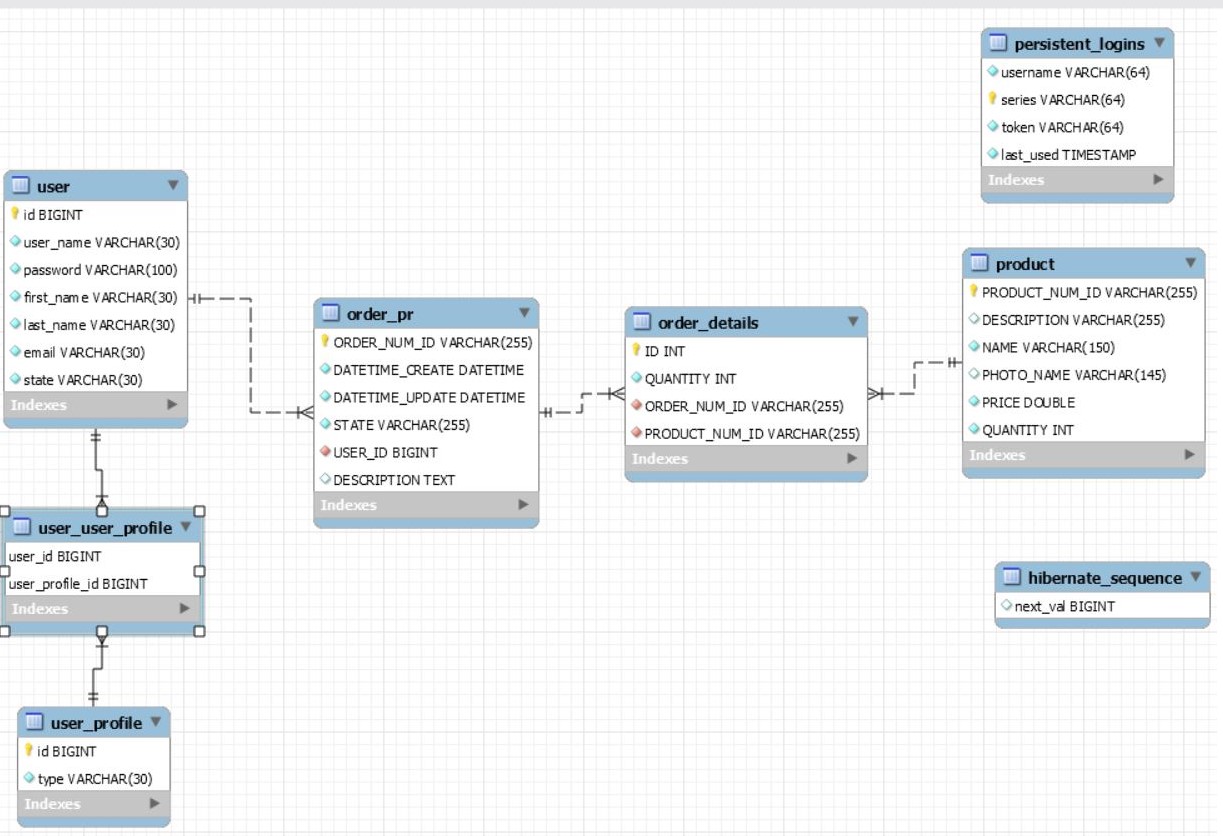
Version 1.0

The application is conceived as a Web Shop, in which customers are known. They can only be added by the administrator. So the whole application is a closed type, a kind of main warehouse and customers who are their well-known distributors (stores). ***Home I About*** us are pages that are advertising and available to all visitors. For all other pages it is necessary to log in, authorizations and access are in accordance with the roles that users receive from the administrator. The application was created for general use (not specialized activities), so for the purpose of more picturesque use, I use images, logo, name of my own company.

1.0 BASE

***Base MySQL 8.0.19***

****

*Picture of the base scheme*

Attached in the *mySql* folder, you have all the necessary sql queries to create the database.

The ***user*** table stores all users with their basic data. The ***password*** column keeps the user passwords stored encrypted. It is linked to the user\_profile table by a multiple-to-multiple link because the idea is that one ***user*** can have multiple roles.

Table ***user\_profile*** contains:

1. ADMIN – The role of the administrator.
2. USER – For now without a special role. Left for the possibility of rapid expansion.
3. EMPMENAGER – Employee manager.
4. WARMENAGER – Warehouse manager (Seller).
5. SHOPMENAGER – Shop menager (Customer).

The first user is the administrator. It is added by query. And the password is generated using the ***QuickPasswordEncodingGenerator*** class of the application, located in the application folder ***generators***. We use ***BCryptPasswordEncoder*** for encryption. The code is 123 a nick name admin.

*INSERT INTO`user` VALUES (1,'admin','$2a$10$94FgHW3muCFTDS4ipVhnm.Z1W1/v2m1PUyYT8LhLW0BJpF.3pbt7K','Nemanja','Matovic','neca@gmai.com','Active');*

The table ***hibernate\_sequence the*** stores the following value to create unique ID values in the tables ***order\_pr*** and ***product***.

The table ***order\_pr*** stores all data on created orders. Creation date and date update order are automatically created using the ***TIMESTAMP*** function of MySql.

The table ***order\_details*** stores the details of all orders.

The table ***product*** stores information about all products.

The table ***persistent\_logins*** stores data on a cookie, that remembers the user when the function ***remebe me*** is turned on in the application. It stores it for as long as specified by the application. In this case an hour.

2.0. APPLICATION

Spring ***MVC 4.1.6.*** was used for application development, ***Spring Security 4.0.1.*** for protection and authentication, and ***Hibernate 4.3.0.*** for integration with ***MySql*** database. **CRUD** operations are performed within transactions. Password storage is done with the help of ***Bcrypt***, and user memory functionality with the help of a custom implementation of **PersistentTokenRepository** with ***Hibernate HibernateTokenRepositoryImpl***. The ***SequenceStyleGenerator*** is used to generate unique primary keys. For display we use ***jsp pages, Bootstrap 4.6., Jquery-3.5.1., JavaScrip, HTML, CSS***.

***Before starting the application, it is necessary to set the file application.properties, which is located in the resource folder of the application for connection to the database. As well as the URL path to the static / photo folder where UPLOAD\_LOCATION images are stored in the ProductControlle.***

2.1. LOGIN/LOGOUT

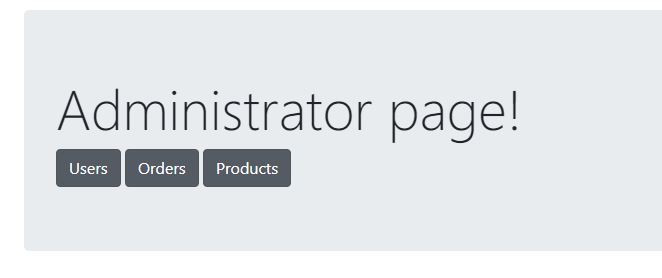
The ***Login*** button is only visible on the ***Home I About us*** page. It leads to a login page. In case the user is already logged in, click on the Login button to go to the home page of the logged in user, and it depends on the role assigned to the user. This is due to the created class ***TargetUrl*** and its method ***determineTargetUrl*** which extracts the roles of the currently logged in user from the identity object and constructs the appropriate URL based on the roles. The class ***CustomSuccessHandler*** extends ***SimpleUrlAuthenticationSuccessHandler*** and tramples its ***handle*** method in which it sets the default redirect strategy to the resulting URL.

On all other pages, the ***Login*** button goes to the user's greeting message and the ***Logout*** button, which logs off the user and returns him to the login page with the message that the user has been successfully logged out. The ***logout*** method provided by the ***PersistentTokenBasedRememberMeServices Spring Sec Security*** class allows us to: Disable an HTTP session, then undo all objects associated with it. Removes the identity certificate. Deletes the context value from the current thread. Deletes the memorized peg in the database.

The username of the logged in user is obtained by the ***getCurrentUser*** function, which is located in the controller classes. It returns the User name from the ***Spring SecurityContextHolder*** class.

The ***Remember me*** button allows the user to be remembered for the next hour.

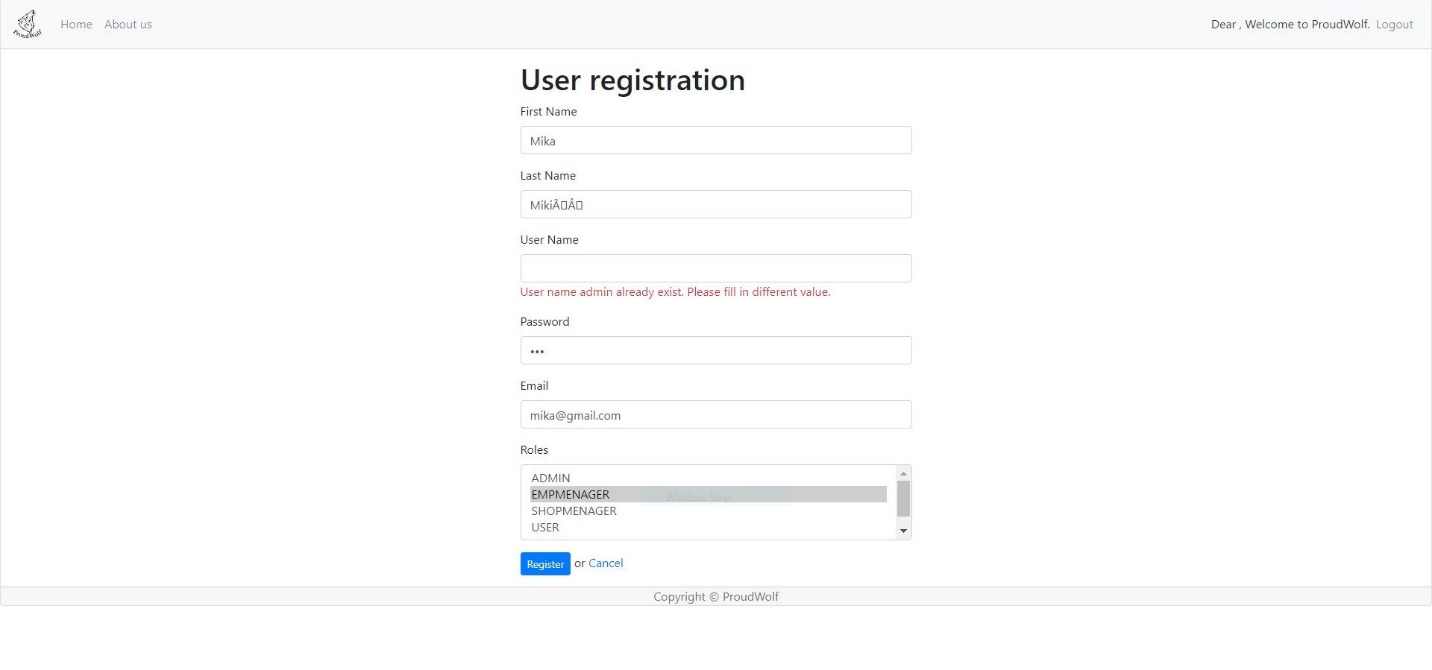
2.2. ADMIN



*Admin page*

In the example the Administrator has all the permissions and sees all the content. So most of the functions will be explained through other users. The ***Users*** button leads to the list of all users, ***Orders*** to all orders, ***Product*** to all products.

The uniqueness of the administrator is only in the fact that he can add, delete and modify users

*User registration form*

To create a form, we use the form tag from the springframework library, and we enable validation on the form with the help of annotations that we add to the fields of the model class. We use ***hibernate-validator 5.1.3.Final*** and ***javax-validation 1.1.0.Final***. With the ***MessageSource*** configuration, we print the desired messages in case the validation is not appropriate. The uniqueness of the username is achieved by using the ***isUserNameUnique*** method of the ***UserServiceImpl class***.

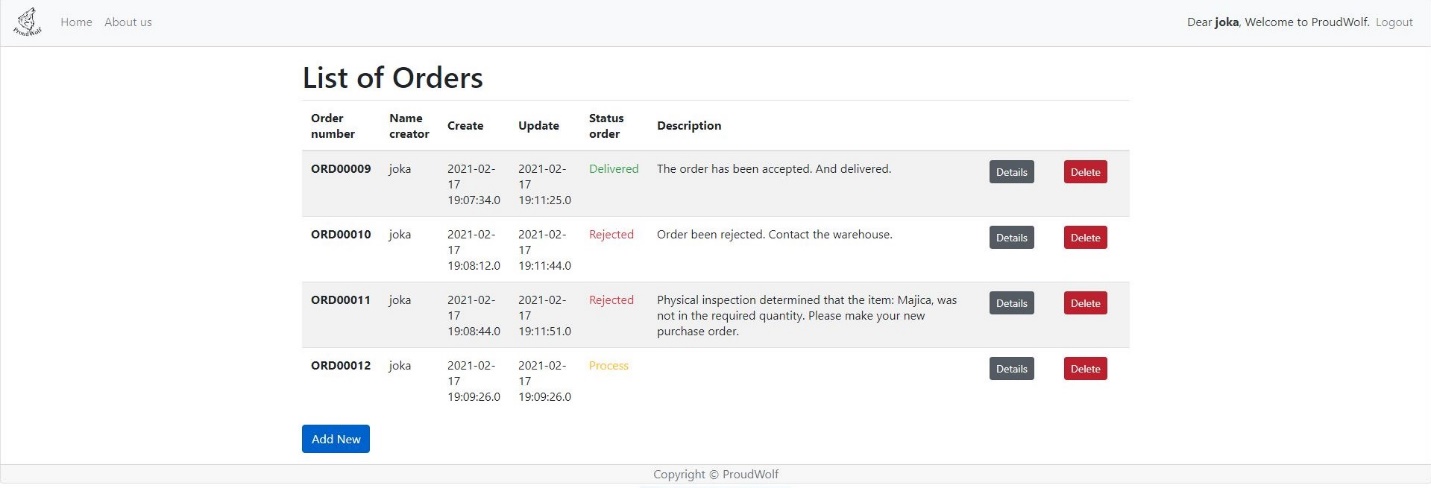
The user cannot be deleted if his data is used in any of the purchase orders.

2.3. EMPMENAGER

*User list*

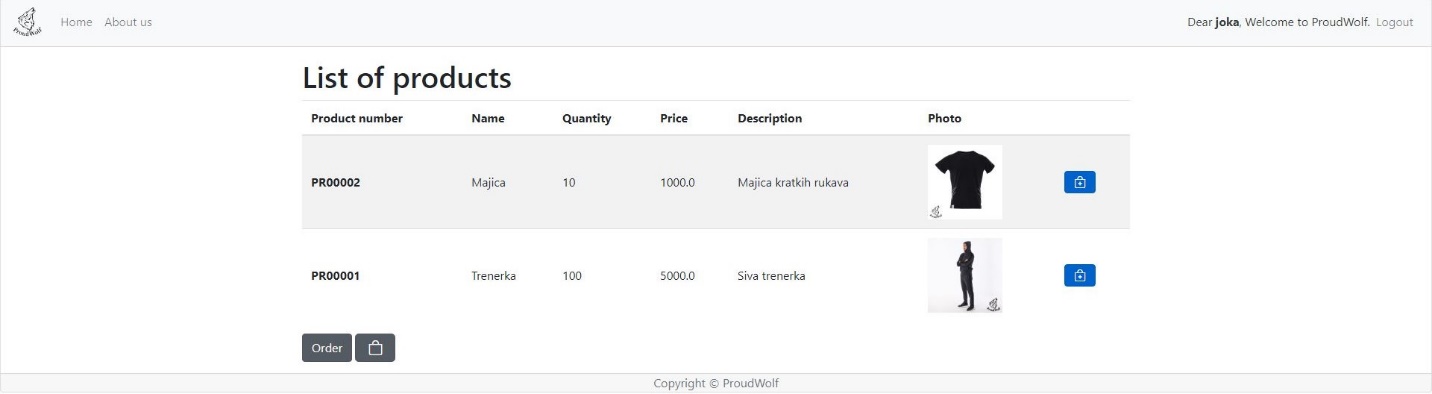
The employee manager has insight into all users (employees) and insight into all orders. Clicking on ***User state*** changes the state of the user and thus gives or takes away access to the application. It is impossible to change the status of the administration.

2.4. SHOPMENAGER

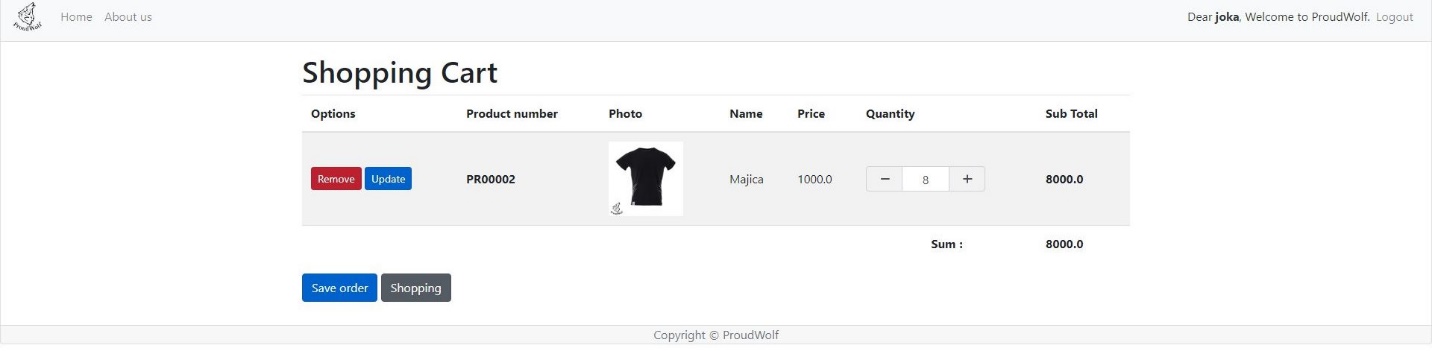


*Order list*

The home page, when he logs in, of the store manager (customer) is a list of all his orders. It can only delete them if they are in processing status. Click on the ***Details*** button to see the details of the purchase order, and by clicking ***Add New*** it goes to the list of all products to create a new one.

*List of all products*

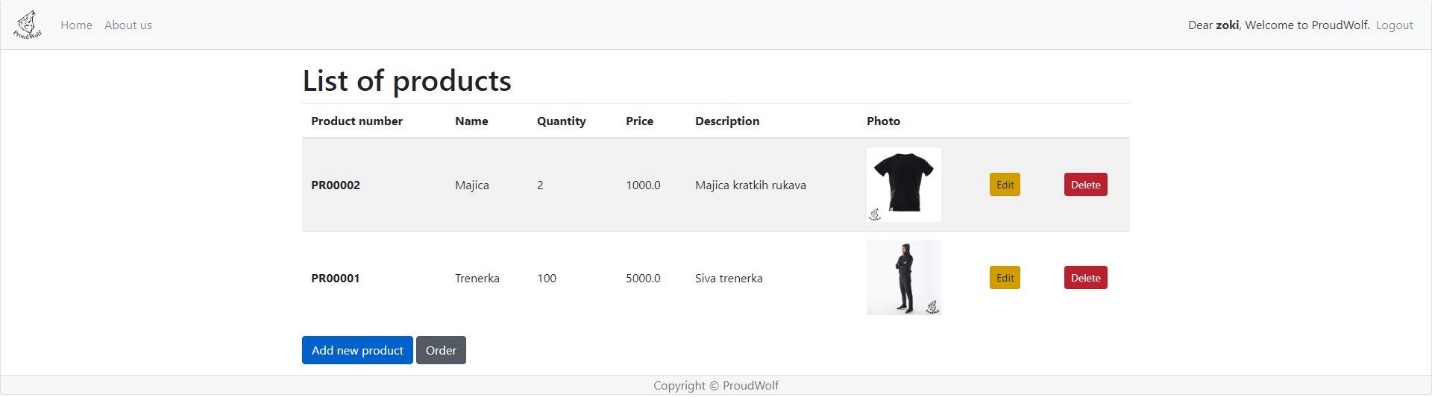
Clicking on the blue basket adds the product to the basket. Clicking on the gray basket gives an insight into it, if there are products in the basket. If the item already exists in the basket, it will not be added, but only an insight into the basket will be provided. This is made possible by the ***isExisting*** method located in the ***OrderController***.



*Cart*

Insight into the basket and the basket can only be reached by the customer. The list of products in the basket is created at the session level. The application remembers all products in the basket until the order is created or the session is terminated. The quantity of the product cannot be 0 or greater than the current state in the warehouse. The ***Sub Total*** column calculates the value of the desired quantity of the item. The ***Sum*** field calculates the value of the order. Clicking the ***Update*** button updates the desired amount, ***Remove*** removes the item. The ***Shopping*** button returns the customer to the list of all products to add a new item. The ***Save order*** button saves the order and deletes all items from the cart.

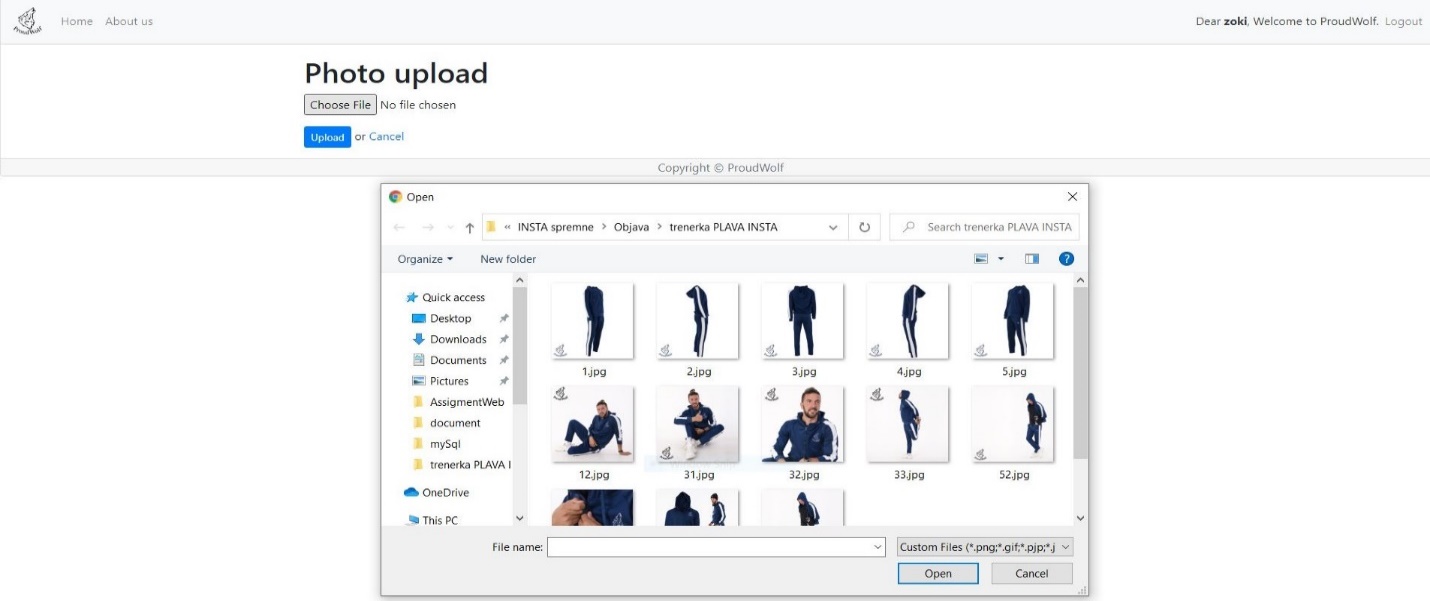
2.5. WARMENAGER



*List of products (warehouse)*

The home page after logging in to the warehouse manager is a list of all products. It can delete products only whose data are not in the purchase orders. And changing and adding new ones is done by creating a form, as with the user. We use the form tag from the springframework tag library, and we enable validation on the form with the help of annotations that we add to the fields of the model class.

When the product is added to the database, a new window opens with a button that takes us to the image upload form.

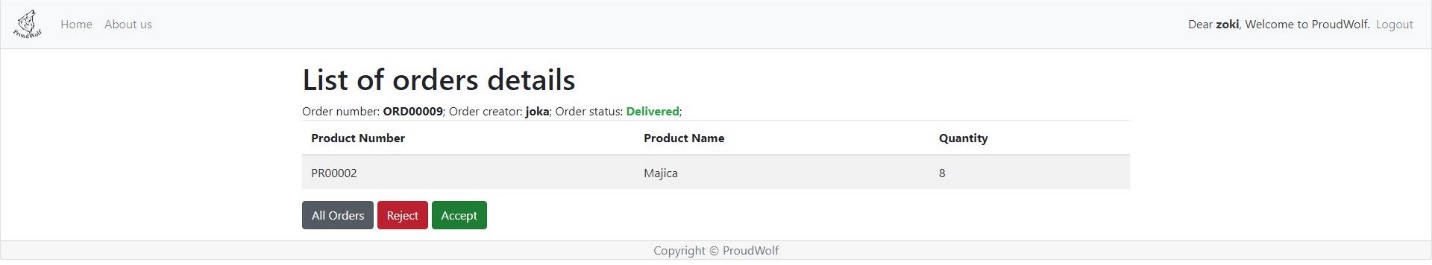


*Add an image*

Uploading is done by configuring the ***StandardServletMultipartResolver*** bin. With the standard implementation of the ***MultipartResolver*** interface, based on the ***Servlet 3.0*** app. The created ***FileBucket*** class is a wrapper. And its ***MultipartFile file*** attribute provided to us by ***Spring org.springframework.web.multipart.MultipartFile*** is a display of the loaded file received in the multipart request.

We save the uploaded image on the file system of the application itself in the ***static / photo*** folder, which is a temporary solution until a server is provided on which to save the images. While the name of the image, which consists of the name of the image of the shipper and the unique number of the product, we store in the database. This ensured a slow opening of other data from the database while the image was being loaded.

To validate that the user has actually selected an image of the desired format and size, we use the ***FileValidator*** class, which we created in the util package. Must be up to 5MB in size and format jpg, png or gif.



*Order details*

The warehouse manager bakes the ***Orders*** button, where all the created orders come from, then by clicking on the ***Details*** button you get the details of each individual order. On this page, he gets the opportunity to click the ***Reject*** button to reject the order and also print a message to the customer in the column description of the table orders - Order been rejected. Contact the warehouse. By clicking the ***Accept*** button, the order is processed automatically. If there is a required quantity of products, it is accepted and the quantities are removed from the warehouse, and a message is printed - The order has been accepted. And delivered. In case the sold quantity no longer exists, it is rejected and the quantities in the warehouse remain unchanged and it is printed from the message - Physical inspection determined that the item: T-shirt, was not in the required quantity. Please make your new purchase order. If the order has been processed, it cannot be processed again.