Group 23
Hans Erling Klevstad, s341872
Brian Banne, s329333

Documentation E-Comm Portfolio 2

Get started

.env setup

In order for the backend to work properly we have stored some environment variables and secrets in a .env file. These variables are crucial for the application to work. This file will be included in the .zip, but just in case, they are also provided below:

REACT_APP_API_URL=https://localhost:8080

GOOGLE_AUTH_SECRET=RLpGkNm2obN95ekSJTOW08Lv
GOOGLE_CLIENT_ID=910772755966-9jn7nj6bdf9bu6t80416etmeli7k3nmb.apps.googleusercontent.com
GOOGLE_REDIRECT_URL=https://localhost:8080/api/auth/login/google/callback

MONGODB_URL=mongodb://localhost:27017/ecomm
JWT_ACCESS_SECRET=SCBiplNPqNCBs3ysVG5IfYsvHy7Yr90I

ADMIN_EMAIL=admin@admin.com
ADMIN_PASSWORD=admin123

SERVER_PORT=8080
MONGODB_URL=mongodb://db:27017/ecomm

Docker Compose

In order to get the application up and running, it should be sufficient to run the command `docker-compose up` from the project root-folder. Docker Compose will then automatically fetch the necessary images and build them.

The frontend will be running on http://localhost:4000, while the backend is running on https://localhost:8080. As we are running express as an https server it is crucial to include https when accessing the server, since there is not a http server listening at that port. Although, we have included a http-server that listens to port 80, and redirects all traffic to https, but since all layers of the application are running on the same domain, this has no practical use in the scope of this project.

Group 23
Hans Erling Klevstad, s341872
Brian Banne, s329333

Documentation

Github

https://www.github.com/hanseklev/portfolie2
Our repo for this project!

API

The swagger-docs for the API is located at https://localhost:8080/docs.
The docs is generated by the package “swagger-autogen” which tracks down existing routes and generates a json from the already implemented routes.

Stack

The stack is a good ol’ MERN-stack with MongoDB, Express, React and Node.js

Frontend

For the frontend we have used the Javascript framework React to create the client interface, alongside with the *axios* module for http-requests to the backend.

Backend/server/db

The server is running on Node.js with express.
We have used mongoose to create database models and handle database-actions

Prometheus

Prometheus is scraping metrics of the http://localhost:80/metrics route. This metrics is served by the middleware “express-prom-bundle” which tracks all the activity on the server. The reason this is running on 80-port is because Prometheus by default scrapes http when on localhost, so therefore we had to expose metrics on the 80-port which then redirects to the https.
Access the tracking software at https://localhost:9100.
Visit https://localhost:3000 for graphs with grafana.

Authentication

The client login utilizes OAuth2 for the login flow. This is a more secure solution since it does not require the user to provide us directly with delicate credentials as passwords.

Group 23
Hans Erling Klevstad, s341872
Brian Banne, s329333
For the admin login, for practical reasons, there is a more conventional solution with the good old email/password combination. These credentials are located in the .env file.

How to use our site!

Depending if you're an admin or a regular user there will be different things you can do.

As a user:
- You don’t need to be logged in to make an order, but it will be easier overall.
- Log in with google authentication 2, for a more secure shopping experience.
- Access your own order page to see all your listed and completed orders.
- Add products to your shopping cart, proceed to checkout to get them to your listed address.
- In the future, apply in competitions and possible discounts.

As an admin:
- You can access all products, add new ones, edit them, or delete if necessary.
(make sure you fill the form correctly and no negative prices :))
- View all orders made by customers.

Notes

Cart

Cart state is kept client-side to reduce API-request and create a better UX, a potential downside is that a product may be out of stock due to another customer buying the last item while browsing the site. But in the scope of this assignment, this should not be an issue.

Known issues // enhancements for the future

- May have to refresh some pages for the content to load properly, especially for the order pages, as both a user and admin.
- Prometheus may need some extra implementations, but we got the basics.
- The alert for the discount may trigger more times than expected if you are really fast with ordering and checking out.
- The client-side handling for expired google-tokens is for the moment not ideal, ideally the user would be logged out. This may throw an error when rebuilding the app and requesting user orders whilst still logged in. A temporary solution is to simply log in and out.
- Due to limited resources, testing was severely impacted by budget cuts. Works fine on my machine ¯_(ツ)_/¯.
- DB stock quantity does not decrease when purchasing an item or adding to cart, this will be featured in a future release.