

# DAT076 Group 11 Report

## Introduction

Online shopping is a huge industry and with friends and family in that business we wanted to get a peek under the hood that is why we made an ecommerce website. Our application is a white label store where you can sell anything you want.

Github (private): <https://github.com/GustavAxelsson/DAT076-group-11>

Message Gustav Axelsson on slack for access.

## Use cases

1. **Login (Authentication):** You can log in as either a customer or an admin depending on which one you are you will have access to different features.
2. **Add a product:** If you are signed in as an admin you have the option of adding a new product to the store. By filling in the needed information: name, price, description and choosing a category (The category is created on the admin page). Your product will be added to the database and can be viewed in the product page.
3. **Create a sale and add products to it:** If you are signed in as an administrator you have the option of adding a sale on the site. By filling in the needed information: name and percentage the sale can be created. Once the sale is created the admin can add products to it, also on the admin page.
4. **Set and display the current sale:** Once sale(s) have been created and products added the admin can, on the admin page, choose a sale as the current sale this will display the products from that sale on the homepage of the app.
5. **View products and make a purchase:** The user adds products from the products page into the cart. These products can then be viewed in the cart page along with the current price of the order. If the user is pleased with the order they can choose to complete the purchase and the order will be added.
6. **View your orders:** if you are signed in as a customer you have the option of viewing your previous orders. The previous orders will display all the products that belong to each order.
7. **Upload picture to product:** When a product is added by the user, the user can upload an image to it on the admin page.

## User manual

Creating an account and logging in: In order to create an account the user can access the "register" button in the top right corner. a dialog will pop up and ask the user for the desired username and password. Once those are filled in another dialog will pop up where the user fills in the required personal details of the account. When the user has filled in the fields of

the second dialog and pressed “submit” it will automatically be logged in. When creating an account the default role for the user is “user” but by changing the role of the user to “admin” in the database the user can relog and upon logging in again it can access the admin pages.

Adding a product: The user in this case an admin can sign with an admin username and an admin password once the user has done that they have access to the admin page (the icon leading to “My pages”). On the admin page the user navigates to the “Add products” tab and can fill in the fields in “Add product”. They enter the name, price, description and choose a category (if one exists otherwise they can create one in the “Add new category” form) of that product and press the “Add” button in order to add the new product to the site.

Adding a sale with products: The user in this case an admin can sign in with an admin username and an admin password once the user has done that they have access to the admin page. On the admin page the user navigates to the “Add products” tab and can fill in the fields in “Add sale”. They enter the name, at what percentage the price reduction should be. Once that is done they can press the “Add” button and the sale is added to the database. To add products to the sale the user stays on the same page and under “Add products to sale” they can use the dropdowns to choose whichever product to add to whichever sale.

Set and view ongoing sale: After adding products, sales and adding products to sales the admin user can, when at the “Add products” tab on the admin page, set the current sale by choosing a sale in the dropdown under “Set current sale”. When the current sale is set the user can navigate to the homepage and view the products of the selected sale.

Making a purchase and viewing the order: The user starts at the homepage and by navigating to the product page they can start viewing the products. If the user wishes to add a product to the cart they do so by clicking the “Add to cart button”. Once the user is pleased with the products they navigate to the shopping cart page by clicking the shopping cart icon at the top. On the shopping cart view they can view their order and if logged in they can make a purchase by clicking the “Purchase” button. If the user wishes to view their previous order(s) they can navigate to “My pages” by clicking the icon at the top of the page. Once they have done that they can view their orders on the “My Orders” tab.

Upload picture to product: When logged in as an admin the user can navigate to “My pages” by clicking the icon in the top. Once there the user can add a picture to an existing product by going to the “Add products” tab. On that tab the user selects an existing product in the “Add image to product” form, presses “Select image”, browses the it’s local storage and selects an image. Once the image is selected the user can press the “Upload button and the image is uploaded.

Edit users: When logged in as an admin the user can change the role of other users by navigating to “My pages” and on the tab “Edit Users” the user can change to the role of any user.

# Design

## Backend:

The backend is written in JakartaEE. To make the database queries more easy to read we are using QueryDSL in the Data Access Object interface. The application is also using Lombok to generate getters, setters and constructors to avoid unnecessary boiler plate code.

To connect the frontend and backend a RESTful API is implemented, more specifically JAX-RS. To prevent unauthorized users from accessing restricted endpoints the application uses the RolesAllowed interface combined with Eclipse Microprofiles. To store passwords the application hashes the passwords with a jBCrypt algorithm provided by mindrot. The salt is generated with a logarithmic round factor of 10 which corresponds to 1024 rounds.

To create tokens we are using Nimbusds jose and jwt-libraries. For testing the backend we use JUnit together with arquillian.

## Database:

The application uses a remote Apache Derby database and EclipseLink as JPA provider.

## Frontend:

The frontend part of the application is written in Angular 11 which is based on TypeScript 4. The application mainly uses Angular material components for styling. For state keeping and stream functionality RxJS is used. To prevent unauthorized clients accessing restricted views the application uses so-called Guards which implements the canActivate interface provided by Angular. This means that if an unauthorized user tries to access a restricted url with the wrong user credentials provided by the JWT, the user will be redirected to the home screen.

The two restricted parts of the application are the admin views, and specific user views. These views and subviews are modularized into their own separate modules and lazy loaded which means that they are only loaded when an authorized user tries to access them. To decode tokens the application uses one of the most used libraries jwt-decode. The application also uses the library ngx-cookie-service to store cookies. Cookies are used for keeping the shopping cart items if the user somehow loses the connection or wants to store the shopping cart items for another time. The frontend application also contains all the dependencies which come from generating a new angular project.

## Application flow

### **Register a user:**

In the top right corner, in the header there is a register button. Clicking this button will prompt the user with a dialog, asking the user to enter desired username and password. When the credentials are filled in and the user presses register which initiates a rest call to authentication service.

The authentication service then validates that the username is unique i.e. no previous user with the same username is stored in the database. If there is, a bad request response is sent


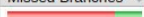








to the user. If not, the authentication service will proceed registering the user by hashing the password with BCrypt algorithm and then store the user credentials in the database. When this is done, the service starts to generate a user token (Jason Web Token). The token is generated by using the private key, together with the user id, a generated token id, and the user's authorization role. We use two authorization roles in this application *user* and *admin*, all new users are set as *user*. The service then sends the token back to the client indicating that the user registration was successful. When the token is received on the client side the token is stored to the browsers local storage area. To remember to send the token in every future communication with the server, the application uses a HTTPInterceptor which injects the token in all http headers.

When this is done, a new view is shown for the user, asking the user to fill in customer information like email, personal number and billing address etc. When this is filled in and the user presses register a new request is called to the server, but this time to the user service (setCustomerOnUser method). The user service extracts the token to identify the user, gets the user id from the token and stores the customer information and responds to the client. If everything was ok, the clients authentication service will submit on the auth user stream that the client is logged in. The header of the application subscribes on this stream and will now show that the user is logged in by showing a field "logged in as: <username>" and a logout button.

## Package diagram

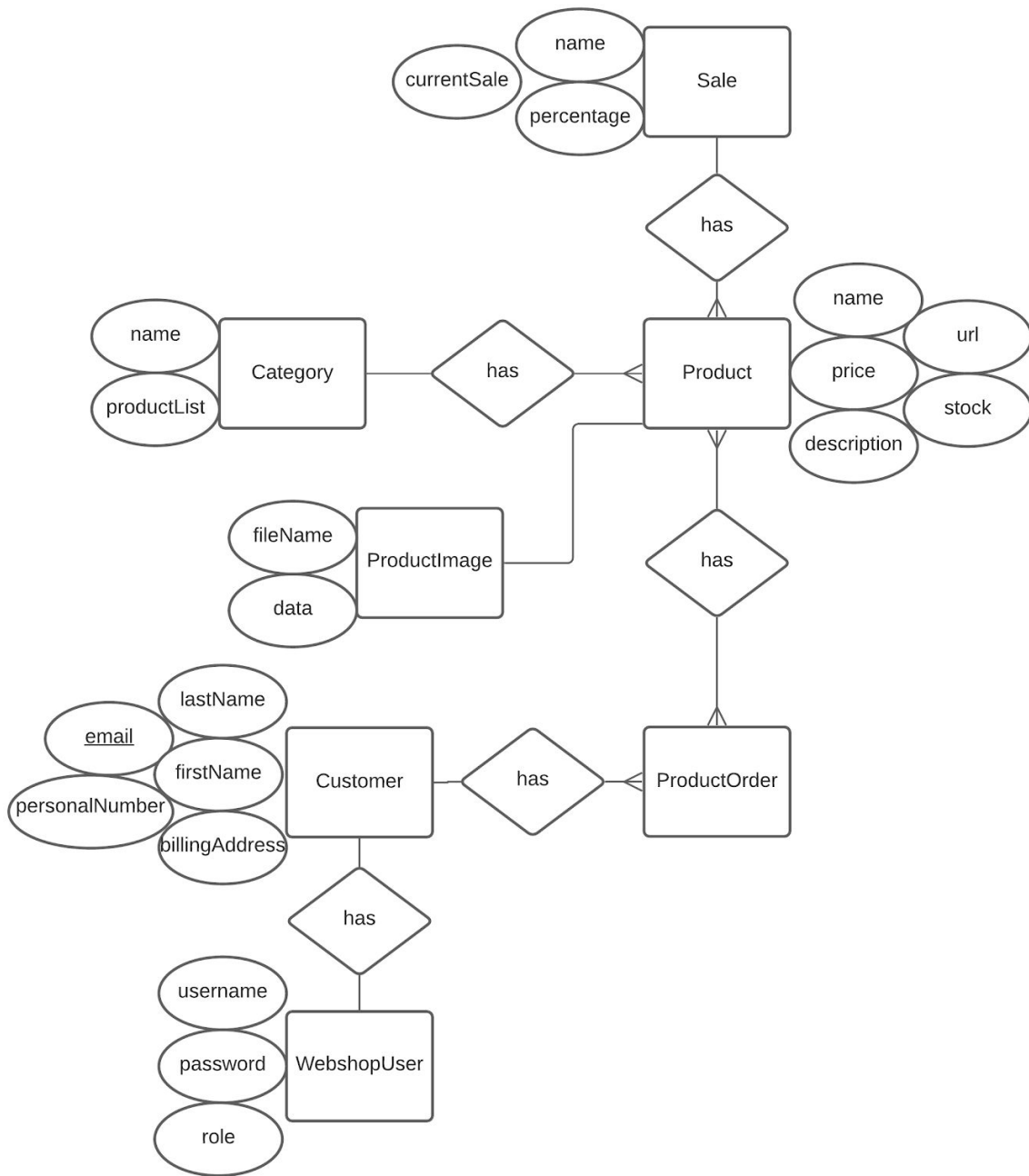
## Coverage report

database\$Manual\_Container\_Configuration\_\_All\_in\_database.exec

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
restApi.model.entity		45%		20%	216 326	62 229	76 177	1 15
restApi.resources		0%		0%	63 63	206 206	37 37	9 9
restApi.model.external.model		0%		0%	21 21	5 5	10 10	1 1
restApi.model.dao		91%		58%	7 57	16 189	2 51	0 8
model.dao		100%		n/a	0 52	0 312	0 52	0 8
Total	2,474 of 5,940	58%	317 of 384	17%	307 519	289 941	125 327	11 41

The code coverage report is located at: database/jacoco-sessions.html

## Model diagram



## Responsibilities

The token generation is inspired by the following repos and sites:

<https://github.com/AdamBien/jwtенizr/>

<https://github.com/tuxtor/microjwt-provider>

<https://dzone.com/articles/a-simple-microprofile-jwt-token-provider-with-paya>

Angular Auth Guards:

[https://medium.com/@ryanchenkie\\_40935/angular-authentication-using-route-guards-bf7a4ca13ae3](https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3)

HTTPInjector:

<https://medium.com/angular-shots/shot-3-how-to-add-http-headers-to-every-request-in-angular-fab3d10edc26>

Up and downloading images:

<https://rieckpil.de/howto-up-and-download-files-with-java-ee-and-web-components/>