Marketplace

By: Liam Mattsson, Marcus Phu, Rasmus Standar
Group: 11
Date: 2023-03-14

1. Introduction

The project is created by Marcus Phu, Rasmus Standar and Liam Mattsson. The project is an assignment for the course DAT076: Web Applications and has been under the supervision of Robin Adams.

This project involves creating a marketplace web application using React.js and Bootstrap for the frontend as well as Node.js and Express.js for the backend. The framework used is the Client-Server framework.

The main functionality of the application are:

Put up sell orders onto the marketplace.
Update or delete their own sell orders.
Buy other users products that don't have a buyer.

To complement these functionalities we have also added the secondary functionality of:

Creating an account.
Log into an account.
Adding products into a cart.
Filtering products based on category.

The project is available with installing instructions on our GitHub page:
[YamadaTTV/DAT076-Web-Applications]

2. Use cases

The projects use cases that have been implemented are:

1. Add user - Let the user register on the homepage
2. Add product - Add a product to the browse page
3. Buy product - Set the product's buyer to corresponding user_id
4. Update product - Let the seller (user) update an ongoing listing
5. Delete product - Let the seller (user) delete an ongoing listing
6. Login user - Let the user login and see the marketplace
7. Filter function - Let the user filter between items in the marketplace

These use cases were implemented because we thought they are the most fundamental functionality for a marketplace application.

3. User manual

To run the application, the user has to clone the github-repo, enter the folder /server and then execute the command “npm run dev”, this will start a server on “localhost:8080”. To access the web application the user then needs to open their browser and open the web-page “localhost:8080”.

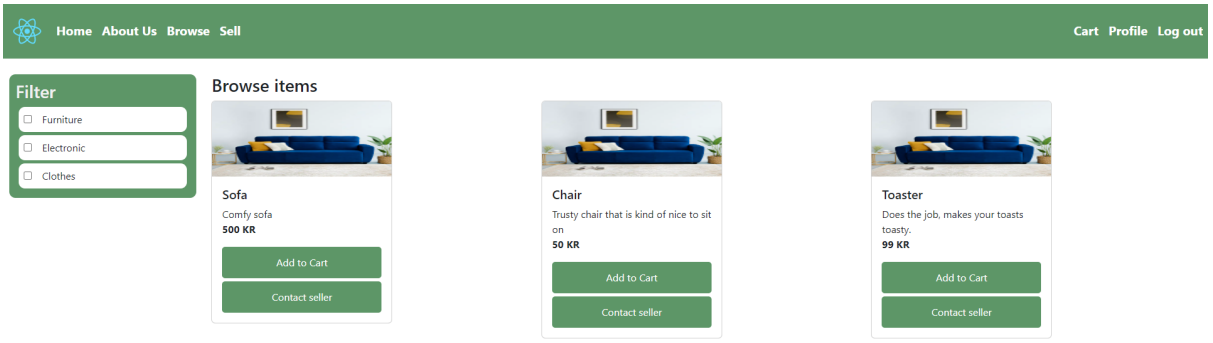
Once the user has opened the web application in their browser they will be greeted with this screen:

The first step for a new user would be to press the register button, this will take the user to the register page.

On the register page the user will have to enter a username, an email and a password. The user will have to come up with a unique username and email and a password that fulfill the requirements. After pressing the register (given that the username, email and password are approved) the user will be returned to the start-page where they now can proceed to log in.

After logging in, the product page (shown below) will be displayed which gives the user plenty of options. On the left side of the screen the user has the possibility to filter the available products by category, where pressing a category will filter so only products with the selected category are displayed. In the header there are clickable buttons that allow the user to access different pages and functions. The most important fields in the header are the cart, profile and sell options. All of these three options are only visible when logged in, and have the following functionalities:

- **Sell:** opens a pop-up (a modal) that allows the user to post a product listing. For the user to post a listing they will have to enter a product title, a product description, choose a category and enter a price for the product.
- **Cart:** Will be displayed as soon as a user presses add to cart on an active listing. By pressing this button the user will get to a page that shows all items the user has currently added to their cart. The user then has the option to remove items from their cart or to press the buy button. If they press the buy button all the items in the cart will be marked as bought and the user will be sent to their profile page.
- **Profile:** By pressing this button the user will be transferred to their profile page. On this page the user can see all the listings that they have posted. They will also be able to update their current listings or to remove the listing altogether. On this page the user will also be able to see a history of all the products that they have bought.



4. Design- Describe the technical construction of your app, including all libraries, frameworks and other technologies used. Include a specification of the API for the backend, i.e. what requests it accepts, and what responses it sends for valid and invalid requests.

Technical construction:

Our web application consists of two big components: The frontend client side and the backend server side. The client side provides that part of the application that will interact with the user and provide a user interface. The client side is built using React.js together with React Bootstrap. The server side is built using Node.js and Express.js. Instead of writing code in regular JavaScript, we have written it in TypeScript. TypeScript is the same as JavaScript but with additional programming rules (type checking) that you can add to write better code and enable easier debugging. The server side consists of 3 main entities, User, Product and Cart. The user is identified by an id number, username, email and password. The Product is identified by a key, name, description, category, price as well as a seller id number. The Cart is identified by a user and an item in the cart.

Here is a short list of used frameworks and libraries:

- React
    - Our main library used to create a responsive frontend.
- Node.js
    - Used for the server as a runtime environment.
- Express
    - Back-end application framework used for building our server.
- Jest
    - Testing framework used for testing applications that run primarily javascript. Used for all our testing.

- Bootstrap
    - A toolkit/library containing components that can be used to make web applications quicker to build and better looking.
- Model, Router, Service
    - The design model used for the server was: Model, Router, Service. This means that code was divided into three layers. The code in the model described the format of the data, in our case for User, Product and Cart. Code in the service layer described what services could be performed linked closely to the Model-layer. Finally the router layer handles requests and responses to and from the server. Router layer code calls the appropriate methods in the service layer and returns responses based on the results of the service layer.

Specification of the backend's API:
This is the specification of the backend's API.
The status code is set following the standard for HTTP responses.

Request to /user:
GET - Returns a list of all users.

Session of the request:
{
"user" ? : User
}
Status codes:
200: List of all users returned
500: Error occurred

POST - Creates a new user.

Body of request should be:
{
"id": number,
"username": string,
"email": string,
"password": string
}
Session of the request:
{
"user" ? : User
}
Status codes:
201: User created
281: User already exists
400: Bad PUT call to server - type error
500: Error occurred

POST to /login - Logins the user

Body of request:
{
"username": string

“password”: string
}
Session of the request:
{
“user” ? : User
}
Status codes:
220: User logged in
280: Unauthorized - Wrong username or password
400: Bad PUT call to server - type error
500: Error occurred

POST to /logout - Logouts the user
Session of the request:
{
“user” ? : User
}
Status codes:
221: User logged out
287: Unable to logout
500: Error occurred

GET to /loggedInUser - Gets the currently logged in user
Session of the request:
{
“user” ? : User
}
Status codes:
215: Returns user object of the logged in user
282: No user logged in
500: Error occurred

Request to /product:

GET - Gets a list of all products.

Session of the request:

{

“user” ? : User

}

Status codes:

200: List of all products returned

500: Error occurred

POST - Creates a new product and returns it.

Body of request:

{

“productName” : string

“productCategory” : string

“price” : number

“sellerId” : User.id : number

“buyerId” : User.id : number | undefined

}

Session of request:

{

“user” ?: User

}

Status codes:

222: Product created

281: User with sellerId does not exist

400: Bad POST call to server - type error

280: Unauthorized - No user logged in

500: Error occurred

DELETE - Deletes a product

Body of request:

{

“key” : number

}

Status codes:

223: Successful request - Product deleted

400: Bad DELETE call - Type error

500: Error occurred

GET to /available - Gets all available products for sale

Status codes:

224: List of all available products for sale returned

500: Error occurred

PUT /buy - Set the buyer-field to a user.

Body of request:

```
{
    “key” : number
}
```

Session of request:

```
{
    “user” ? : User
}
```

Status codes:

225: Product bought
283: Product does not exist
400: Bad PUT call to server - Type Error
282: User does not exist
500: Error occurred

PUT /update - Updates the fields of a product.

Body of request:

```
{
    “key” : number
    “productName” : string | undefined
    "productDescription: string | undefined
    “productCategory” : string | undefined
    “price” : number | undefined
}
```

Session:

```
{
    “user”: User | undefined
}
```

Status codes:

226: Product updated
400: Bad PUT call to server - type error
283: No product with id exists
500: Error occurred

GET to /sellerListings - Gets a list of all products sold by the user

Session of request:

```
{
    “user” ? : User
}
```

Status codes:

227: List of all products sold by user returned
280: Unauthorized - User not logged in
500: Error occurred

GET to /boughtProducts

Session of request:

{

“user” ? : User

}

Status codes:

228: List of all products sold by user returned

280: Unauthorized - User not logged in

500: Error occurred

GET /filterProducts - Gets a list with all products that have been filtered.

Body of request:

{

“categoriesArray” : string[]

}

Status codes:

229: Successful request - List of all filtered products returned

500: Error occurred

PUT /filteredProducts/addCat - Adds a category to the filter

Body of request:

{

“productCategory” : string

}

Status codes:

230: Category removed from filter

284: Category could not be added

400: Bad PUT call to server - type error

500: Error occurred

PUT /filteredProducts/removeCat - removes a category to the filter

Body of request:

{

“productCategory” : string

}

Status codes:

231: Category removed from filter

285: Category could not be removed

400: Bad PUT call to server - type error

500: Error occurred

Request to /cart:

GET - Get all products in the cart

Session of request:

{

“user” ? : User

}

Status codes:

232: Gets all products in cart

283: No content - No items in cart

280: Unauthorized - No user logged in

500: Error occurred

POST - Add product to cart

Body of request:

{

“product” : Product

}

Session of request:

{

“user” ? : User

}

Status codes:

233: Product successfully added

280: Unauthorized : User not logged in

500: Error occurred

DELETE - Removes product from cart

Body of request:

{

“product” : Product

Session of request:

{

“user” ? : User

}

Status codes:

234: Product successfully added

283: Product does not exist in cart

280: Unauthorized : User not logged in

500: Error occurred

DELETE to /empty - Empties the cart

Session of request:

{

“user” ? : User

}

Status codes:

235: Cart has been emptied

286: Could not find cart

280: Unauthorized : User not logged in

500: Error occurred

5. Responsibilities

The project has had all members contributing to all parts of the project. We tried to let each member have some sort of contribution with the preparation and mockup, the frontend and the backend.

For the preparation and mockup: all of us created our own version of a mockup and later on decided which and what part to implement from each mockup. Therefore the design of the application is a mix of our mockup.

For the frontend: Similar to the previous we all had some kind of responsibility. Liam took the responsibility for creating the index page, header and footer, and since the index page was the first page it was also the base that set the structure for all other pages. Later on Marcus and Rasmus took responsibility for the product page where Rasmus built the container for all the product listings and Marcus built the container for filter functionality, sell modal and also login- and registration form.
During the course we also had to refactor the code from HTML + CSS to React. This responsibility was given to all of us to handle each page that we were responsible for.

With more functionalities added we also had to create new pages. A quick summary of each page every member were responsible for:
Liam: Index page, Header, Footer, CartPage
Marcus: LoginForm, RegisterForm, SellPage, ProductPage, Product
Rasmus: ProductPage, ProfilePage, AboutUsPage, AddedProduct, Product,

For the backend: All of us have contributed to the backend. We all had a meeting where we decided which entities were needed for the project which resulted in User and Product.

Rasmus and Liam were responsible for the first implementation of User meanwhile Marcus was responsible for Product. But later on every person had to do their own implementations of methods in both of these files when creating a function that needed new router and service functions.

Rasmus later also implemented a cart entity to handle the cart functionality to make it easier to handle the cart.

For testing: During the testing phase we all have been responsible for different parts of testing. Liam was responsible for the unit test of Product, Rasmus was responsible for the unit tests of User, and Marcus created an integration test.

Later on Liam had the responsibility for testing of: Cart.
Marcus took care of tests for: LoginForm, RegisterForm, SellForm
Rasmus took care of tests for: ProfilePage and Header

Special props: During the project all members had different types of contribution that has to be commended.

Rasmus: Rasmus took responsibility for the refactoring of how our routing worked. Having a previous implementation of everything sending handlers for each state, Rasmus refactored this into our routing using a Page state instead. Rasmus also was responsible for the session implementation so we can have several users interacting simultaneously.

Marcus: Marcus took responsibility for creating functionality such as filtering the products, login- and registration, unit tests and a sort of a project leader role handing out tasks, and making sure there was continuous documentation of the API during the project.

Liam: Liam took responsibility for creating the first page (index page). He also implemented checks for all inputs, including the login-, register and sell page. Liam was also contributing to the refactor into React.js from HTML + CSS.