D0018E Lab Assignment: An Internet e-commerce site using an SQL database

by
Group 1
Hugo Wangler — 970818-1194
hugwan-6@student.ltu.se

Martin Terneborg — 960902-9138
termar-5@student.ltu.se

Aron Strandberg — 940902-4214
arostr-5@student.ltu.se

Robin Vernström Persson — 941129-5414
robper-6@student.ltu.se

December 30, 2018

Contents

1 Executive Summary

This report documents the group's solution to create an e-commerce website using a relational database. The assignment was to create an e-commerce website with a certain set of functionalities and it had to use a relational database to store and retrieve information. The purpose of this assignment has been to gain knowledge in fundamental database design, the query language, pros and cons of a relational database and developing a website.

The e-commerce website is hosted on a virtual machine provided by the student section LUDD[1] and the webserver is running on Apache2. The database is a MariaDB database and the language for application programming is PHP. In addition JavaScript was used to make the website dynamic.

The website supports user accounts, to register and login, searching for products, filtering products by categories, adding and removing products to a user unique shopping cart and storing completed transactions in user unique order history. Admin accounts have also been implemented with specific features such as modifying-, archiving- and adding products, categories, viewing customer orders and changing their status.

As a part of the course requirements the project was executed with a agile work process using the SCRUM model.

In the end the assignment was successfully completed, that is, a functional website with a relational database was implemented. Certain features that the group would like to implement and that would improve the website still exist, but have been chosen to not be implemented in favor of finishing the assignment.

[1] https://ludd.ltu.se

2 System design

2.1 User stories

2.1.1 Roles

For this website to serve as an e-commerce website two different users/roles were identified. The first user is the customer who will use the website to buy products and the second one is the administrator who administers the website.

2.1.2 Customer

The key requirements identified for the customer is

- Buy products from the website
- Register on the website
- Log- in and out from the website
- View all their orders
- Rate and comment on bought (and delivered) products
- View and change their shopping cart

2.1.3 Administrator

The key requirements identified for the administrator is

- Add new products to the website
- Modify existing products
- Stop selling certain products by archiving them
- Re-stock products which has been sold out
- View all the orders made by customers
- Change the status of orders
- Log- in and out as an administrator

2.2 Use-cases

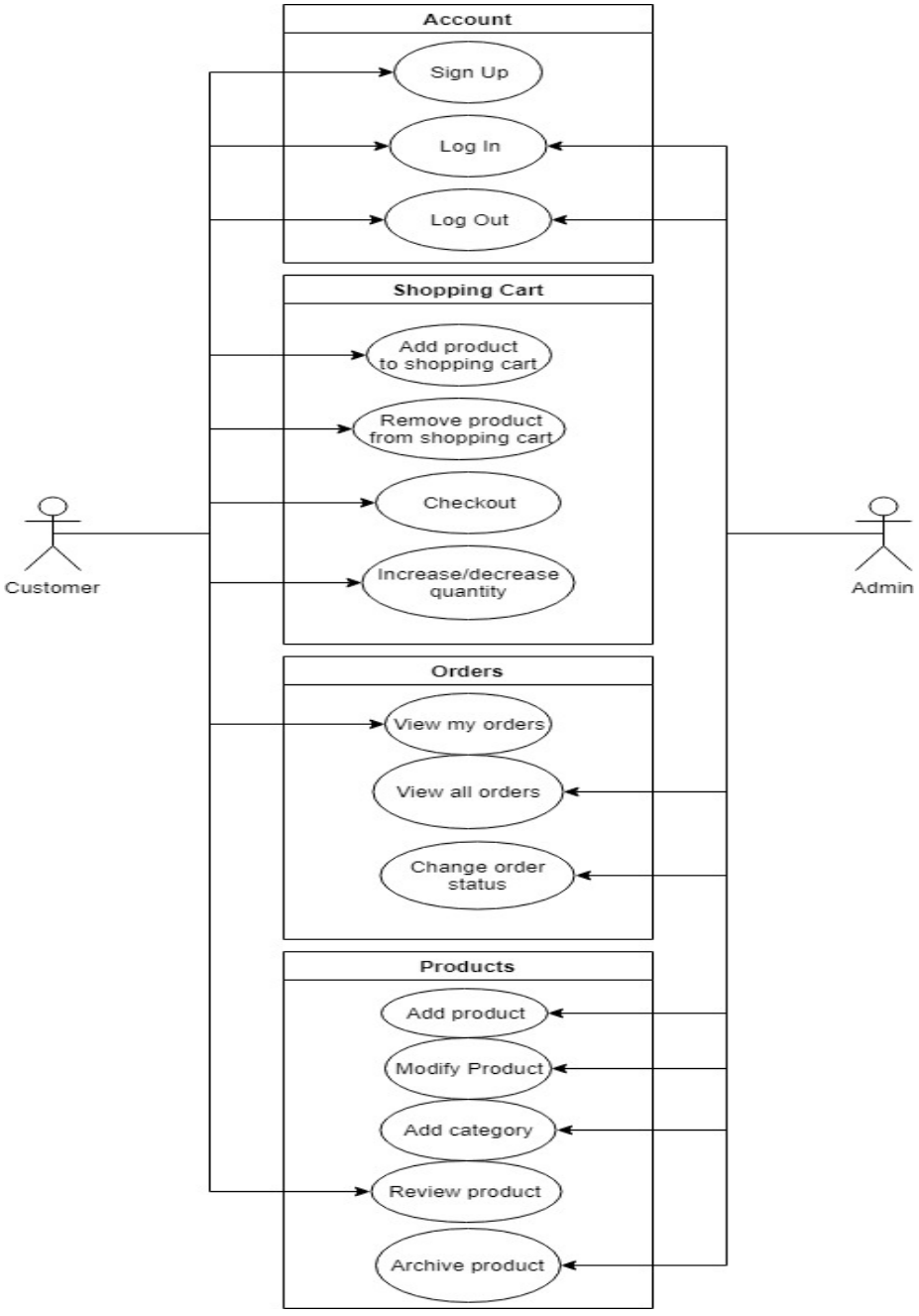

Figure 1: Use-case diagram for the actors

3 System Architecture & Implementation

The setup of this project contained a lot of options concerning the system architecture. The web server Apache was chosen and is accompanied by the database MariaDB. The reasoning behind this was mainly that they both felt well established and are open source. To host the server the Dust Cloud service, which is a virtual machine running Ubuntu 16.04 with a public IP address, provided by LUDD was used. No framework e.g. Bootstrap has been used but the basic template, HTML5 Bones (Devlin 2018) were used in the beginning of the project to streamline development. The template has been sequentially pulled out of the project during the entire implementation process until it was completely removed.

To communicate between the web server and the database PHP 7.0 was chosen as application programming language since there is a lot of documentation and examples of using this language. The lab group also had no prior experience of using PHP and thus this provided a possibility to gain new experience. In order to make the website interactive JavaScript was used. Finally, to enhance the user experience, the client side programming in JavaScript was partly implemented using AJAX (asynchronous JavaScript and XML), which is a technique for creating fast and dynamic web pages by making parts of it update without reloading the whole page (w3schools n.d).

3.1 Database

The relational database schema can be viewed in Figure 4. As can be seen there is a common pattern: most tables are identified using an id. This is necessary as the other attributes in these tables can not uniquely identify entries in the table. Note that this is not true for the tables *Users* and *Products* which could be uniquely identified by their email and product name respectively, however, this has been purposely avoided as this would make querying the database more tedious and also increase the required storage of the database since every user and product would have to contain an entire string rather than just an id (a number). Also note that the tables *Categories* and *OrderStatuses* do not use an id, but instead are identified by their name. The reason for this is that, while suffering from the same storage increase as previously mentioned for *Users* and *Products*, querying the database for order statuses and categories using indices would require knowing what arbitrary index has been assigned to that status and category, and if for some reason that index were to change it could possibly break functionalities on the website.

Apart from the automatic indices created of the primary- and foreign keys in all the tables, two additional indices were added. The reasoning behind adding these two indices were that the attributes were used in the WHERE clause of a lot of queries and they were not subject to frequent changes. Thus the performance loss of frequently needing to update the index file were avoided and

queries could be sped up.

One of the attributes indexed were *email* in the *Users* table which was implemented as a UNIQUE index. This attribute is needed in all queries regarding user log in to confirm that the customer is registered on the website. Since every customer only registers to the website once, the *email* will never change and the index file will not need updating once it has been inserted. The second indexed attribute was *archived* in the *Products* table which was implemented as a plain INDEX. The reasoning behind this is that the attribute is extensively used in most of the queries regarding products since most of the time either non-archived or archived products needs to be shown on the website. Also, the shelf life of a product can normally be assumed to be long enough for the index file not needing frequent updating.

In the following subsections some reasoning behind definitions and considerations of certain functionalities are discussed.

3.1.1 Users

To achieve a close representation of a real e-commerce website user sign up and user log in was implemented. This was made possible by using PHP sessions and storing user information in the database table *Users*. Many functions on the website are not available if you are not logged in. The primary key *id* in the *Users* table is used to identify which user is performing actions and the *role_id* attribute is used to decide whether they are authorized to perform them. It is also a foreign key from the *Orders* and *ShoppingCartLines* tables.

When a user signs up they have to submit their first name, last name, email and password. To log in they use the email and password they selected when they signed up. Users remain logged in until they decide to log out.

As previously mentioned in section 2 two kinds of users exist, admin and customer. Admins cannot order any products but they have a lot of administrative functionality available to them, for example adding-, modifying- and archiving products, adding categories and changing the status of placed orders.

3.1.2 Products

The *Products* table has the attributes *id, name, price, stock, img_url, cat_name, archived*. The *id* is the primary key, and *name* is what the search functionality on the site uses to search for products and has the constraint of being unique. The site also lets the user filter products by categories using the *cat_name* attribute. In case an item is to no longer be sold an admin can mark a product as archived using the *archived*. The attribute *img_url* is the url to the image file located on either the website or some other site.

3.1.3 Shopping cart

One of the major parts in this lab was to implement the functionality of a shopping cart for the customers. This was achieved, in the database, by creating the table *ShoppingCartLines* which represents a product that exists in the shopping cart. In Figure 4 the definition of *ShoppingCartLines* can be seen. A shopping cart was identified as being a list of products with varying quantity added to the shopping cart by the customer. Thus the attributes of *ShoppingCartLines* is *user_id*, *product_id* and *quantity*. In order to prohibit a user from placing an item in their shopping cart during a possible sale and not placing the order until after the sale (in effect extending the sale), the database does not store any price attribute for the items in the shopping cart and instead the prices are checked when placing the order. Since every user only has one shopping cart line for each product in their shopping cart the primary key was identified as $(user\_id, product\_id)$. Note that a customer can only have one shopping cart, unlike orders, therefore the cardinality between the customer and shopping cart is one-to-one. Thus the shopping cart lines do not have to refer to an intermediate table representing the shopping cart (like the case with orders), but can instead refer directly to the user, which reduces the total amount of tables needed.

This provided the customer with the option of adding several products to the shopping cart instead of only being able to order one item at a time. A shopping cart page for the customer was also implemented where the current shopping cart is displayed and options for increasing and decreasing the quantity of each product, as well as removing all the products of one type, is available. A button for checking out (placing) the order was also added. This button transfers the customers current shopping cart into an order.

3.1.4 Orders

Similar to the table *ShoppingCartLines*, entries in *OrderLines* represent an item in the order and have the attributes, *user_id*, *product_id*, *quantity*. Unlike the shopping cart, the user can have multiple orders. Thus each order line entry needs to know which order it is referring to. Since every order line refers to exactly one order but every order contains one or more order line the cardinality between *Orders* and *OrderLines* is one-to-many. When a user "checks out" their shopping cart it is turned into a new order and a lot of queries to the database have to be made:

1. A new row has to be inserted into the *Orders* table
2. Get the price, stock, name of the product and whether or not the product is archived from the *Products* table
3. Delete the shopping cart line from the *ShoppingCartLines* table

4. Insert a new order line into the *OrderLines* table referring to the newly created order

5. Update the stock of the product

6. Repeat 2-5 for every item in the shopping cart

If any of the these queries fail the checkout should fail and the shopping cart should not be processed, thus none of the other queries can have an effect on the database, even if they were successful. This achieved is by using a transaction that is started before any of the above queries and rolled back (the database resets to the state it was in before the transaction started) in case any failure occurs. The transaction also makes sure that two users can not checkout their shopping cart simultaneously, which could cause a faulty stock value if a client reads the stock value in step 2 before another user updates the stock in step 5.

Finally, after a successful checkout the order can be viewed by the customer in the *My Orders* page created. Here the customer can view each order placed and see what products each order contained along with the sum of the order.

3.1.5 Grading and comments

Every grade and comment is stored in the *Reviews* table. Each product can be rated with a grade between 1 and 5. Only after the user have rated the product they can comment on the product. The reason for this restriction was to simplify implementation and increase modularity of the review functionalities. It was also done to ease the user experience i.e. to not force the user to comment. There are separate JavaScript files and PHP files for adding a grade and adding a comment. Therefore the grading functionality can be used on a different page without the comment functionality and vice versa, hence modularity. Since the JavaScript files have been written with AJAX there exists a possibility to make the review page fully asynchronous, i.e. if you for example grade a product the grade will be shown instantaneously without having to reload to page.

One important restriction in the grading and commenting of products is that only users who have bought and received the product, i.e. the order status of the product is *delivered* or *returned*, can review the product. The implementation of this restriction was challenging since nested select statement had to be used. The challenge was that every user with their unique foreign key *user_id* could have had one or more *Orders*. Every order is referred to with the foreign key *user_id* and the order also stores the status of the order. The nested statement is introduced since every order can have one or more *OrderLines* with the primary keys *user_id* and *product_id*. Other restrictions are that the Admin can not review the product and only view the reviews of the product. However this restriction is not implemented in the database instead the admin is redirected to a different product page. A visitor will have the possibility to view the add rating and comment field as well as other users reviews. The visitor can not add

any rating or comments but they are introduced to the possibility to review a product. The reason is to recognize that there are other users who have done these reviews and that the user can perform a review if they register and buy a product.

4 E-R Diagram

Figure 4 depicts the schema for the websites database generated using the reverse engineering tool in MySQL Workbench. Note that in the tables *ShoppingCartLines* and *OrderLines* the primary key consists of two attributes. For *ShoppingCartLines* the primary key is *(user_id, product_id)* and for *OrderLines* the primary key is *(order_id, product_id)*. This will uniquely identify each row in the table without needing a separate key and thus no redundant data is stored in the tables.

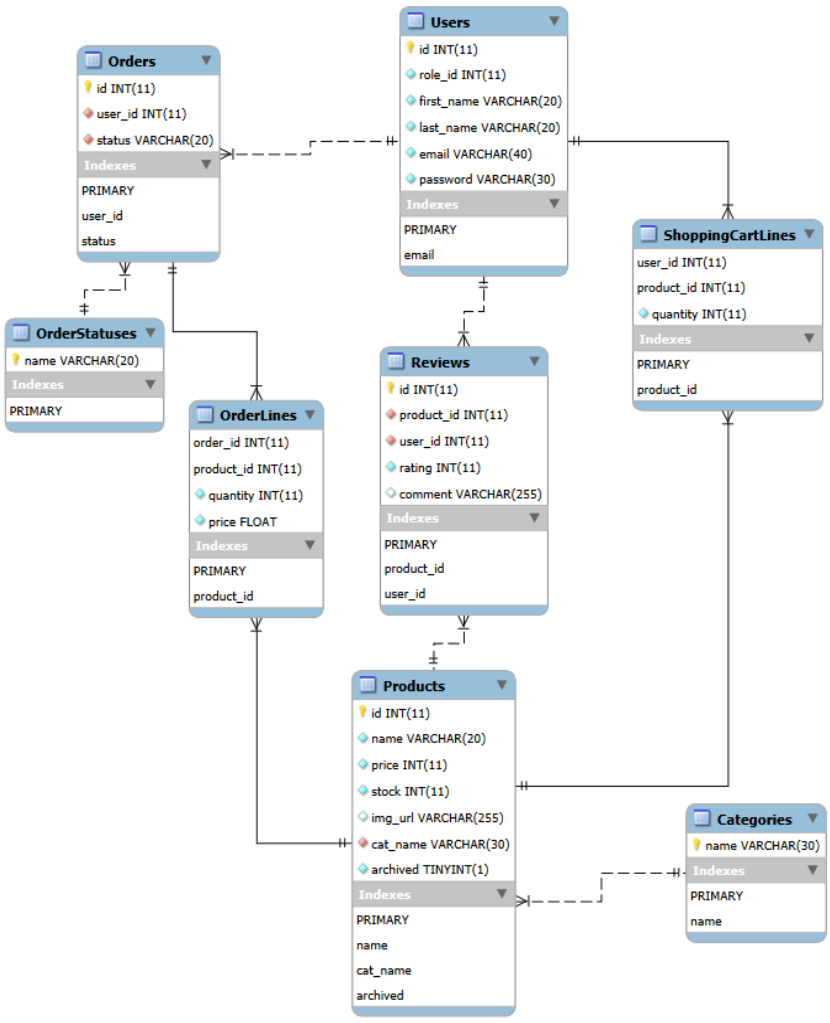

Figure 2: E-R diagram of the entities and relationships in the database.

5 Backlog

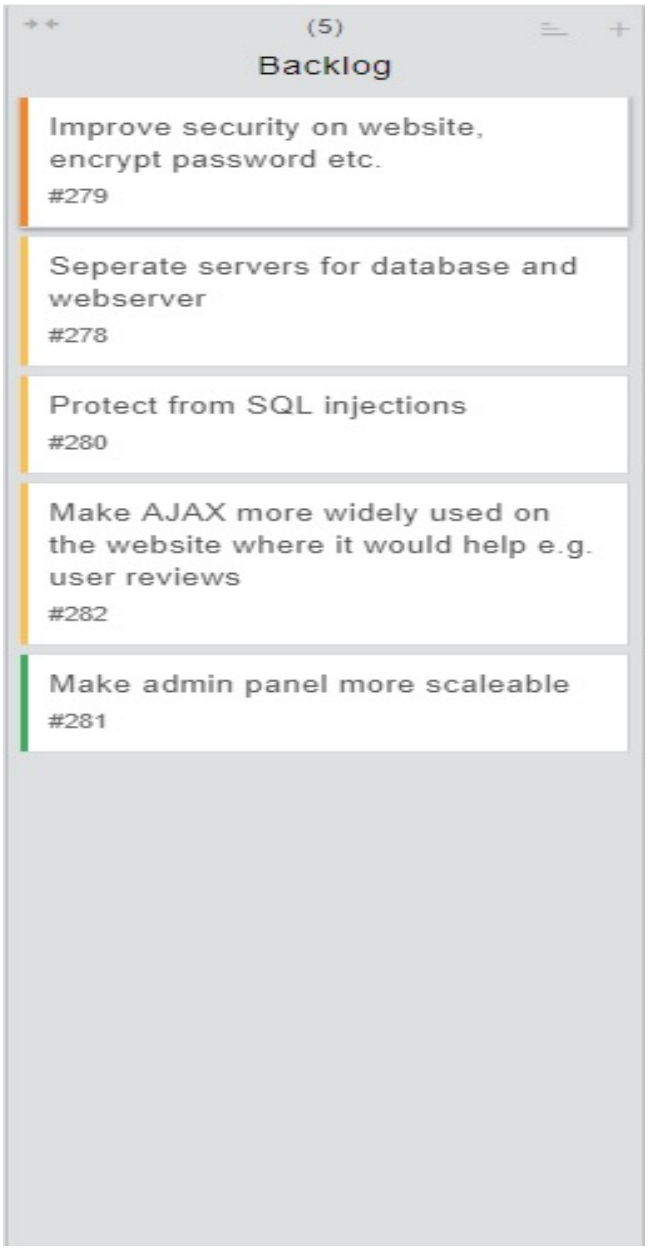

Figure 3: Backlog after sprint 3

6 Test cases

6.1 Customer test cases

6.1.1 Account creation and product ordering

This test case consists of creating a user, logging in as said user, adding a product to the shopping cart and checking out the shopping cart and finally viewing your order history.

On the main page of the website,

- Start by creating a user. Click on Sign Up in the upper right corner.
- Fill all fields with optional input and press Sign Up.
- After being redirected to the main page, press Log In.
- Enter the user information you chose and press Log In.
- After being redirected to the main page, order any products you wish by clicking Add to Cart on selected products.
- When finished, press Shopping Cart and proceed to check out.
- When checked out, press My Orders and then press the order id that refers to the transaction you just completed.
- View your order and observe that it is correct.

6.1.2 Comment and rating

This test case consists of logging in as a customer and trying to comment or rate a product that you have not previously ordered. Then order that product through the new product pages. Then logging in as admin and changing the status of the order to delivered. When the status has been changed log in as the same customer and try to comment and rate the product again.

Preparation needed to conduct this test:

- A user account.
- Several comments/ratings on the product "Rocket".

On the main page of the website,

- Press Log In and proceed to log in as a customer.
- Click on any product other than "Rocket".
- Try to rate the product by pressing an optional star.

- Try to place a comment by writing optional text into the text box and then press Submit.
- Order the product by pressing Add To Cart.
- Proceed to the shopping cart by pressing Shopping Cart and place the order by pressing Checkout.
- Go to the product page and try to rate and comment the product (it should fail)
- Press Log Out then press Log In. Log in as admin.
- Go to the Admin Panel by pressing Admin Panel.
- Change the status of the order you just placed as a customer by selecting "Delivered" from the drop down menu and then pressing Set Status.
- Press Log Out and then log back in as a customer.
- Press My Orders and observe the status of the previously placed order.
- Press the item that you ordered and try to rate the product and submit a comment.
- Observe your comment and rating on the right side of the product page.
- Search for the product "Rocket" by typing a part of the word into the search bar. Press the magnifying glass icon.
- Go to the product page for "Rocket". Observe past comments and ratings on the right side of the product page.

6.2 Admin test cases

6.2.1 Admin management

This test consists of logging in as admin and viewing and testing admin functionality.

Preparation needed to conduct this test:

- A number of orders from various users completed.

On the main page of the website,

- Press Log In and proceed to log in as admin.
- Once redirected to the main page click Admin panel.
- Try to modify a existing product with new values (string, int, int, string). Click Modify.

- Go back to the main page and observe the changes.
- Click Admin panel. Add a new product with optional values (string, int, int, string, string). Click Submit.
- Go back to the main page and observe the changes.
- Click Admin panel. Create a new category and modify a product to use this new category.
- Observe the changes with the modified product by selecting the newly added category from the drop-down button Categories and observe that the product is listed.
- Click Admin panel. Modify the status of an order and then click on the order id to view that order.
- Observe the contents of the order and manually check the database to see that it is correct.

7 System limitations

One major flaw that the website currently has is security. One example of this is that a user can not see the "Admin Panel" button but if they access the script from the URL they are allowed to make changes only admins should be able to make. Another potential flaw is performance, which could suffer as the database grows. The performance on a large database has not been tested.

Security and performance of the entire system is limited to the available development time and given resources, and was therefore not a focus in this project. The main server security and performance concerns identified are

- The usage of the given server environment which is the "LUDD virtual machine". In which both the web server and database are running on the same machine which brings security flaws, since no separation of front and back-end can be implemented.
- Because the server runs on a virtual machine there will be some amount of performance loss, which could be prevented by running the server on an operating system and hardware. Also the virtual machine consists of
  - 2 GB RAM memory
  - dual core Intel Xeon E5-2678 v3 CPU with a clock speed of 2.5GHz
  - 18 GB of storage

  These components will limit the performance of the website and the amount of data that can be stored in the database. Also, the CPU could easily be a bottleneck when handling a lot of simultaneous users.

8 Discussion

While working on the project some of the major problems were to learn PHP from no previous experience, setting up a server with a database, dealing with different kinds of users and using AJAX. This became easier with time as the group became more used to the system and the environment as a whole.

Since the group adapted an agile working process using the SCRUM model, via the online SCRUM board Zube[2], testing all the different features implemented from the backlog was important. The approach the group had to testing these were proactive in a way where someone developed a feature, tested it themselves and then requested final testing from the other group members. If the group members were satisfied and the implementation seemed to be working it was added to the main build of the project. This led to fewer bugs making it onto the main build of the project. A flaw in the development was that while developing, all testing were done locally on the group members's personal machines to avoid conflicts, which means that the development environment did not necessarily match every aspect of the deployed environment. While no problems due to this flaw occurred during development, it easily could have.

If the development of the website were to continue there are still improvements left to be completed in future work. One of the possibilities for future improvements of the website, as noted in the backlog in Figure 3, is to make use of AJAX in more parts of the website. The process of implementing functionality using AJAX is very slow and was thus not used in every suitable part of the website. This is because the web page requires predefined sections for all of the data that needs to be updated which requires a lot of consideration, in contrast to just printing the new data directly from the PHP scripts.

The administration page (accessed by pressing the "Admin Panel" button while logged in as an admin) also needs some improvements. The major reason behind this is that the page it is currently not scalable in regards to the number of products on the website. The cause to this is that the product list, in which the admin can modify products, continues endlessly and forces the admin to scroll through the entire list before other admin functionality such as adding category can be used. This could for example be solved by redirecting the admin, from the admin panel, to different pages dependent on what functionality they wants to use.

Overall, the assignment of creating this project has given the developers knowledge in database design, good understanding of the query language and application programming in PHP, javascript, CSS and HTML. The pros and cons of using a SQL database when creating an e-commerce website versus a document oriented database as NoSQL was also considered. One disadvantage

[2]https://zube.io

could be performance when it comes to handling big amounts of data. Thus this could become a problem if the website were to be populated with a lot of registered customers and products. An advantage of a SQL database is its implementation of transactions, which guarantees all of the ACID (Atomicity, Consistency, Isolation, Durability) properties. An e-commerce website does not want to lose profit from customer dissatisfaction as a result of loss of customer data or database/infrastructure failures. Therefore the assignment has also highlighted the importance of avoiding a database failure or infringement, and thus also the importance of SQL databases.

References

Devlin, I. (2018). *Back to basics*. Retrieved November 15, 2018, from https://html5bones.com

w3schools. (n.d). *AJAX Introduction*. Retrieved December 13, 2018, from https://www.w3schools.com/php/php_ajax_intro.asp

Appendix A. Project access

To view the entire project go to the projects GitHub Repository[3]. Note that the repository is private so the reader will need access to view the repository. The reader should be able to setup the web server and website on their own machine by following the instructions in the *README.md* file in the repository. To create the database use the php script dbSetup.php located in the folder website/php/db.

To view the deployed website, go to 130.240.202.225. In order to log in as admin enter admin as both password and user name.

[3]https://github.com/dynematic/D0018E_Group1

Appendix B. Sprint progress

This appendix covers the groups progress made during each of the three sprints it took to complete the assignment.

Sprint 1

During sprint 1 we made it through 100% of our backlog, depending on how you look at it. The feature "Implement customer ordering functionality" could be considered as completed in the scope of the first sprint. It is however such a broad statement that it accounts for functions we have yet to develop.

We also managed to implement a functioning server, database and basic website. We made PHP-scripts to create a database, create the appropriate tables, filling them with some products and then fetching them to display on the website.

The website is very basic, containing a section for displaying products stored in the database and placing an order on them. What we currently display is the products name, prize, stock and rating. User functionality and roles have yet to be implemented.

Sprint 2

As of sprint 2 the website has had major re-designs and functionality implementations. The navigation menu has been extended with buttons for *admin panel*, *log -in/out*, *sign-up*, *shopping-cart* and *my orders*. The buttons will now be visible or hidden depending on if the user is a none-customer, customer or administrator.

A shopping cart has also been fully implemented and the customer can now checkout i.e. buy the product and view their order. The admin can via the admin panel add products, modify products, view all orders and add categories. The database has been re-worked and fully implemented. Each product can now have a category and the pricing routine has been changed to allow for eventual sales. Development functionalities for handling the database has been implemented for improved development environment.

At the end of sprint 2 the mechanism for handling the product stocks was redone to only subtract from the stock when the user checked out their shopping cart. This also required the implementation of transactions when querying the database at checkout.

Other mentionable functionalities is an initialization script for the header, re-directions to home, hidden passwords in entry-fields and disabled add to cart

functionality for the administrator.

Preparations for sprint 3 have been made. In the beginning of sprint 3 a product page will be implemented that will display the product, give the customer the possibility to rate and comment on the product and view other users' comments. Overall testing, improved code structure and further design implementations are planned for sprint 3. As previously mentioned in this report, the group hopes to finish the website by sprint 3.

Sprint 3

During this sprint we finished our lab. The major functionality that was added this sprint was searching for products, product pages and general improvement of the database. Orders also had "Status" associated with them to be able to discern the progress of a placed order. Admins have the ability to change these statuses through the admin panel. Comments have also been added to the code and general clean up of the code has been performed.

The search function takes input from a user and compares that input to the names of products and displays any matches on the main page. To assure qualitative searches it compares the input to the entire product name and accepts matches on part of the product name. Meaning if a user searches for "ock" in the hopes of finding the product "Rocket" they can.

The biggest addition to this sprint was definitely product pages. Each product now has a unique page that users can utilize to order, rate and comment on the product. They can also see previous comments made by other users. A user can only comment or rate a product that they have previously ordered and had delivered.

The database has been improved by adding indices where we saw fit. This helps speed up the time it takes to query the database. As previously mentioned we also added "Status" to orders so users have the ability to for example cancel the order by contacting an admin.

Appendix C. Website interface

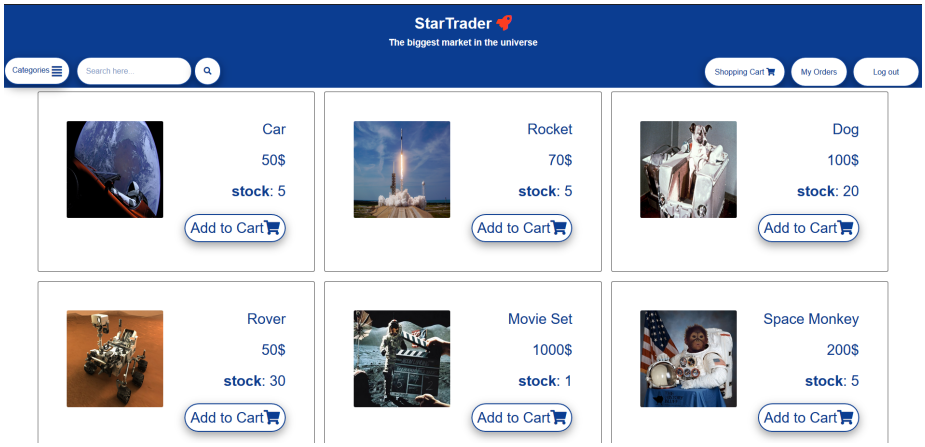

Figure 4: Print screen of the websites front page.