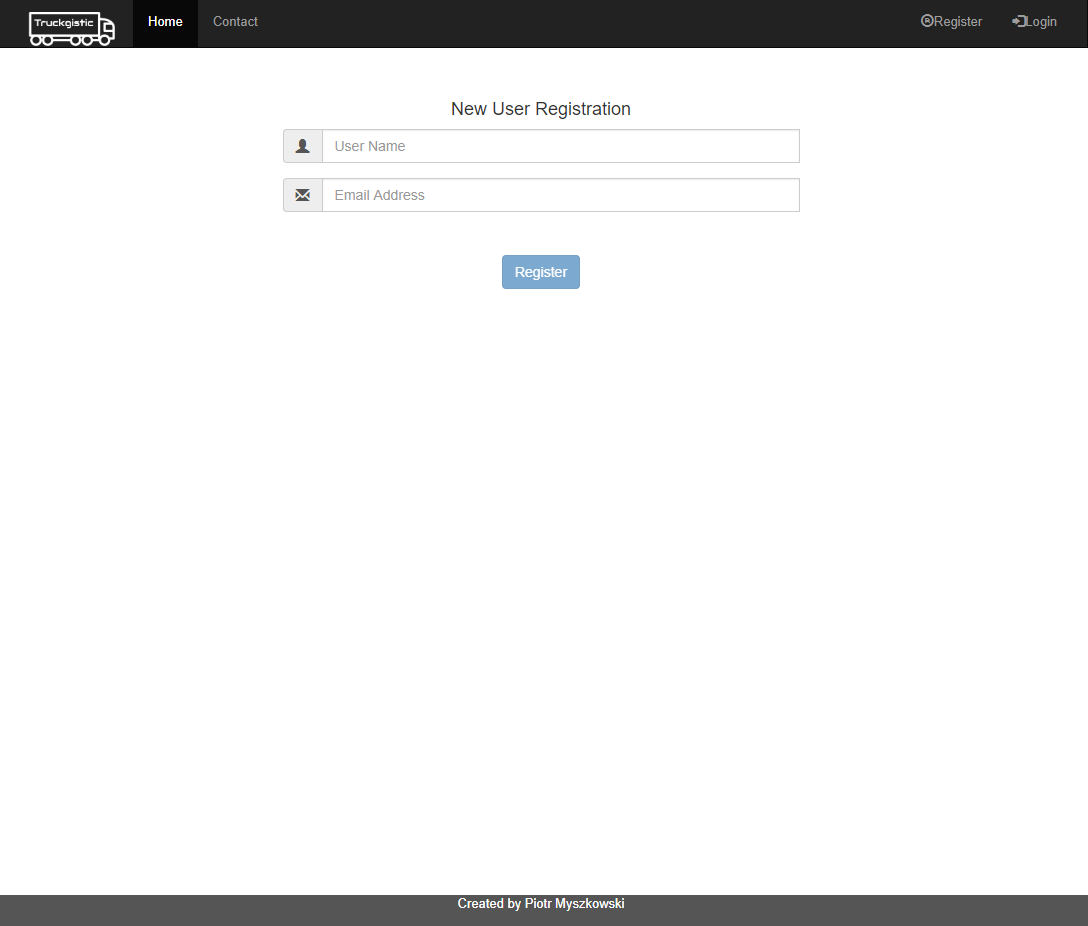


Figure 4.12 – Register page

User has to provide unique email as well as username. Then the system sends the email that was provided by user with the link he has to go in order to create his password. This method make sure that the email is correct and user has access to this email.

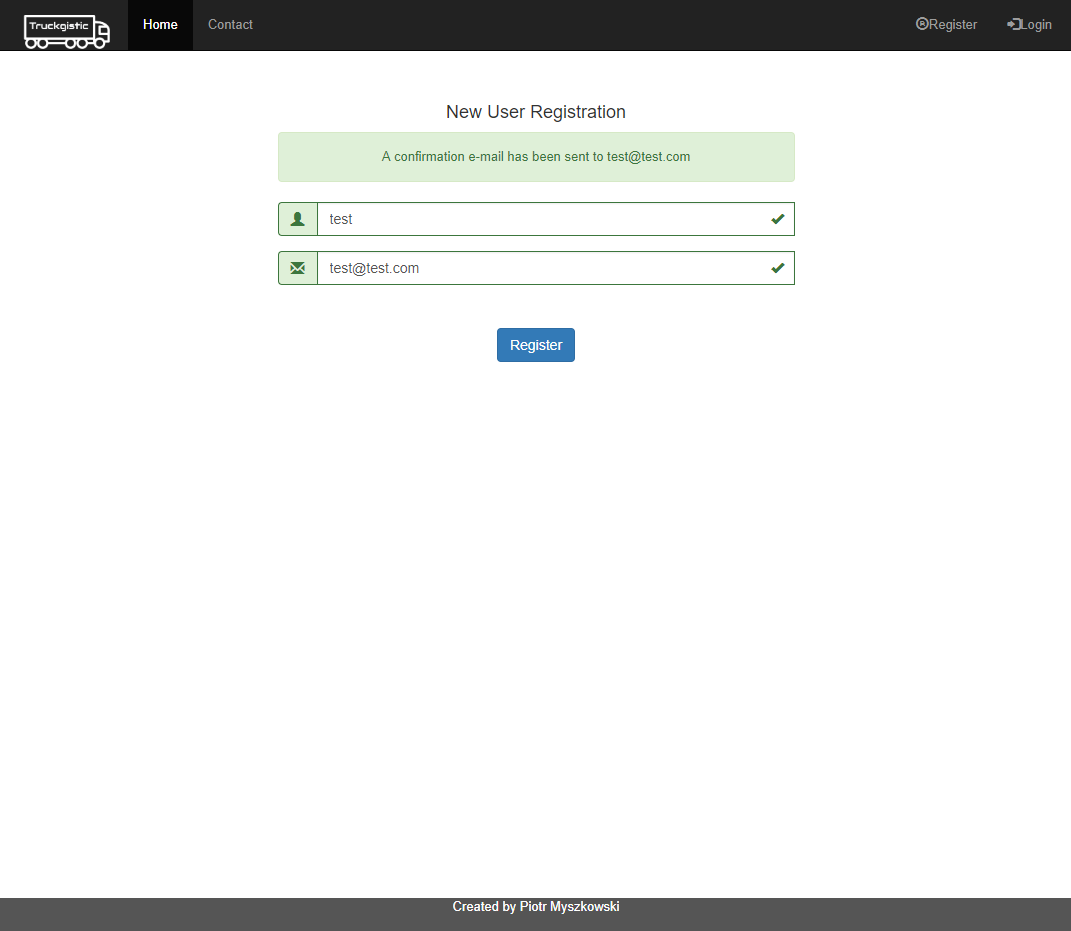


Figure 4.13 – Confirmation page

When user click on the link provided in the email send by the web application, he will be redirected to the website to enter his password. Password has to have one upper letter, special character, number and minimum of 8 characters.

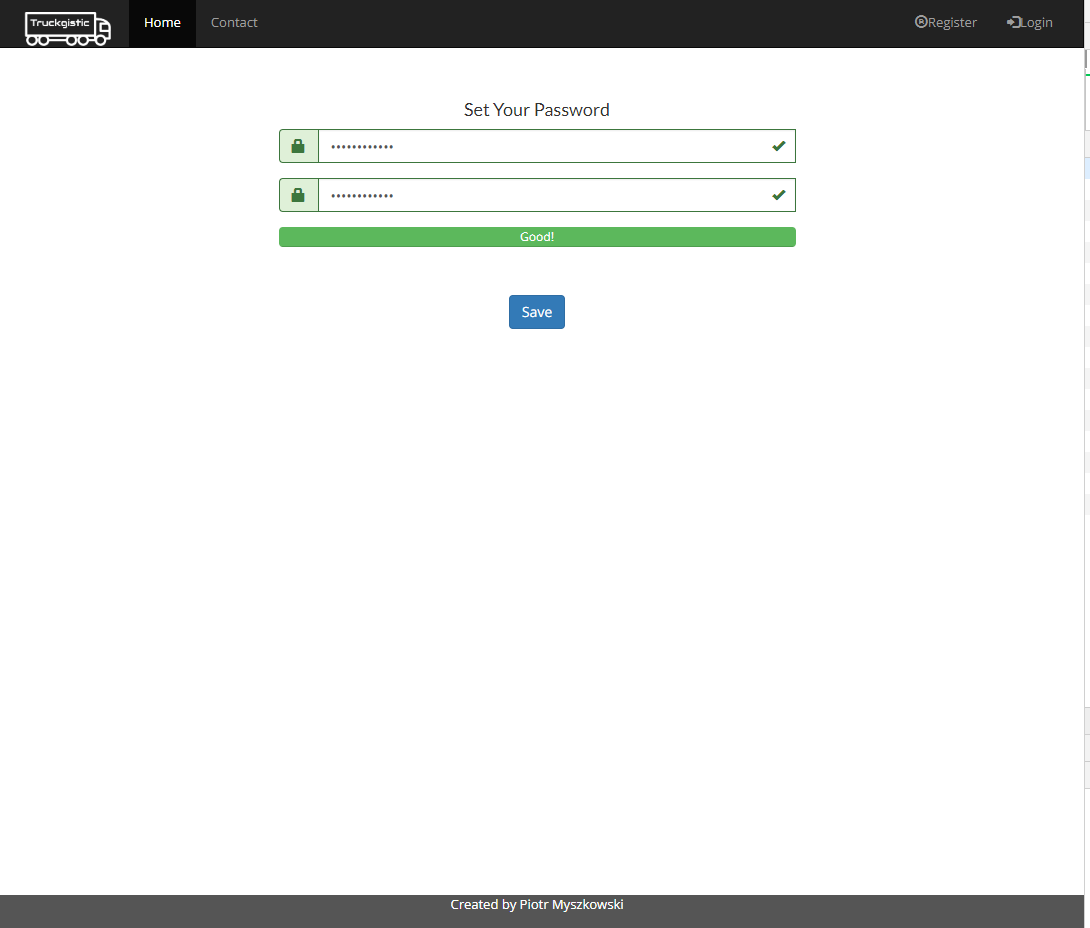


Figure 4.13 – Set password page

After registration process is completed user can finally login to the website.

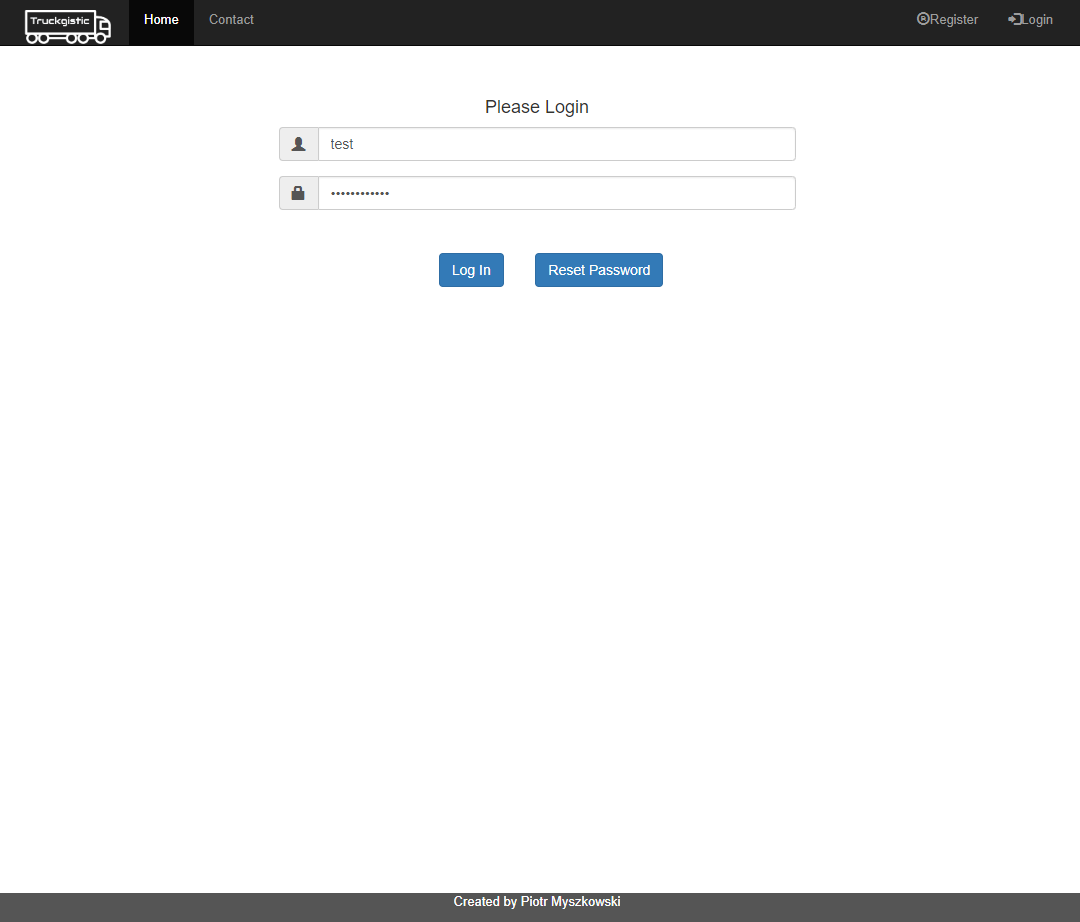


Figure 4.14 – Login page

There are two types of user’s, manager and driver. Managers can add/remove/edit loads, customers, facilities, drivers. When manager creates new driver our web application creates new account with different permissions. After successful login user will see similar page to the one below. Since we are having manager type account we can see tabs for loads, customers, facilities, drivers.

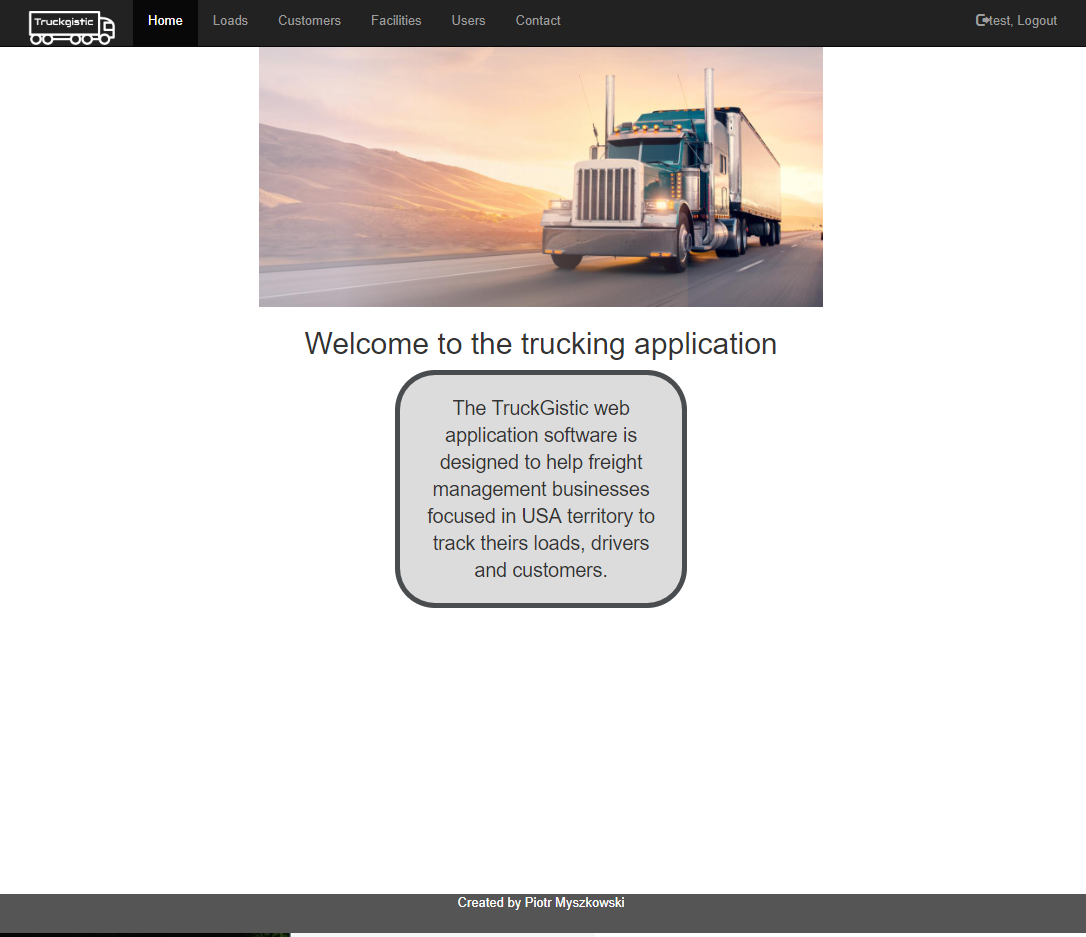


Figure 4.15 – Home page after login

On the Loads tab user can add/remove/edit/search or export to csv existing loads. Whenever there is new load or update on the load from different user we would see this update right away because we are subscribed to the websocket broker of the server. Whenever there is an update server sends, to the all clients subscribed to it, a message to update the content.

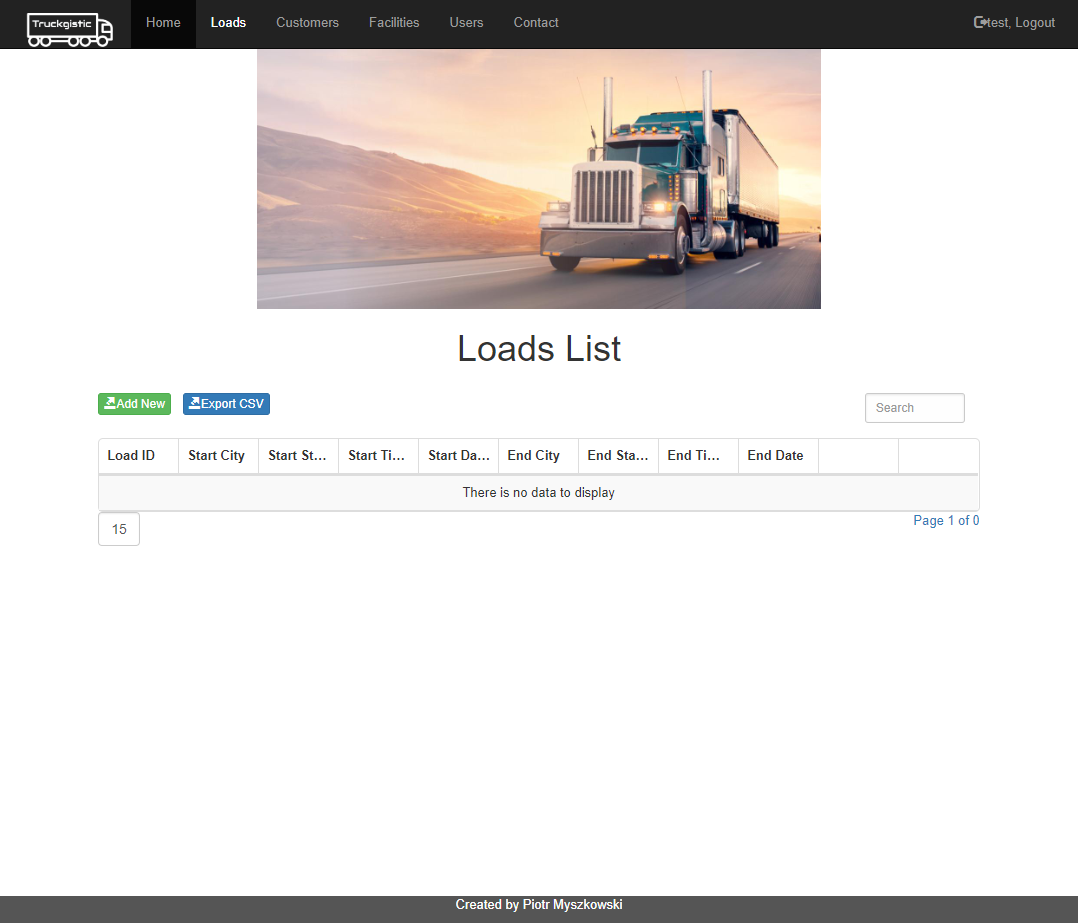


Figure 4.16 – Load list page

To create a new load, we simply click on Add New and a new pop up window appear. From Customer dropdown we select a customer that we are transporting this freight for. Application can calculate the cost of the service by giving it cost per mile and miles (if we don’t specify the miles calculator will evaluate the distance between two or more points by itself). Next, we have Driver that this load will be assign to. Another input is the global instructions/notes for this load.

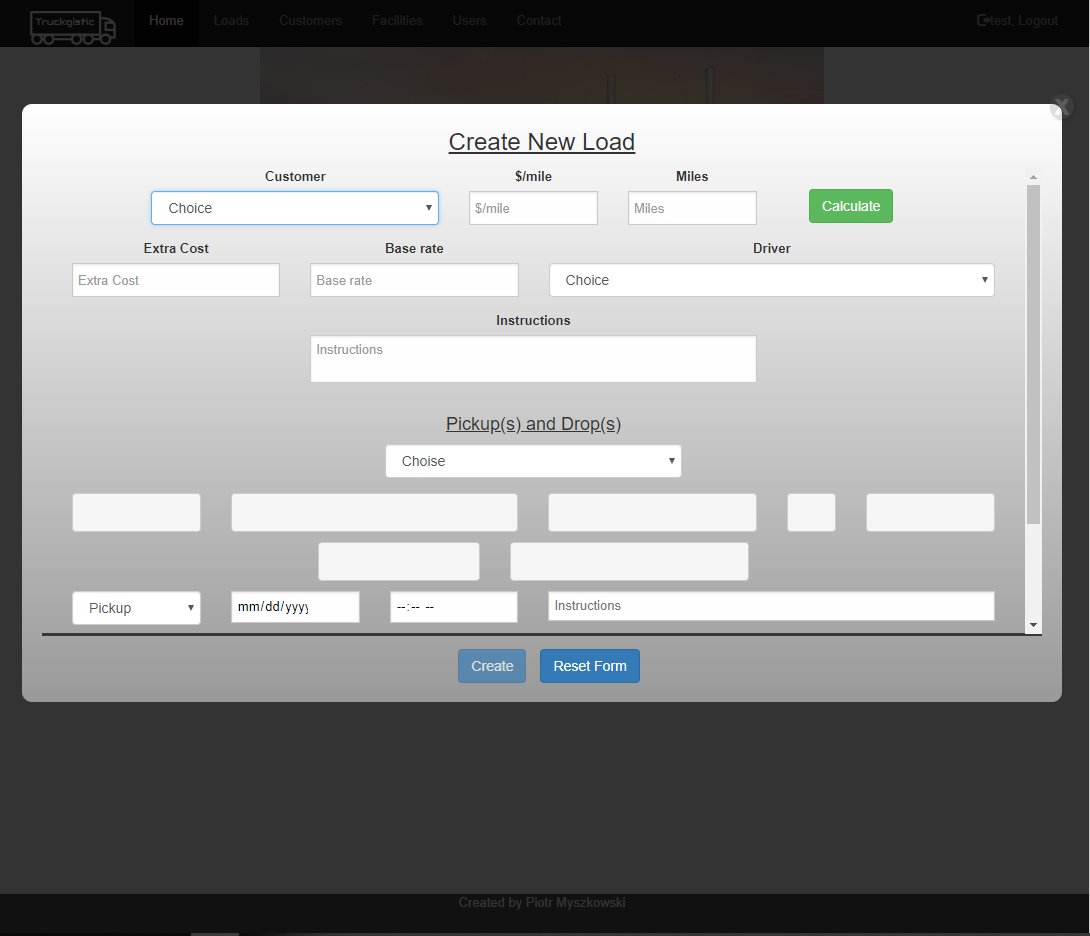


Figure 4.17 – Load create load

Second section of our form is Pick and Drop tool. Here we selecting from dropdown facilities we have in our database. These represent pickup or delivery location. Blank input you see on the picture below are information about the facility selected in the dropdown. When we are done with selecting the facility we have to choose the type of the stop (pickup/delivery), time and date. After we are done with filling out the form we can hit create button. The load will be created and server will notify all users that there is new load.

Customer’s page is responsible for managing customers. You can create, delete, edit, search customers or export all customers to CSV file and all users will be able to see the change right away on their UI without reloading the page. Customers are required field for creating a new load.

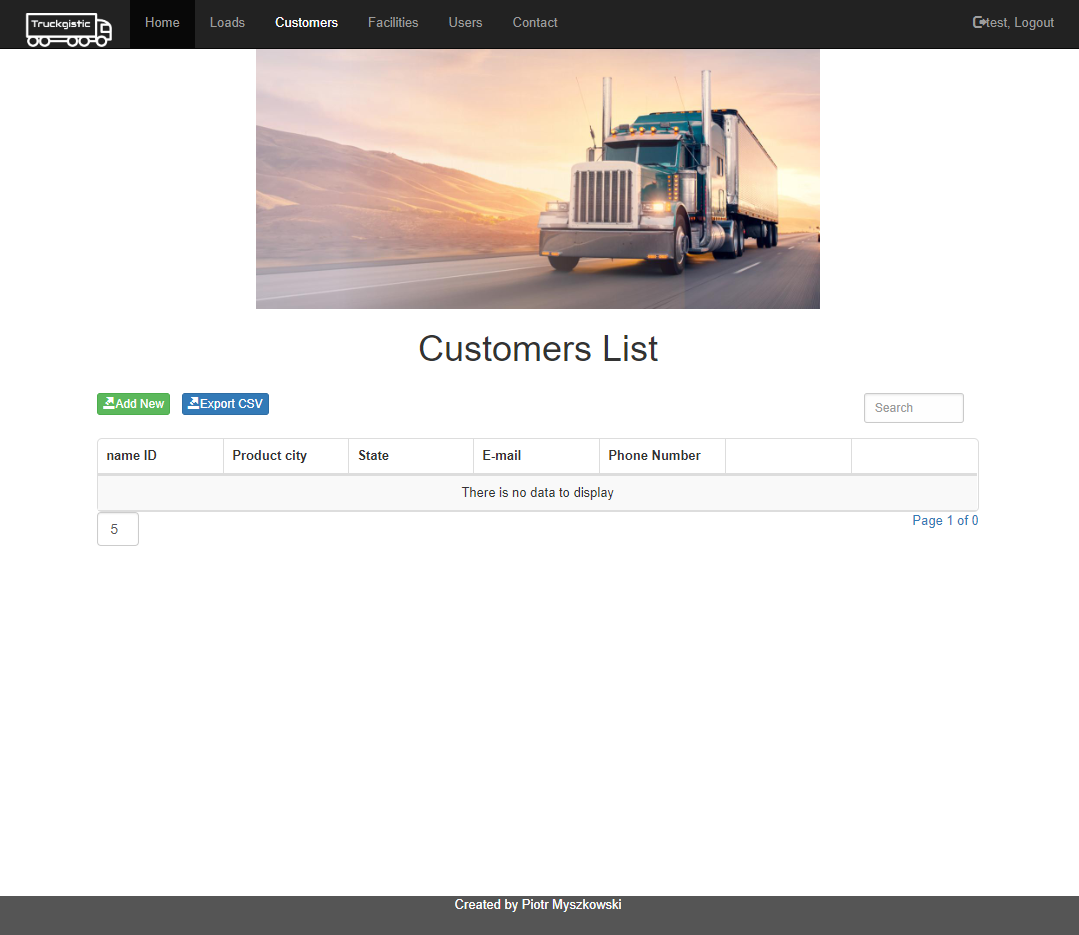


Figure 4.17 – Customer’s page

When creating a new customer, we have to know their detailed information’s. Like MC Number, DOT Number, Company, address, etc. There is description field that let user add additional notes about the customer, it is optional.

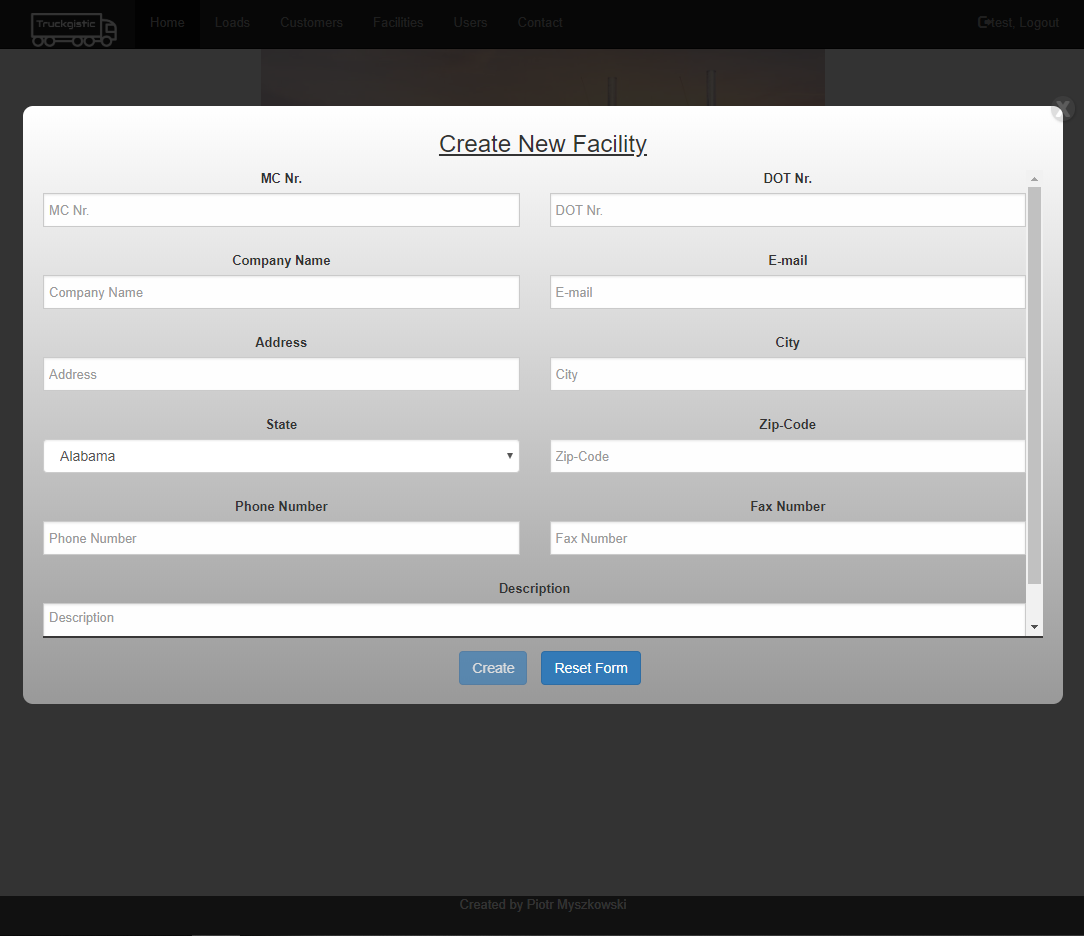


Figure 4.18 – Create new customer

In Facility page it is really similar to Customer page. You can do exactly same task except here our entity is facility.

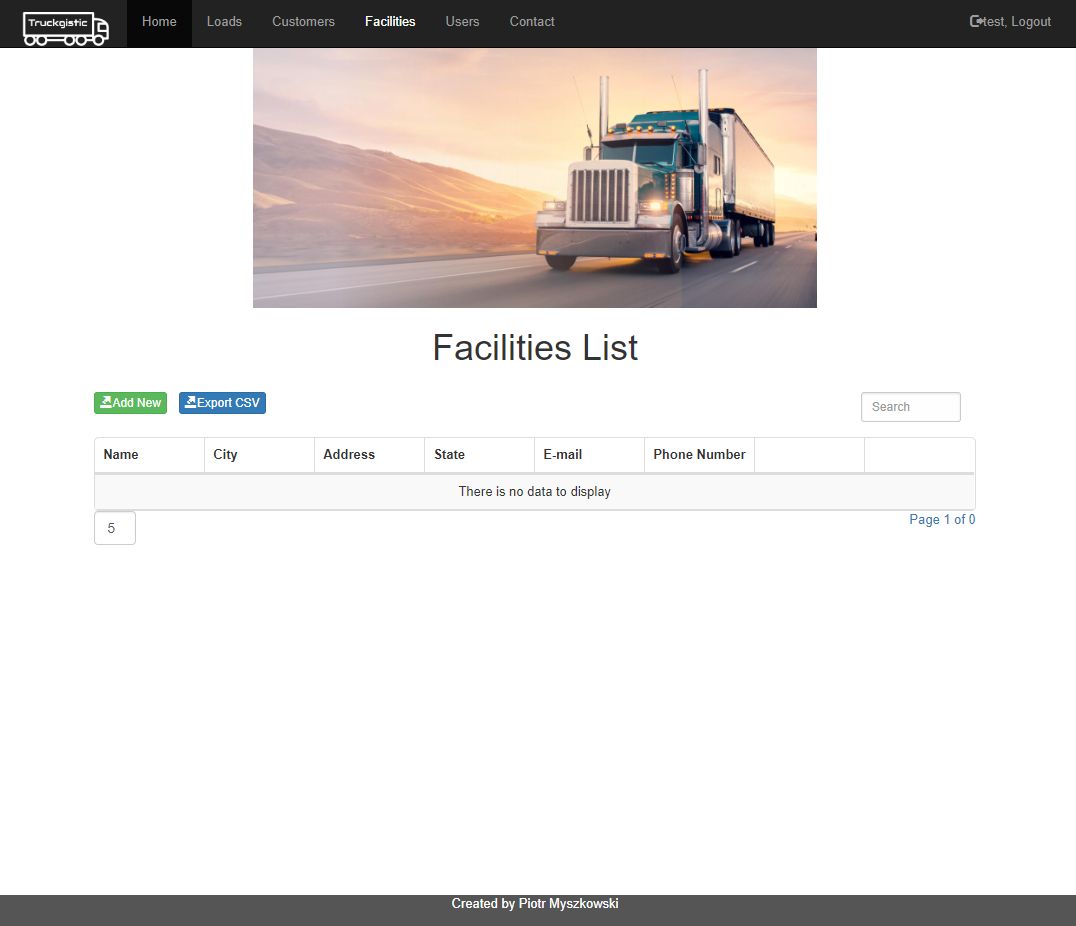


Figure 4.19 – Facilities List

Create facility process is similar to create customer process as well. There are fewer inputs to populate since this part in logistics industry is as not critical as customer who will pay us for the service.

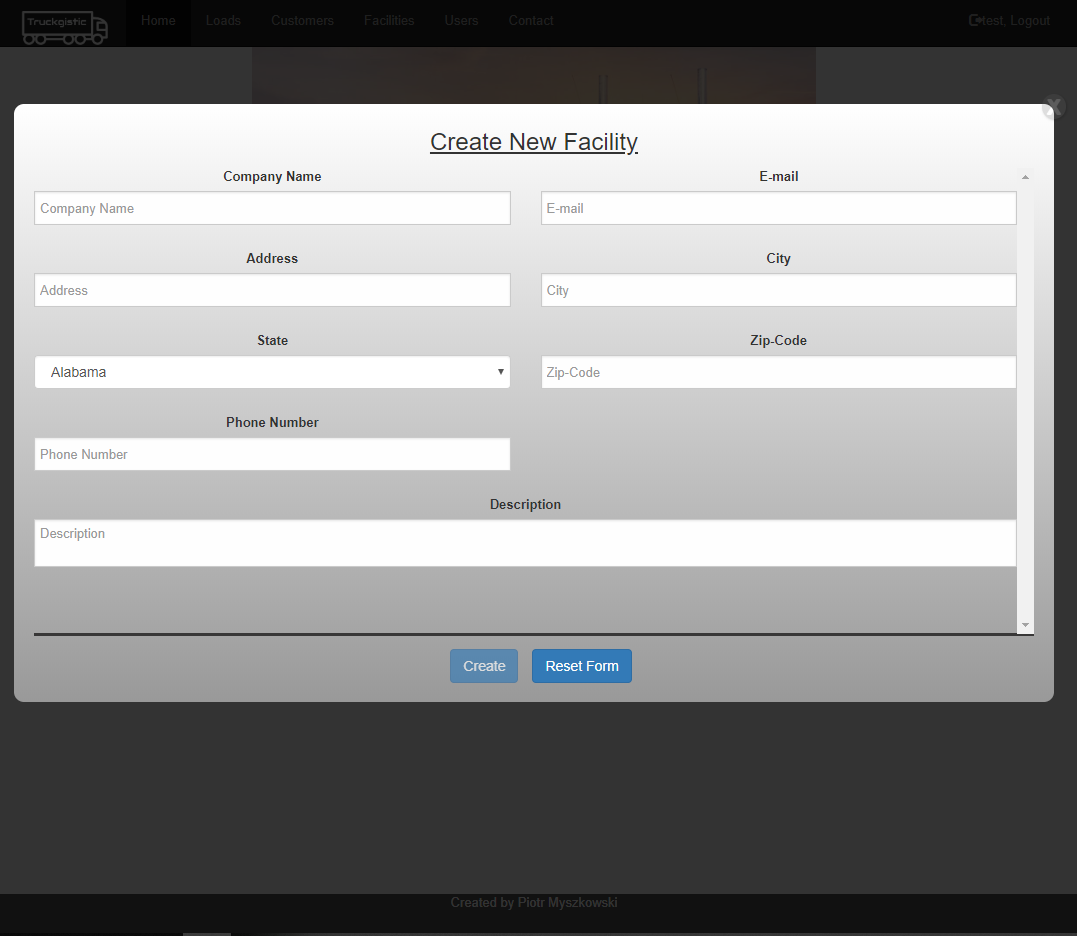


Figure 4.20 – Create new facility

Our next page is users/driver page. Here we can manage our drivers. When user creates new driver our web application creates new account that this driver can access the application from any browser or device and check loads he is assign to. Most of the functionality on this page are inherit from Customers, Facilities and Loads pages. We can also delete, remove, edit drivers.

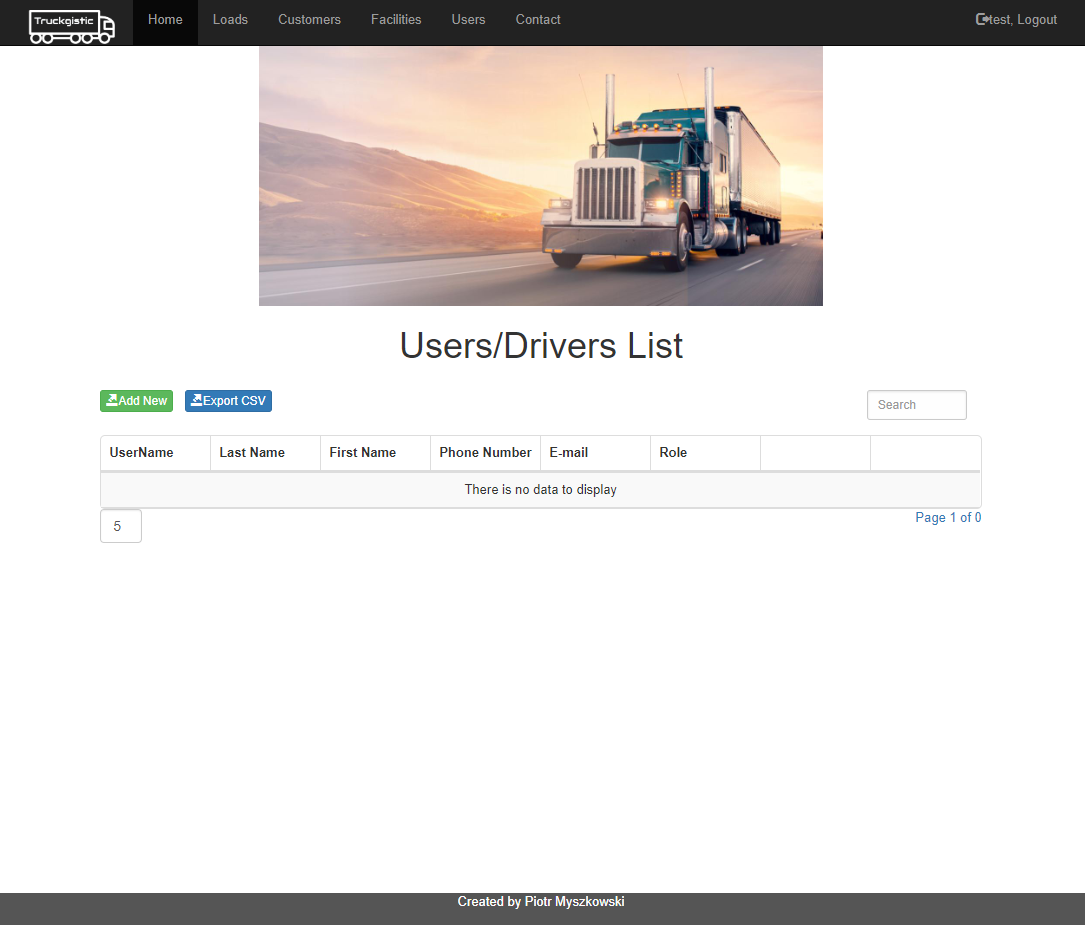


Figure 4.21 – Users list

The creation form for new driver has basic information about the driver. Notice that there is no password specified when creating new driver. It is because driver will receive the link where he can create his own password.

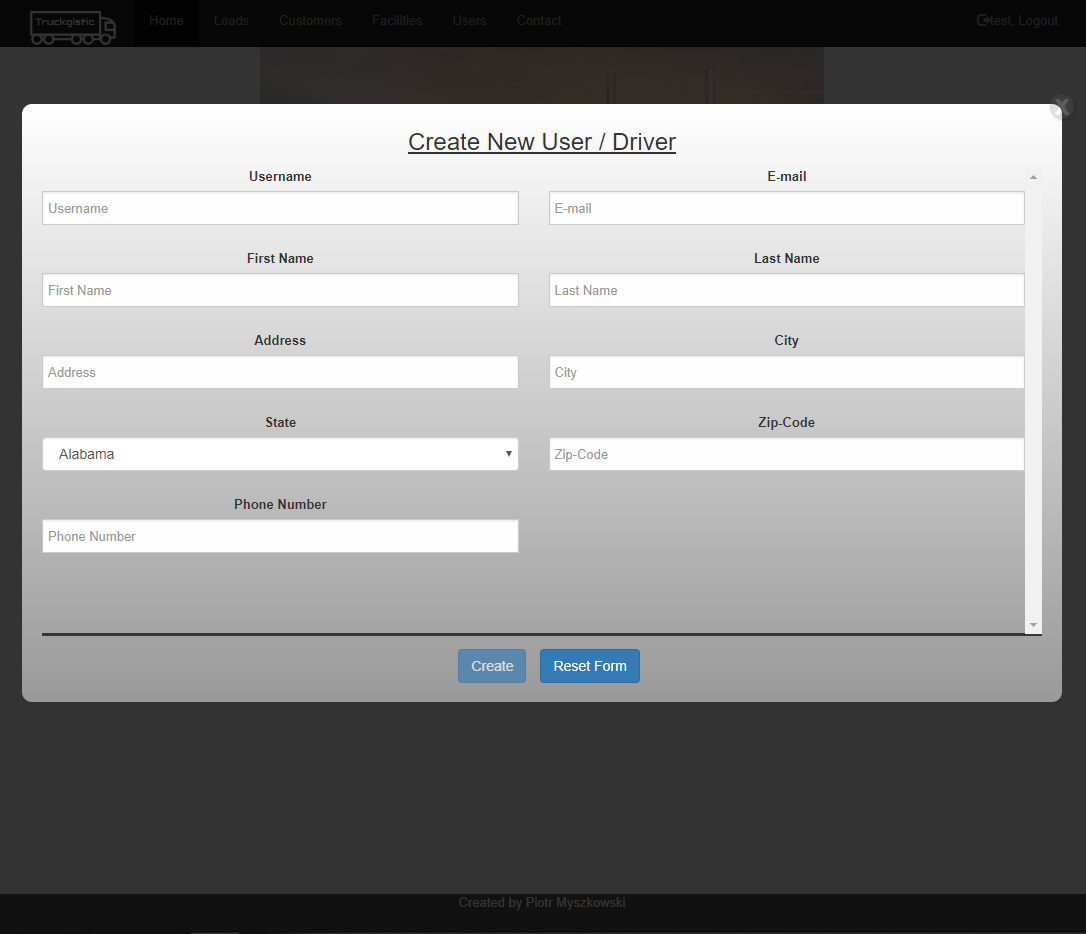


Figure 4.22 – Users create user

Driver account is an account with limitless possibility. Driver is able only to see loads are assign to it. Each driver account belongs to a different manager account.

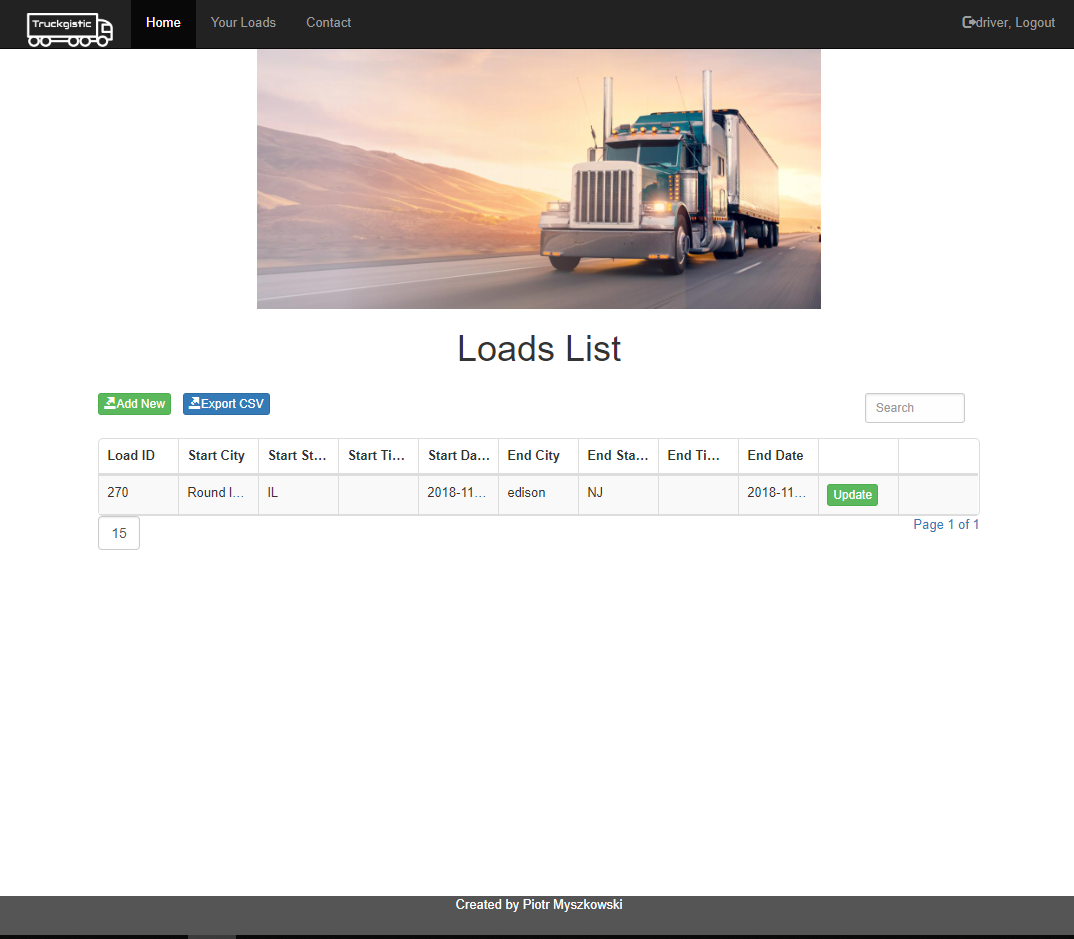


Figure 4.23 – Driver account loads

The last page Contact contain location and contact information.

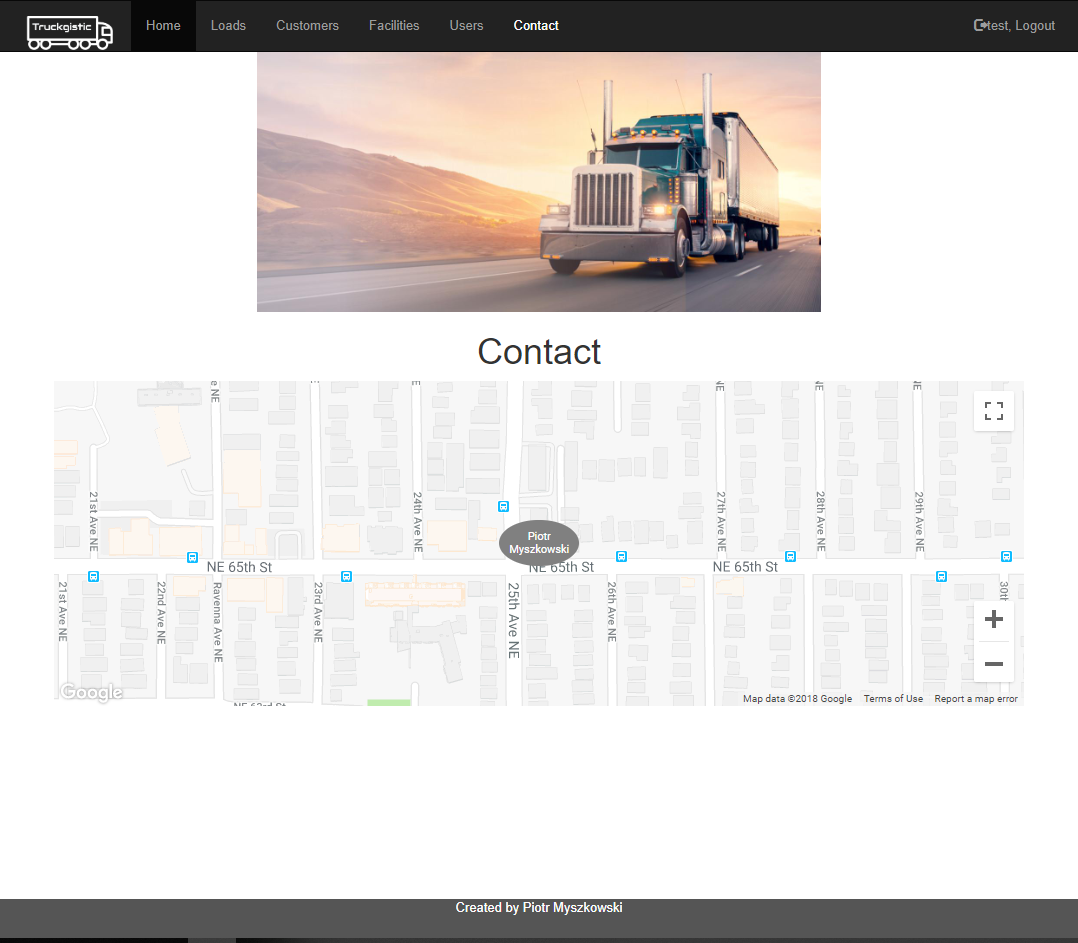


Figure 4.23 – My contact information’s

Providing a successful user experience requires a balanced approach throughout the development life cycle. To ensure this balance, you must not only focus on implementing the functionality required to complete a task but also on how the task is exposed through the user interface. However, the user interface of an application is only one aspect of the overall user experience. Other aspects of the user experience that are not visible to the user, but are integral to an application and critical to its usability, include start up time, latency, error handling, and automated tasks that are completed without direct user interaction.

# 5 Testing

Testing is a process that ensures that the quality of the software adheres to the standard laid down by the requirements specification and performs smoothly with the fewest possible glitches. For this, a technique called unit testing is the simplest to achieve and can be applied on all applications in their development. Unit testing allows each component of the application to be tested separately. Further, when these individual components are integrated and tested to ensure that multiple components collaborate to work well in the system, it is called integration testing.

In a development environment, usually test codes are written separately and are not mixed with the normal code. In fact, testing a program is distinctly a separate project and is stored in a separate directory. The tests constitute test cases that are applied to test the application manually or automatically. Automated test cases are applied repeatedly at different phases of the software development process. This, by the way, is the most recommended process of the Agile framework. And Spring, being a proponent of the framework, readily supports it.

The major advantage of applying dependency injection is that it makes a program code much easier to be tested because the code is much less dependent on the container, as may be the case with other development. The POJOs are individually testable by simple instantiation without adhering to any dependency. Sometimes, the fine line between unit and integration testing overlaps at the best interest of testing and we need to go beyond unit testing and start performing integration testing without deploying the application or connecting to another infrastructure. In such a case, we can mock objects to obtain the value of integration yet test our code in isolation. For example, we can test a service layer or a controller by stubbing or mocking DAO objects, without actually accessing the persistent data.

Spring Boot provides a set of utilities and annotation to help in the process of unit and integration testing and, of course, makes life easier. The core items for the testing are contained in the modules called spring-boot-test and the configuration is provided by the modules called spring-boot-test-autoconfigure.

We can simply use the spring-boot-starter-test in pom.xml and transitively pull all the required dependencies in a Spring application. The support libraries for testing as pulled by the Maven file are as follows:

– JUnit: It is the standard Java unit testing framework which provides an up-to-date foundation for developer-side testing on the JVM;

– Spring Test: Utilities and Integration support for Spring Boot applications;

– AssertJ: A set of assertions to provide meaningful error messages that leverage readability and easy debugging;

– Hamcrest: Provides a library of matcher object that helps in creating flexible expressions;

– Mockito: Java Mocking framework;

– JSONassert: Helps in writing JSON unit test and testing REST interfaces;

– JsonPath: Xpath for Json.

Spring Boot provides a volley of annotation to designate a test class and test specific parts of an application. For example, we can use *@SpringBootTest* annotation to enable Spring Boot test features. This annotation loads *ApplicationContext* used in a test, through *SpringApplication*. Typically, the *@SpringBootTest* annotation is used in association with *@RunWith(SpringRunner.class)* and does a whole lot of thing behind the scene apart from just loading *ApplicationContext* and having spring beans auto wired to the test instances.

The Spring test framework integrates well with test frameworks such as Junit, Mockito, and so on. As a result, unit and integration testing becomes easy with a more meaningful outcome. Springs makes it simpler by leveraging annotation-based support. For example, to unit and integration test the Spring TestContext framework, we can use @RunWith, @WebAppConfiguration, and @ContextConfigurationto load a Spring configuration and inject WebApplicationContext into the MockMvc.

## 5.1 Functional Testing

Table 5.1 – Register and login tests (Positive Test Case)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Actual Result | Expected Result | Test Result | |
| Start web application server | Show homepage on localhost:8080. | Show homepage on localhost:8080. | PASS | |
| Register page | Display register page. | Display register page. | PASS | |
| Register new user | Create user account and receive confirmation email | Create user account and receive confirmation email | PASS | |
| Create password | From link send over email create new password for a new account | From link send over email create new password for a new account | PASS | |
| Login | Login to the web application | Login to the web application | PASS | |
| Logout | Logout from a web application and destroy the session | Logout from a web application and destroy the session | PASS | |
| Create password as driver | From link send over email create new password for a new account | From link send over email create new password for a new account | PASS | |
| Show list of loads | Show list of loads that belongs to the account | Show list of loads that belongs to the account | | PASS |
| Show list of customers | Show list of customers that belongs to the account | Show list of customers that belongs to the account | | PASS |
| Show list of facilities | Show list of facilities that belongs to the account | Show list of facilities that belongs to the account | | PASS |
| Show list of drivers | Show list of drivers that belongs to the account | Show list of drivers that belongs to the account | | PASS |
| Create a new load | Create new load and show it on the page without refreshing the page | Create new load and show it on the page without refreshing the page | | PASS |
| Create a new customer | Create new customer and show it on the page without refreshing the page | Create new customer and show it on the page without refreshing the page | | PASS |
| Create a new facility | Create new facility and show it on the page without refreshing the page | Create new facility and show it on the page without refreshing the page | | PASS |
| Create a new driver | Create new facility and show it on the page without refreshing the page and send confirmation email | Create new facility and show it on the page without refreshing the page and send confirmation email | | PASS |
| Update existing load | Update existing load with a new information | Update existing load with a new information | | PASS |
| Update existing customer | Update existing customer with a new information | Update existing customer with a new information | | PASS |
| Update existing facility | Update existing facility with a new information | Update existing facility with a new information | | PASS |
| Update existing driver | Update existing driver with a new information | Update existing driver with a new information | | PASS |
| Delete a load | Delete existing load and update page without reloading | Delete existing load and update page without reloading | | PASS |
| Delete a customer | Delete existing customer and update page without reloading | Delete existing customer and update page without reloading | | PASS |
| Delete a facility | Delete existing facility and update page without reloading | Delete existing facility and update page without reloading | | PASS |
| Delete a driver | Delete existing driver and update page without reloading | Delete existing driver and update page without reloading | | PASS |

# 6 Economics

TruckGistics is a web application that works as CRM allowing different freight companies register create accounts for their drivers and start using it with less then couple minuets of registering. Its designers aimed it primarily at a security, notifications and easy user interface.

1. **Calculation of cost of materials for project accomplishment**

The estimate of costs for carrying out of scientifically research work settles payments under following clauses. Calculation is performed under the formula:

|  |  |  |
| --- | --- | --- |
|  | *n* |  |
|  | Рм = Ктр  Нpi Ci |  |
|  | *i*1 |  |

Where; K*TP* – the coefficient considering hauling expenses

(K*TP* ≈ from 1.0 to 1.10) for the project we accept KTP = 1;

Н*pi* - norm of the expense a material kind on the project;

C*i* - unit selling price a material kind, dollar.;

n - Quantity of applied kinds of materials.

Calculation to produce in the table.

Table 1 – Calculation of costs for the materials necessary for accomplishment of the project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| The name of materials | Unit of measure | The price, dollar | Quantity | The sum, dollar. |
|
|
|
| 1 | 2 | 3 | 4 | 5 |
| Paper format А1 | Sheet | 1 | 1 | 1 |
| Paper format А4 | Sheet | 0.85 | 40 | 34 |
| Stationery | - | - | - | 80 |
| Materials for experiences and designing | - | - | - | 150 |
| The sum of expenses | - | - | - | 271 |

The estimate of costs for carrying out of scientifically research work settles payments under following clauses:

1. **Calculation of a base salary of the personnel occupied with accomplishment of works under the project.**

The size of costs settles payments under the formula:

|  |  |  |
| --- | --- | --- |
|  | *n* |  |
|  | Рow = Кnp  Tci \* Ni \* ti |  |
|  | *i*1 |  |

Where, T*Ci* - a wage rate for a day, categories of workers, dollar

N*i* - quantity of workers of a category;

t*i* - time of actual work of the worker of a category under the project, day;

К*np* - coefficient of awards on bonus systems,

(К*np* ≈ from 1,10 to 1,40) for the project it is accepted К*np*=1,2.

Calculation to produce in the table.

Table 4 - Calculation of costs under the clause «a personnel base salary pay» under the project

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| The name of categories of workers and posts | Quantity of units, the people | Salary for one month, dollar | Coefficient of bonus surcharge | Expenditures of labor, months | The sum, dollar |
|
|
|
|
| 1 | 2 | 3 | 4 | 5 | 6 |
| The supervisor of studies of the project | 1 | 250 | 1.2 | 3 | 750 |
| The engineer | 1 | 200 | 1.2 | 3 | 600 |
| The sum of expenses | - | - | - | - | 1350 |

1. **Calculation of an additional salary of the contractors, including the various payments provided by the labor law, under the formula:**

Additional wages include a variety of performers stipulated by the labor

legislation of the payment and is calculated according to the formula:

Pnw = Pow \* ,

Where, Н*nw* - the specification of an additional salary,

Н*nw* ≈ from 10 to 25 %, for the project it is accepted Н*nw*=20 %.

Pnw = 1350 \* = 270 dollar

1. **Calculation of deductions to social insurance under the formula:**

Poc = (Pow + Pnw) \* ,

Where, H*oc* - rate of deductions on social insurance (the tax), the N*oc* = 34,6 %.

Poc = (1350 + 270) \* = 560.52

1. **Calculation of expenses on scientific business trip under the formula:**

We calculate the other expenses for materials scientific and technical information and the fee for the use of internet and telephone, etc.

The cost is calculated according to the formula:

Pkom = Pow \* ,

Where, Hkom – the specification on scientific business trip expenses,

Hkom ≈ from 5 to 20%, for the project we accept Hkom = 20%.

Pkom = 1350 \* = 270 dollars

1. **Calculation of common enterprise expenses under the formula:**

Indirect cost includes the cost of management and overhead cost, calculated according to the formula

Pkoc = Pow \* ,

Where, Нkос - the specification of indirect expenses,

Нkос ≈ from 50 to 100 %, for the project it is accepted Нkос = 90 %

Pkoc = 1350 \* = 1215 dollars

1. **Calculation of the complete cost value of the project:**

The total cost of scientific and technical products is determined as the sum of all cost in all respects (clauses 1-6) as according to the formula:

Cn = Pm + Pow + Pnw + Poc + Pkom + Pkoc

Cn = 271+1350+270+560.52+270+1215=3936.52 dollars

1. **On level of profitability in percentage of the complete cost value the profit settles payments**:

At the average level of profitability in percent of the total cost is determined by the target profit unit of scientific and technical products according to the formula:

Pr = Cn \* ,

Where, Yp - profitability level,

Yp ≈ from 10 to 30 %, for the project we accept Yp = 30%.

Pr = 3936.52 \* = 1180.60 dollars

1. **Calculation of the price of the project under the formula:**

To determine an approximate (estimated) wholesale price of scientific and technical products according to the formula

B*n* = C*n* + P*r*

B*n =* 3936.52 + 1180.60 = 5117.12 dollars

1. **Calculation of the tax to value added (VAT) under the formula:**

The Value Added Tax is determined by the formula:

VAT = Bn \* ,

Where, Hvat - the tax rate on vat (the tax), Hvat = 20%.

VAT = 5117.12 \* = 1023.42 dollars

1. **Calculation of the price of the project taking into account the VAT under the formula:**

To determine the selling price of scientific and technical products with VAT according to the formula:

B = B*n* + VAT

B = 5117.12 + 1023.42 = 6140.54 dollars

Calculation of costs for the project and the project price are resulted in table 3.

|  |  |  |
| --- | --- | --- |
| Clauses of costs | Calculation | The sum dollar. |
| 2 | 3 | 4 |
| Materials (Pm) | Table 1 |  |
| Base salary ( Pow ) | Table 2 |  |
| The additional salary ( Pnw ) | 1350 \* | 270 |
| Deductions in population social insurance  found ( Poc ) | Poc = (1350 + 270) \* | 560.52 |
| Scientific business trip expenses ( Pkom) | 1350 \* | 270 |
| Common enterprise expenses (Рkoc) | 1350 \* | 1215 |
| Total the cost value (Cn) | 271+1350+270+560.52+270+1215 | 3936.52 |
| Profit ( Pr) | 3936.52 \* | 1180.60 |
| The project price (Bn) | 3936.52 + 1180.60 | 5117.12 |
| The value-added tax (VAT) | 5117.12 \* | 1023.42 |
| The price from the VAT (B) | 5117.12 + 1023.42 | 6140.54 |

## Conclusion

TrackTreck where estimated time of arrival prediction is based on Chicago Transit Agency services and also map integration of stop and routes has been achieved. Costs for development of such system have constituted 6140.54.

# Conclusion

TruckGistic web application is based on SQL database, Java Spring Framework responsible for API/authentication layer and ReactJS the library of JavaScript will do front end of the application. Everything can be hosted on the cloud under one VM server but if scalability is the concern we can easily divide each of the layer to be run on independent container or VM that can help scale up or scale down the resources depending on the demands.

From here this web application can be enhance with more features to help freight companies adopt it in their business. For example we can make integration with different accountant software’s like Quick Books to make processing invoices quicker and more efficient.

Another good concept is to integrate this web application with blockchain technology since big companies that are dependent of transportation like Walmart already announced it that in order to deliver goods the company who makes delivery has to be on the blockchain network.

There are many different ways to make this application more unique and on demand especially if we would integrate blockchian technology, AI, IoT devices to this solution.

Costs for development of such application have constituted at 6140.54 dollars.

# References

[1] Spring Documentation on Java Spring Framework + ReactJS [Electronical source] -

<https://spring.io/guides/tutorials/react-and-spring-data-rest/> (Date 04/25/2018)

[2] ReactJS Documentation [Electronical source] -

<https://reactjs.org/> (Date 03/25/2018)

[3] Grails [Electronical source] –

<http://guides.grails.org/react-spring-security/guide/index.html> (Date 08/25/2018)

[4] Codebyamir [Electronical source] –

<https://www.codebyamir.com/blog/user-account-registration-with-spring-boot> (Date 06/17/2018)

[4] Stvmlbrn [Electronical source] –

<https://blog.stvmlbrn.com/2017/01/16/form-validation-with-react.html> (Date 08/25/2018)

[5] Stackoverflow [Electronical source] –

<https://stackoverflow.com/questions/44824590/javascript-react-sessionstorage-if-loggedin-change-nav-links> (Date 08/25/2018)

[6] Spring [Electronical source] –

<https://spring.io/docs> (Date 09/25/2018)

[7] Documentation on Java 8 [Electronical source] –

<https://docs.oracle.com/javase/8/docs/api/> (Date 02/23/2018)

[8] Digitalocean [Electronical source] –

<https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-8-on-centos-7> (Date 04/25/2018)

[9] Tomcat [Electronical source] –

<http://tomcat.apache.org/> (Date 02/14/2018)

[9] Centos [Electronical source] –

<https://www.centos.org/about/>

[10] Digitalocean MySQL on Centos [Electronical source] –

<https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-centos-7> (Date 01/02/2018)

[11] MySQL [Electronical source] –

<https://www.mysql.com/> (Date 01/01/2018)

[12] MySQL [Electronical source] –<https://dev.mysql.com/doc/refman/8.0/en/tutorial.html> (Date 01/01/2018)

# Appendix A

**Source Code**

This appendix contains some source code of the web application.

FreightCRMApplication.java

/\*

\* Copyright 2015 the original author or authors.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package com.freight.crm;

import java.util.Arrays;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.security.data.repository.query.SecurityEvaluationContextExtension;

*@SpringBootApplication*

public class FreightCRMApplication {

public static void main(String[] args) {

SpringApplication.run(FreightCRMApplication.class, args);

}

*@Bean*

public SecurityEvaluationContextExtension securityEvaluationContextExtension() {

return new SecurityEvaluationContextExtension();

}

}

SecurityConfiguration.java

package com.freight.crm.security;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

@Autowired

private SpringDataJpaUserDetailsService userDetailsService;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(this.userDetailsService);

auth.authenticationProvider(authenticationProvider());

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/built/\*\*","/bower\_components/\*\*", "/\*.js", "/js/\*\*","/modalForm.css","/homePage.css",

"/\*.jsx", "/main.css", "/login2.css","/confirm", "/register", "/resetPassword","/css/\*\*","/","/img/\*\*").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.httpBasic()

.and()

.csrf().disable()

.logout()

.logoutSuccessUrl("/");

}

@Bean

public DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();

authenticationProvider.setUserDetailsService(userDetailsService);

authenticationProvider.setPasswordEncoder(passwordEncoder());

return authenticationProvider;

}

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

CustomerEventHandler.java

package com.freight.crm.event.handlers.ws;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.rest.core.annotation.HandleAfterCreate;

import org.springframework.data.rest.core.annotation.HandleAfterDelete;

import org.springframework.data.rest.core.annotation.HandleAfterSave;

import org.springframework.data.rest.core.annotation.HandleBeforeCreate;

import org.springframework.data.rest.core.annotation.HandleBeforeDelete;

import org.springframework.data.rest.core.annotation.HandleBeforeSave;

import org.springframework.data.rest.core.annotation.RepositoryEventHandler;

import org.springframework.hateoas.EntityLinks;

import org.springframework.messaging.simp.SimpMessagingTemplate;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Component;

import static com.freight.crm.WebSocketConfiguration.\*;

import com.freight.crm.entity.Customer;

import com.freight.crm.entity.User;

import com.freight.crm.service.IUserService;

@Component

@RepositoryEventHandler(Customer.class)

public class CustomerEventHandler {

private final IUserService userService;

private final SimpMessagingTemplate websocket;

private final EntityLinks entityLinks;

@Autowired

public CustomerEventHandler(SimpMessagingTemplate websocket, EntityLinks entityLinks, IUserService userService) {

this.websocket = websocket;

this.entityLinks = entityLinks;

this.userService = userService;

}

@HandleAfterCreate

public void newCustomer(Customer customer) {

this.websocket.convertAndSend(

MESSAGE\_PREFIX + "/newCustomer", getPath(customer));

}

@HandleAfterDelete

public void deleteCustomer(Customer customer) {

this.websocket.convertAndSend(

MESSAGE\_PREFIX + "/deleteCustomer", getPath(customer));

}

@HandleAfterSave

public void updateCustomer(Customer customer) {

this.websocket.convertAndSend(

MESSAGE\_PREFIX + "/updateCustomer", getPath(customer));

}

@HandleBeforeSave

public void beforeUpdateCustomer(Customer customer) {

String name = SecurityContextHolder.getContext().getAuthentication().getName();

User usr = this.userService.findByUsername(name);

if (usr == null ) {

throw new NullPointerException("No user found");

}

customer.setUser(usr);

}

@HandleBeforeDelete

public void beforeDeleteCustomer(Customer customer) {

String name = SecurityContextHolder.getContext().getAuthentication().getName();

User usr = this.userService.findByUsername(name);

if (usr == null) {

throw new NullPointerException("No user found");

}

customer.setUser(usr);

}

@HandleBeforeCreate

public void applyUserInformationUsingSecurityContext(Customer customer) {

String name = SecurityContextHolder.getContext().getAuthentication().getName();

User usr = this.userService.findByUsername(name);

if (usr == null) {

throw new NullPointerException("No user found");

}

customer.setUser(usr);

}

/\*\*

\* Take an {@link Customer} and get the URI using Spring Data REST's {@link EntityLinks}.

\*

\* @param customer

\*/

private String getPath(Customer customer) {

return this.entityLinks.linkForSingleResource(customer.getClass(),

customer.getCustomerID()).toUri().getPath();

}

}

LoadController.java

package com.freight.crm.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.ApplicationEventPublisher;

import org.springframework.context.ApplicationEventPublisherAware;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.rest.core.event.AfterDeleteEvent;

import org.springframework.data.rest.core.event.BeforeDeleteEvent;

import org.springframework.data.rest.webmvc.PersistentEntityResourceAssembler;

import org.springframework.data.rest.webmvc.RepositoryLinksResource;

import org.springframework.data.rest.webmvc.RepositoryRestController;

import org.springframework.data.web.HateoasPageableHandlerMethodArgumentResolver;

import org.springframework.data.web.PagedResourcesAssembler;

import org.springframework.hateoas.ExposesResourceFor;

import org.springframework.hateoas.PagedResources;

import org.springframework.hateoas.Resource;

import org.springframework.hateoas.ResourceAssembler;

import org.springframework.hateoas.ResourceProcessor;

import org.springframework.hateoas.mvc.ControllerLinkBuilder;

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import com.freight.crm.entity.Load;

import com.freight.crm.entity.User;

import com.freight.crm.service.ILoadService;

import com.freight.crm.service.IPickDropService;

import com.freight.crm.service.IUserService;

import lombok.extern.slf4j.Slf4j;

@Slf4j

@ExposesResourceFor(Load.class)

@RequestMapping("/api")

@RepositoryRestController

public class LoadController implements ApplicationEventPublisherAware, ResourceProcessor<RepositoryLinksResource> {

private ILoadService loadService;

private final IPickDropService pdService;

private final IUserService userService;

private ApplicationEventPublisher publisher;

@Autowired

private PagedResourcesAssembler<Load> assembler;

@Override

public void setApplicationEventPublisher(

ApplicationEventPublisher publisher) {

this.publisher = publisher;

}

public LoadController(ILoadService loadService, IPickDropService pdService, IUserService userService,

ApplicationEventPublisher publisher) {

super();

this.loadService = loadService;

this.pdService = pdService;

this.userService = userService;

this.publisher = publisher;

}

@RequestMapping(method = RequestMethod.DELETE, value = "/loads/deleteLoadALL")

public ResponseEntity<?> deleteLoad(@RequestBody Long id) {

User usr = userService.findByUsername(SecurityContextHolder.getContext().getAuthentication().getName());

Load load = loadService.findOne(id);

if(!usr.getRole().equals("ROLE\_MANAGER") || load.getUser().getUserID() != usr.getUserID()) {

return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);

}

publisher.publishEvent(new BeforeDeleteEvent(load));

pdService.deleteByLoad(load);

loadService.delete(load);

publisher.publishEvent(new AfterDeleteEvent(new Load()));

return ResponseEntity.ok(id);

}

@RequestMapping(method = RequestMethod.GET, value = "/loads/loadsByDriver")

HttpEntity<PagedResources<Resource<Load>>> getloadsByDriver(PersistentEntityResourceAssembler entityAssembler,Pageable pageable) {

Page<Load> persons = loadService.findByDriver(pageable, SecurityContextHolder.getContext().getAuthentication().getName());

//Page<Load> persons = loadService.findAll(pageable);

@SuppressWarnings({ "unchecked", "rawtypes" })

PagedResources<Resource<Load>> resource = assembler.toResource(persons,

(ResourceAssembler) entityAssembler);

resource.add(ControllerLinkBuilder.linkTo(LoadController.class).slash("profile/loads").withRel("profile"));

return new ResponseEntity<>(resource, HttpStatus.OK);

}

@Override

public RepositoryLinksResource process(RepositoryLinksResource resource) {

resource.add(ControllerLinkBuilder.linkTo(LoadController.class).withRel("profile"));

return resource;

}

}

Client.js

'use strict';

var rest = require('rest');

var defaultRequest = require('rest/interceptor/defaultRequest');

var mime = require('rest/interceptor/mime');

var uriTemplateInterceptor = require('./api/uriTemplateInterceptor');

var errorCode = require('rest/interceptor/errorCode');

var baseRegistry = require('rest/mime/registry');

var registry = baseRegistry.child();

registry.register('text/uri-list', require('./api/uriListConverter'));

registry.register('application/hal+json', require('rest/mime/type/application/hal'));

module.exports = rest

.wrap(mime, { registry: registry })

.wrap(uriTemplateInterceptor)

.wrap(errorCode)

.wrap(defaultRequest, { headers: { 'Accept': 'application/hal+json' }});

uriListConverter.JS

define(function() {

'use strict';

/\* Convert a single or array of resources into "URI1\nURI2\nURI3..." \*/

return {

read: function(str /\*, opts \*/) {

return str.split('\n');

},

write: function(obj /\*, opts \*/) {

// If this is an Array, extract the self URI and then join using a newline

if (obj instanceof Array) {

return obj.map(function(resource) {

return resource.\_links.self.href;

}).join('\n');

} else { // otherwise, just return the self URI

return obj.\_links.self.href;

}

}

};

});

Customer.JS

'use strict';

import validator from 'validator';

import Form from 'react-validation/build/form';

import Input from 'react-validation/build/input';

import Button from 'react-validation/build/button';

import SelectValidator from './components/SelectValidator';

import InputValidator from './components/inputValidator';

import DescInput from './components/DescInput';

import {BootstrapTable, TableHeaderColumn, ExportCSVButton} from 'react-bootstrap-table';

import CreateDialog from './CustomerComponents/CreateDialog';

import UpdateDialog from './CustomerComponents/UpdateDialog';

import CustomerList from './CustomerComponents/CustomerList';

import React, { Component } from 'react';

const ReactDOM = require('react-dom')

const when = require('when');

const client = require('../client');

//export { default as form } from './Hoc/form';

//export { default as button } from './Hoc/button';

const follow = require('../follow'); // function to hop multiple links by "rel"

const stompClient = require('../websocket-listener');

const root = '/api';

class Customers extends React.Component {

constructor(props) {

super(props);

this.state = {customers: [], attributes: [], page: 1, pageSize: 5, links: {}};

this.updatePageSize = this.updatePageSize.bind(this);

this.onCreate = this.onCreate.bind(this);

this.onUpdate = this.onUpdate.bind(this);

this.onDelete = this.onDelete.bind(this);

this.onNavigate = this.onNavigate.bind(this);

this.refreshCurrentPage = this.refreshCurrentPage.bind(this);

this.refreshAndGoToLastPage = this.refreshAndGoToLastPage.bind(this);

}

loadFromServer(pageSize) {

follow(client, root, [

{rel: 'customers', params: {size: pageSize}}]

).then(customerCollection => {

return client({

method: 'GET',

path: customerCollection.entity.\_links.profile.href,

headers: {'Accept': 'application/schema+json'}

}).then(schema => {

this.schema = schema.entity;

this.links = customerCollection.entity.\_links;

return customerCollection;

});

}).then(customerCollection => {

this.page = customerCollection.entity.page;

return customerCollection.entity.\_embedded.customers.map(customer =>

client({

method: 'GET',

path: customer.\_links.self.href

})

);

}).then(customerPromises => {

return when.all(customerPromises);

}).done(customers => {

this.setState({

page: this.page,

customers: customers,

attributes: Object.keys(this.schema.properties),

pageSize: pageSize,

links: this.links

});

});

}

// tag::on-create[]

onCreate(newCustomer) {

follow(client, root, ['customers']).done(response => {

client({

method: 'POST',

path: response.entity.\_links.self.href,

entity: newCustomer,

headers: {'Content-Type': 'application/json'}

});

})

}

// end::on-create[]

onUpdate(customer, updatedCustomer) {

client({

method: 'PUT',

path: customer.entity.\_links.self.href,

entity: updatedCustomer,

headers: {

'Content-Type': 'application/json',

'If-Match': customer.headers.Etag

}

}).done(response => {

/\* Let the websocket handler update the state \*/

}, response => {

if (response.status.code === 403) {

alert('ACCESS DENIED: You are not authorized to update ' +

customer.entity.name);

}

if (response.status.code === 412) {

alert('DENIED: Unable to update ' + customer.entity.name + '. Your copy is stale.');

this.loadFromServer(this.state.pageSize); }

});

}

onDelete(customer) {

client({method: 'DELETE', path: customer.entity.\_links.self.href})

.done(response => {/\* let the websocket handle updating the UI \*/},

response => {

if (response.status.code === 403) {

alert('ACCESS DENIED: You are not authorized to delete ' +

customer.entity.name);

}

if (response.status.code === 409) {

alert('ERROR: Unable to DELETE ' + customer.entity.name + '. There is history on file.');

}

});;

}

onNavigate(navUri) {

client({

method: 'GET',

path: navUri

}).then(customerCollection => {

this.links = customerCollection.entity.\_links;

this.page = customerCollection.entity.page;

return customerCollection.entity.\_embedded.customers.map(customer =>

client({

method: 'GET',

path: customer.\_links.self.href

})

);

}).then(customerPromises => {

return when.all(customerPromises);

}).done(customers => {

this.setState({

page: this.page,

customers: customers,

attributes: Object.keys(this.schema.properties),

pageSize: this.state.pageSize,

links: this.links

});

});

}

updatePageSize(pageSize) {

if (pageSize !== this.state.pageSize) {

this.loadFromServer(pageSize);

}

}

// tag::websocket-handlers[]

refreshAndGoToLastPage(message) {

follow(client, root, [{

rel: 'customers',

params: {size: this.state.pageSize}

}]).done(response => {

if (response.entity.\_links.last !== undefined) {

this.onNavigate(response.entity.\_links.last.href);

} else {

this.onNavigate(response.entity.\_links.self.href);

}

})

}

refreshCurrentPage(message) {

follow(client, root, [{

rel: 'customers',

params: {

size: this.state.pageSize,

page: this.state.page.number

}

}]).then(customerCollection => {

this.links = customerCollection.entity.\_links;

this.page = customerCollection.entity.page;

return customerCollection.entity.\_embedded.customers.map(customer => {

return client({

method: 'GET',

path: customer.\_links.self.href

})

});

}).then(customerPromises => {

return when.all(customerPromises);

}).then(customers => {

if (!(window.location.href.substring(0,46) == "http://localhost:8080/Customers#updateCustomer")){

this.setState({

page: this.page,

customers: customers,

attributes: Object.keys(this.schema.properties),

pageSize: this.state.pageSize,

links: this.links

});

}

});

}

componentDidMount() {

this.loadFromServer(this.state.pageSize);

stompClient.register([

{route: '/topic/newCustomer', callback: this.refreshAndGoToLastPage},

{route: '/topic/updateCustomer', callback: this.refreshCurrentPage},

{route: '/topic/deleteCustomer', callback: this.refreshCurrentPage}

]);

}

render() {

return ( <div className="col-sm-10 col-sm-offset-1">

<CreateDialog attributes={this.state.attributes} onCreate={this.onCreate}/>

<CustomerList className="col-sm-12 mx-auto text-center" page={this.state.page}

customers={this.state.customers}

links={this.state.links}

pageSize={this.state.pageSize}

attributes={this.state.attributes}

onNavigate={this.onNavigate}

onUpdate={this.onUpdate}

onDelete={this.onDelete}

updatePageSize={this.updatePageSize}

refreshCurrentPage = {this.refreshCurrentPage}/>

</div>

)

}

}

export default Customers;